Research Directions: Cyber-Physical Systems

www.cambridge.org/cbp

Results

Cite this article: Lu P, Cleaveland M, Sokolsky O, Lee I, and Ruchkin I (2025). Repairing neural network-based control policies with safety preservation. *Research Directions: Cyber-Physical Systems*. **3**, e5, 1–18. https://doi.org/10.1017/cbp.2025.10003

Received: 11 September 2024
Revised: 2 August 2025
Accepted: 16 September 2025

Keywords:
Neural network repair; learning-enabled control policies; control policy repair

Corresponding author:
Pengyuan Lu; Email: pelu@seas.upenn.edu

Research Directions

CAMBRIDGE UNIVERSITY PRESS

Repairing neural network-based control policies with safety preservation

Pengyuan Lu[1], Matthew Cleaveland[1], Oleg Sokolsky[1], Insup Lee[1] and Ivan Ruchkin[2]

[1]University of Pennsylvania, Philadelphia, USA and [2]University of Florida, Gainesville, USA

Abstract

Neural network (NN)-based control policies have proven their advantages in cyber-physical systems (CPS). When an NN-based policy fails to fulfill a formal specification, engineers leverage NN repair algorithms to fix its behaviors. However, such repair techniques risk breaking the existing correct behaviors, losing not only correctness but also verifiability of initial state subsets. That is, the repair may introduce new risks, previously unaccounted for. In response, we formalize the problem of Repair with Preservation (RwP) and develop Incremental Simulated Annealing Repair (ISAR). ISAR is an NN repair algorithm that aims to preserve correctness and verifiability—while repairing as many failures as possible. Our algorithm leverages simulated annealing on a barriered energy function to safeguard the already-correct initial states while repairing as many additional ones as possible. Moreover, formal verification is utilized to guarantee the repair results. ISAR is compared to a reviewed set of state-of-the-art algorithms, including (1) reinforcement learning-based techniques (STLGym and F-MDP), (2) supervised learning-based techniques (MIQP and minimally deviating repair) and (3) online shielding techniques (tube MPC shielding). Upon evaluation on two standard benchmarks, OpenAI Gym mountain car and an unmanned underwater vehicle, ISAR not only preserves correct behaviors from previously verified initial state regions, but also repairs 81.4% and 23.5% of broken state spaces in the two benchmarks. Moreover, the signal temporal logic (STL) robustness of the ISAR-repaired policies is higher than the baselines.

Introduction

Cyber-physical systems (CPS) have been deployed in many safety-critical scenarios, including driverless vehicles, self-navigating robots and medical devices (Jazdi 2014). Starting from the 1990s, researchers have discussed the advantages of using learnable models for closed-loop control policies in CPS (Carbonell et al., 1983; Mitchell 1997; Sammut et al., 1992). Compared to optimization-based control, learning-based control moves the computational overhead from online optimization to offline training, reducing the risks caused by computational delays at run time. Moreover, learned control policies can adapt to dynamics that are complex or not explicitly known, as long as training data is available. This advantage becomes more evident as deep learning with neural networks is introduced to implement control policies. With their capabilities of approximating complex functions, neural network (NN)-based control policies have attracted significant attention in reinforcement learning (Arulkumaran et al., 2017; Kaelbling et al., 1996; Wiering and Van Otterlo 2012) and learning from demonstrations (Argall et al., 2009; Ravichandar et al., 2020; Schaal 1996).

However, NN-based control policies have a major drawback: they lack formal guarantees of safety. Research in formal methods studies how to rigorously prove that certain specifications (such as safety) are met in autonomy. The specifications are usually in temporal logics, such as linear temporal logic (LTL) (Pnueli 1977) signal temporal logic (STL) (Maler and Nickovic 2004), which formalizes properties of continuous signals over time. The proofs are commonly done in the process of verification (Hasan and Tahar 2015). Without such proofs, counterexamples of a specification may exist and pose a severe threat to safety-critical CPS. For instance, a driverless car could collide with a pedestrian, or an insulin admin device could overdose a diabetic patient. Unfortunately, it is not only difficult to establish safety specifications but also prove that they are met by NN-based control policies. This issue is a part of a more general question: How to ensure safety in learning-enabled CPS? (Nicola Paoletti and Jim Woodcock 2023)

Although it is hard to construct safety-guaranteed NN-based policies, an easier target is to remove unsafe counterexamples due to the NN parameters, a procedure known *NN repair* (Cohen and Strichman 2022; Fu and Li 2021; Fu et al., 2022; Lu et al., 2023; Lyu et al., 2023; Sohn et al., 2019; Sotoudeh and Thakur 2021; Tokui et al., 2022; Usman et al., 2021). We identify three major categories of repair algorithms for NN-based control policies: (1) Offline, without any

demonstration data, these learned policies can be fine-tuned with the guidance of formal specifications, using reinforcement learning (Hamilton et al., 2022; Venkataraman et al., 2020); (2) When demonstrations are available, say from a safe model predictive control (MPC), they can serve as a supervision signal, and the repair is thus done via imitation learning (Majd et al., 2023; Majd et al., 2021; Zhou et al., 2020); (3) At run time, an additional layer of operation, known as shielding, can be appended at the end of the policy network to intervene when the original NN output fails to accomplish the task (Wabersich and Zeilinger 2018). The details of these categories are discussed in Section 2.3.

Nevertheless, state-of-the-art NN repair algorithms have yet to consider one important feature: the preservation of correct behaviors. Specifically, the repair procedure may compromise the existing behaviors that satisfy the task specification. For example, a healthcare service robot (Holland et al., 2021) can successfully attend to its patient upon alarms, except for starting in the kitchen. After the repair, although it is now able to attend to the patient in the kitchen, the robot drives unsafely inside the patient's bedroom, posing a risk to the patient's well-being—which did not happen before and was introduced by the repair. Unfortunately, none of the research on NN-based control policy repair has addressed this issue of correctness preservation. By "preservation" we mean that a correct behavior guaranteed by a sound verifier *must remain guaranteed* after repair. Losing the guarantee would introduce new and untested behaviors into the system.

A naive approach to repair with preservation is divide-and-conquer: incrementally repair the incorrect behaviors one-by-one. Unfortunately, this procedure faces the challenge of *catastrophic forgetting*, a phenomenon widely studied by researchers in multi-task and continual learning (Liu 2017; Ruvolo and Eaton 2013; Thrun 1998; Van de Ven and Tolias, 2018). That is, fine-tuning the parameters of a learning model for a second objective would lower the performance in the first one—a nature of multi-objective optimization (Kendall et al., 2018). In the case of NN repair, preserving old behaviors while acquiring correctness on new ones also suffers from this challenge, as all behaviors are produced by a shared learned representation. Therefore, modifying the shared representation would affect the outcome of all behaviors, including the ones that are previously correct.

To overcome the challenge above, we formalize the problem as *Repair with Preservation (RwP)*. Specifically, the problem asks for an alternative control policy NN that (i) still accomplishes the task with guarantees from previously verified initial states and (ii) maximizes the number of additional repaired initial states. As a solution to RwP, we propose Incremental Simulated Annealing Repair (ISAR), which safeguards the previously successful initial states during repair and generates the repaired policy together with verification results. This algorithm partitions the initial state set into finitely many regions. Each region first passes through a sound (passing verification guarantees correctness) but incomplete (not passing does not necessarily mean incorrectness exists) verifier, such as Verisig (Ivanov et al., 2021; Ivanov et al., 2019), to formally prove whether trajectories from this entire region are able to accomplish the task. During repair, simulated annealing (Kirkpatrick et al., 1983) is applied to a barrier-guarded energy function (Hertog et al., 1992; Hauser and Saccon 2006; Polyak 1992) on STL robustness (Fainekos and Pappas 2009; Maler and Nickovic 2004) to greedily repair the initial state regions one by one, while protecting the previously successful regions. After the repair, the new control policy will be checked by the verifier again to produce formal guarantees.

We evaluate ISAR on two case studies of safety-critical CPS: an unmanned underwater vehicle (UUV) (Ruchkin et al., 2022) and OpenAI Gym mountain car (MC) (Brockman et al., 2016). In these case studies, ISAR is compared against state-of-the-art baselines, including (1) offline fine-tuning guided by reinforcement learning with rewards generated by formal specification (Hamilton et al., 2022; Venkataraman et al., 2020), (2) offline fine-tuning guided by supervised learning (Majd et al., 2021; Zhou et al., 2020) and (3) online intervention by tube MPC shielding, which uses MPC to replace a control policy's action when the current action is estimated to generate an unsafe trajectory (Wabersich and Zeilinger 2018). The results show that ISAR is the only method that completely preserves the verification guarantees while repairing additional incorrect behaviors. Quantitatively, ISAR preserves all verified initial state regions during the repair, breaking none of them, while repairing 81.4% and 23.5% of the failed regions in UUV and MC, respectively. It also results in the highest minimal STL robustness per region, meaning that the successful regions are the furthest from being broken, and the failed regions are the closest to being repaired.

This journal article provides one approach to answer the question on improving safety in learning-enabled CPS (Nicola Paoletti and Jim Woodcock 2023). It serves as an extension of a conference paper (Lu et al., 2024), which makes the following contributions:

1. A formalization of the Repair with Preservation (RwP) problem, which calls for preserving verifiable correctness during the repair of NN-based control policies in safety-critical CPS.
2. ISAR algorithm, which leverages STL robustness of trajectories as an energy function in simulated annealing, to safeguard correct behaviors during repair.

This extension paper makes the following *additional contributions*:

1. A taxonomy of approaches to NN-based control policy repair, supported by a detailed review of the state-of-the-art literature.
2. An experimental comparison of ISAR with several state-of-the-art baselines on standard benchmarks. Results show that, unlike the baselines, ISAR preserves verifiable correctness while repairing erroneous behaviors.
2. An evidence-based analysis of the ISAR strengths and weaknesses compared to state-of-the-art repair techniques.

Methods

Definitions

To formalize our method, we first introduce the following definitions.

Notation and system model

Our problem setting is based on the following formalization. Let $t = 0, 1, 2, \ldots$ denote the discrete time steps, S be a continuous physical state space and A be a continuous action space.

We adopt a standard notation of a system from the literature on dynamical systems and reinforcement learning (Wiering and Van Otterlo 2012). Let π: $S \mapsto A$ denote a deterministic control policy; i.e., the current action is $a_t = \pi(s_t)$. The control policy π_θ is parameterized by some $\theta \in \Theta$, with Θ being a fixed parameter space such as the weights and biases for some neural network architecture. The environment dynamics is f: $S \times A \mapsto S$, i.e., the next state $s_{t+1} = f(s_t, a_t)$.

Signal temporal logic

Our problem is also based on STL (Maler and Nickovic 2004), a logical formalism to specify properties of continuous signals, such as trajectories of physical states in a continuous state space. In CPS, STL is used for both monitoring and verification of task accomplishment (Ruchkin et al., 2022). On a trajectory denoted as $\bar{s} := s_0 s_1 \dots s_T$, $\forall s_t \in S$, the grammar of STL is defined as

$$\varphi ::= \top \mid g(\bar{s}) < 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 U_{[t_1,t_2]} \varphi_2. \tag{1}$$

Here, $\top$ is a tautology and g: $S^T \mapsto \mathbb{R}$ is a real-valued function on signals. Temporal operator $U_{[t_1,t_2]}$ denotes the until operator, and $\phi_1 U_{[t_1,t_2]} \phi_2$ means ϕ_2 must hold at some time $t \in [t_1,t_2]$ and ϕ_1 must always hold before t. The until operator can be converted into two additional temporal operators (i) eventually/finally operator $F_{[t_1,t_2]}$ and (ii) always/globally operator $G_{[t_1,t_2]}$. The exact definition of STL semantics can be found in the original paper (Maler and Nickovic 2004).

STL robustness

We adopt STL for two reasons. First, it is more expressive than other temporal logic languages such as LTL, as STL leverages real-valued predicates to express the system's physical state. Second, STL robustness can be used as a soft objective function to guide the learning process (Fainekos and Pappas 2009).

Conventional STL semantics produce a Boolean outcome, indicating whether a given trajectory accomplishes its task. STL is also equipped with quantitative semantics (Fainekos and Pappas 2009; Gilpin et al., 2020; Hamilton et al., 2022), a robustness score to evaluate the degree of a trajectory satisfying an STL formula ϕ. Specifically, the robustness score ρ: $S^T \times \mathbb{N}_{\geq 0} \times \Phi \mapsto \mathbb{R}$ is a real-valued function that evaluates $\rho(\bar{s}, t, \varphi)$ on three inputs: a trajectory $\bar{s}$ of length T, a starting time step t of the trajectory and an STL formula $\phi \in \Phi$. The reader is referred to classic sources (Fainekos and Pappas 2009) for its definition. Based on this definition, $\rho \geq 0$ means that the trajectory starts at time t with length T is able to fulfill the STL formula ϕ, i.e., it accomplishes the specified task. Likewise, $\rho < 0$ means it fails. The larger the absolute value of ρ, the higher the degree of accomplishment or failure.

We assume all tasks are given in the form of STL specifications, and the following subroutine is available to compute performance on a task ϕ from an initial state s_0, under dynamics f and policy π.

Definition 1 (Robustness Computing Subroutine). *Given a dynamics f and an STL-specified task ϕ, a robustness computing subroutine $rob_{f,\phi} : S \times \Pi \mapsto \mathbb{R}$ is a function that computes the STL robustness of a trajectory from an initial state s_0 under a control policy π. That is,*

$$\mathrm{rob}_{f,\varphi}(s_0, \pi) := \rho(\bar{s}, 0, \varphi) \tag{2}$$

Here, trajectory $\bar{s} = s_0 s_1 \cdots s_T$, with $a_t = \pi(s_t)$ and $s_{t+1} = f(s_t, a_t)$ for all $t \in [0, T)$. The set Π denotes the space of all policies.

In other words, $\mathrm{rob}_{f,\phi}(s_0,\pi)$ evaluates the degree of task accomplishment from initial state s_0 under control policy π.

Sound but incomplete verifier

Our problem setting also relies on verifiers (Ivanov et al., 2019). A verifier formally proves whether a continuous subset of initial states $S' \subset S$ produces trajectories that all satisfy a specification ϕ under a policy π and dynamics f. Formally, a mathematical abstraction of a verifier is given as follows.

Definition 2 (Verifier). *With a given dynamics f and an STL specification ϕ, a verifier is a mapping $ver_{f,\phi} : 2^{S_{init}} \mapsto \{0, 1\}$ that checks whether the system starting from any state in a subset of initial states $\mathcal{S} \subseteq \mathcal{S}_{init}$ is able to satisfy ϕ under f.*

In plain words, if we input a subset of initial states into a verifier, it returns 1 if all initial states will end up satisfying ϕ in f, or 0 if there exists an initial state that will end up violating ϕ in f.

A sub-category of verifiers is sound but incomplete verifiers, which we are able to obtain from off-the-shelf techniques, such as Verisig (Ivanov et al., 2021; Ivanov et al., 2019). Such verifiers are conservative. That is, if a region of initial states passes this verifier, it is guaranteed that all states in this region can successfully accomplish the task (i.e., the verifier is sound). However, the implication does not hold in the other direction (i.e., the verifier is incomplete). To define such a verifier, we first define a partition in the initial states as follows.

Definition 3 (Initial State Partition). *Given dynamics f: $S \times A \mapsto S$, a control policy π: $S \mapsto A$ partitions the initial states $S_{init} \subset S$ into successful and failed ones: $S_{init} = S_\pi^s \cup S_\pi^f$. That is,*

$$\begin{aligned} S_\pi^{\mathrm{s}} &:= \{s_0 \in S_{init} : \mathrm{rob}_{f,\varphi}(s_0, \pi) \geq 0\} \\ S_\pi^{\mathrm{f}} &:= S_{init} \backslash S_\pi^{\mathrm{s}} \end{aligned} \tag{3}$$

Definition 3 formalizes a partition of the initial states under a control policy π. In plain words, S_π^{s} is the set of initial states from which π satisfies ϕ ("s" stands for "success"). Likewise, S_π^{f} is the set of initial states from which π does not satisfy, or equivalently, fails on ϕ ("f" stands for "failure"). With the formalization above, we identify a feasible repair problem that takes account of STL robustness trade-offs among trajectories from different initial states.

With the above partition defined, we have Definition 4.

Definition 4 (Sound but Incomplete Verifier). *Given dynamics f, a verifier of property ϕ is abstracted as a function $ver_{f,\phi} : 2^S \times \Pi \mapsto \{0, 1\}$. On a subset of initial states $S' \subseteq S$ for a given control policy π, the verifier outputs whether all initial states in S' produce trajectories that satisfy ϕ. A verifier is sound but incomplete iff*

$$\begin{aligned} &(\forall S' \in 2^S \cdot \mathrm{ver}_{f,\varphi}(S', \pi) = 1 \Longrightarrow S' \subseteq S_\pi^{\mathrm{s}}) \wedge \\ &(\exists S'' \in 2^S \cdot S'' \subseteq S_\pi^{\mathrm{s}} \wedge \mathrm{ver}_{f,\varphi}(S'', \pi) = 0). \end{aligned} \tag{4}$$

In Equation (4), the term on the first line means soundness, and the second line means incompleteness.

Simulated annealing

Simulated annealing is a stochastic optimization algorithm that aims to find the global optima of an objective function, which is often equivalently referred to as an *energy function* (Kirkpatrick et al., 1983); Serafini 1994; Suman and Kumar 2006). We denote the energy function as e: $\Theta \mapsto \mathbb{R}$, i.e., a real-valued function that depends on variables $\theta \in \Theta$.

The algorithm works as follows. At every iteration, it randomly perturbs previous variables θ to obtain new variables θ', e.g., by adding Gaussian noise to each variable. Then, the energy function is evaluated to see if the new $e(\theta')$ is better than the previous $e(\theta)$. Specifically, if $\Delta_e := e(\theta') - e(\theta) \geq 0$, the new variables are accepted. Otherwise, if $\Delta_e < 0$, a second check is performed by tossing a biased coin with acceptance probability

$$\Pr[\text{acceptance}] := \exp(-\Delta_e/\tau). \tag{5}$$

Here, $\tau > 0$ is called the temperature, with higher values leading to more exploration. This second criterion is formally known as the Metropolis-Hastings criterion, which encourages the exploration of new variables, even if the new energy function is slightly worse. At the end of every iteration, the temperature cools down, e.g., by multiplying a cooling factor $\alpha \in (0,1)$. So the amount of exploration decreases throughout the procedure. The algorithm runs until either the energy function converges or a maximum number of iterations is reached.

Compared to other optimization algorithms, such as gradient descent, simulated annealing is able to escape local optima due to its stochastic nature and encouragement in exploration. We will use a modified version of this technique in our main algorithm.

Problem: repair with preservation (RwP)

Next, we formalize the problem to be solved based on the above definitions. To formulate the problem, we first define a partition of the initial states S_{init} under a control policy π.

Formulating the RwP problem

Based on the initial states partition defined in Definition 3, we formulate the RwP problem as follows.

Definition 5 (Repair with Preservation (RwP) Problem).

$$\begin{aligned} &\text{maximize}_{\pi'} \int_{S_\pi^{\mathrm{f}}} \mathbb{1}(\text{rob}_{f,\varphi}(s_0,\pi') \geq 0) ds_0 \\ &\text{subject to } S_\pi^{\mathrm{s}} \subseteq S_{\pi'}^{\mathrm{s}}, \end{aligned} \tag{6}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which returns 0 if the input proposition is false and 1 if true.

Equation (6) aims to find an alternative control policy π' to repair as many previously failed initial states in $S_\pi{}^{\mathrm{f}}$ as possible, while protecting the previously successful initial states $S_\pi{}^{\mathrm{s}}$ from being compromised. Trivially, a control policy that satisfies the constraint is the original policy π. Therefore, it is guaranteed that there exists a solution to this problem.

To solve for the non-standard integral in (6), we approximate the problem by partitioning the initial state set into finitely many regions, i.e., $S_{init} = S_1 \cup \cdots \cup S_M$. There exist two types of initial state regions S_i: (i) successful: all states in the region are successful under a given policy π, i.e., $S_i \subseteq S_\pi{}^{\mathrm{s}}$ and (ii) failed: not all states in the region are successful under π, i.e., $S_i \subseteq S_\pi^{\mathrm{s}}$. The problem becomes

$$\begin{aligned} &\text{maximize}_{\pi'} \sum_{S_i \not\subset S_\pi^{\mathrm{s}}} \mathbb{1}\left(S_i \subseteq S_{\pi'}^{\mathrm{s}}\right) \\ &\text{subject to } S_i \subseteq S_\pi^{\mathrm{s}} \Longrightarrow S_i \subseteq S_{\pi'}^{\mathrm{s}}. \end{aligned} \tag{7}$$

The repair problem in Equation (7) aims to repair as many failed regions as possible, while safeguarding the already successful ones. This discretized version of the problem approaches Equation (6) as the partition becomes finer.

Formulating the verification-aided RwP problem

To solve Equation (7), one more key issue remains: how do we know whether *all states* in a region S_i would lead to success? To provide guarantees for infinite (but bounded) regions, we leverage a sound but incomplete verifier introduced in Section 2.1.4.

With a verifier available, we aim to address the following main problem, which first requires partitioning the initial state space into regions $S_1, \ldots, S_M$. We know that a region S_i is successful if it passes verification, i.e., $\text{ver}_{f,\phi}(S_i, \pi_\theta) = 1$. We would like to protect such regions, preserving the correct behaviors. While protecting these regions, we would like to maximally repair failed regions. However, the incomplete property of the verifier means that a repaired region may still be classified as unsafe by the verifier. Consequently, we estimate a region S_i is repaired if all initial states in a uniformly sampled finite subset $\hat{S}_i \subset S_i$ can accomplish the task, i.e., STL robustness ≥ 0.

The size of regions is a design parameter. Since the verifier is conservative, the larger regions we have, the less likely we are able to capture the small subsets of initial states that are correct. However, larger regions would also lead to smaller numbers of regions, providing computational efficiency. Indeed, picking the sizes of regions exhibits an accuracy-efficiency trade-off. For simplicity, we assume the partition on regions is already given. Then the main problem is formalized as follows.

Main Problem (Verification-aided RwP). We would like to accomplish a given STL task ϕ under dynamics $f: S \times A \mapsto S$. The current control policy $\pi: S \mapsto A$ is unable to accomplish this task from all initial states in $S_{init} \subset S$. Therefore, we aim to find an alternative control policy π' by repairing a subset of initial states while safeguarding another. Specifically, we are given a partition $S_{init} = S_1 \cup \cdots \cup S_M$ of the initial state space. Moreover, for each region S_i, K initial states are uniformly sampled, forming a finite subset $\hat{S}_i$.

Assuming an STL robustness function $\text{rob}_{f,\phi}$ as in Definition 1, the regions to be repaired are

$$\mathcal{S}_\pi^{\mathrm{f}} := \left\{S_i : \exists s_0 \in \hat{S}_i, \text{rob}_{f,\varphi}(s_0,\pi) < 0\right\}, \tag{8}$$

The regions to be protected are the ones passing a sound but incomplete verifier $\text{ver}_{f,\phi}$ as in Definition 4, i.e.,

$$\mathcal{S}_\pi^{\mathrm{s}} := \{S_i : \text{ver}_{f,\varphi}(S_i,\pi) = 1\}. \tag{9}$$

To find an alternative control policy π', we solve the following optimization problem[1].

$$\begin{aligned} &\text{maximize}_{\pi'} \sum_{S_i \in \mathcal{S}_\pi^{\mathrm{f}}} \mathbb{1}\left(\forall s_0 \in \hat{S}_i, \text{rob}_{f,\varphi}(s_0,\pi') \geq 0\right) \\ &\text{subject to } S_i \in \mathcal{S}_\pi^{\mathrm{s}} \Longrightarrow \text{ver}_{f,\varphi}(S_i,\pi') = 1. \end{aligned} \tag{10}$$

That is, our goal is to design a repair algorithm that maximizes the number of repaired initial state regions and safeguards the already successful ones. The verifier provides a formal proof of constraint satisfaction for a solution control policy. **In other words, preserving the verified (guaranteed) correctness in the initial state regions is the top priority, i.e., constraint, while repairing additional incorrect regions is the second priority.**

One remark is that this main problem seeks a single new control policy π'. There exists an alternative problem setting, which aims to obtain one policy per region S_i and switch between these policies at runtime. However, researchers in learned control policies have already discussed the disadvantages of retaining distinct per-task policies instead of using a shared policy across tasks. That is, the latter would provide higher efficiency in computation and resources (Hessel et al., 2019; Parisotto et al., 2016); Rusu et al., 2016). In contrast, researchers in multi-task and continual learning have already shown satisfactory performance in using one shared model for multiple sub-tasks (Ruvolo and Eaton 2013). Therefore, our research targets the solution of a shared control policy

representation. It would be interesting for future work to explore the alternatives.

Baseline methods of NN-based policy repair

We identify candidate solutions to the main problem in the literature. Here, we review three categories of NN-based control policy repair methods. These state-of-the-art techniques also serve as baselines in our experiments. Notice that all these baseline methods are not designed to preserve guaranteed correctness, and therefore, we call for a new method.

Offline fine-tuning by formal specifications

When demonstrations of correct behaviors are not available, repairing NN-based policies may be guided by formal specifications. Existing literature constrains the repair problem with additional assumptions. One such assumption is the finite number of counterexamples, which can be searched and repaired iteratively (Bauer-Marquart et al., 2022). If counterexamples are finite and the NN is piecewise linear, such as ReLU-activated networks, existing methods can provide various guarantees. These guarantees include locality in repair, i.e., altering the behavior of a specific input would only affect the behaviors of a small surrounding neighborhood (Fu and Li 2021; Majd et al., 2021), as well as provable minimality in modification (Goldberger et al., 2020). For infinite counterexamples, a typical assumption is that a forward pass of the NN can be approximated as affine mappings with bounded errors, so that reachability analysis would apply (Yang et al., 2022). These methods fit well when the underlying NN follows these assumptions, with applications such as robotic prosthesis (Majd et al., 2023).

For general NN architectures and infinite counterexamples, researchers have leveraged reinforcement learning (RL) (Arulkumaran et al., 2017; Kaelbling et al., 1996; Wiering Van Otterlo 2012) to fine-tune the policy. The assumption is that the environment, or a simulator of the environment, is available for the controlled agent to explore and collect data by itself.

By definition of RL, the underlying dynamics are modeled as a Markov decision process (MDP), and the goal is to find a policy that maximizes the total reward of a given reward function on this MDP. However, the major concern is how to incorporate the specification in this setting. In real-world applications such as CPS, specifications are commonly formalized as temporal logic formulas, which are non-Markovian. That is, how good or how bad an action is not only depends on the current state but also on the history. Therefore, if a reward is designed to measure the degree of satisfying a temporal logic specification (such as STL robustness), a memory cache has to be used to remember the previous states. This dependency on history complicates learning.

When the specification is a LTL (Pnueli 1977), this issue of being non-Markovian shall be resolved by product MDP (Bozkurt et al., 2020; Ding et al., 2014; Fritz 2003; Mukund 1997; Sadigh et al., 2014; Sickert et al., 2016; Vardi 2005). That is, the underlying MDP model is augmented with an additional finite automaton, which is equivalent to the LTL. Then, all learning is done on this augmented MDP, without an additional memory cache needed. However, when the specification is STL, the equivalent automaton, namely τ-MDP, is proven to be intractable (Aksaray et al., 2016). So far, no tractable equivalent automaton can be found (Sickert 2015), and scientists therefore find alternative solutions. One example is to identify an approximate automaton instead of an equivalent one. Here, researchers augment the MDP with an automaton known as F-MDP (Venkataraman et al., 2020), and STL robustness is used as the reward function. Here, the state space is augmented and no additional memory cache is required. Still, designing the automaton for F-MDP is a non-trivial problem. Alternatively, a more straightforward solution, known as STLGym (Hamilton et al., 2022), is to keep an additional memory cache and compute the non-Markovian STL robustness as a reward. The memory is constantly monitored and updated during training time.

Offline fine-tuning by imitation learning

When demonstrations of correct behaviors are available, offline fine-tuning of the NN-based policy can be done by supervised learning, also known as imitation learning (Hussein et al., 2017).

Researchers have formulated this problem as a mixed integer quadratic program (MIQP) (Majd et al., 2021). In this paper, the demonstrator is a synthesized safe model predictive control (MPC), and the solution is to fine-tune a given layer of parameters in an NN, such that the outputs of the NN will fall within a specific safety range. The MIQP adjusts the given layer's weights to minimize the same MSE loss, while subject to the safety constraint on outputs. This approach has been applied to the repair of NN-controlled prosthesis (Majd et al., 2023), where the output control signal must be restrained within a range to avoid drastic motions of the prosthetic limbs.

An alternative approach is to conduct supervised learning on the entire NN instead of one selected layer. Known as minimally deviating repair (Zhou et al., 2020), this method trains an NN-based policy to imitate a safe MPC by fine-tuning all its layers. The objective is to minimize the deviation in NN parameters while repairing the errors, in the hope that the previous correctness shall be preserved. Still, small deviations in NN parameters do not necessarily imply that previously correct behaviors will not be broken. In addition to data to imitate, context knowledge can be provided by formal specifications, leading to the hybrid information setting for repair (Zhou et al., 2023).

Online intervention by shielding

Different from the above two categories, another method is to add an additional operation at the end of the NN-based policy at runtime. This additional operation is known as shielding, which identifies faulty NN outputs and intervenes (Bastani 2021; Li and Bastani 2020; Wabersich et al., 2021). The identification is commonly rollout based, i.e., the shield assumes a known dynamics, and conducts reachability analysis to see if the specification could possibly be violated. A typical state-of-the-art rollout-based shield is tube MPC (Wabersich and Zeilinger 2018). At every time step t, a safety check is done by computing whether a fallback policy, synthesized by MPC, can maintain safety within the horizon from $t+1$, if the current control action is executed. If not, the algorithm intervenes by switching to the fallback policy, whose safety is ideally guaranteed at the previous time step $t-1$. Still, the risk of this method comes from long horizons and complex dynamics, which lead to errors in reachability analysis and MPC optimization. Under such a risk, the shield may intervene to correct the behavior, causing failures in preservation.

Incremental simulated annealing repair (ISAR)

This subsection details our method design of ISAR, illustrated in Figure 1. As shown, our approach incrementally selects one initial state subset where counterexamples reside. The NN-based policy is then repaired by simulated annealing under a particular energy

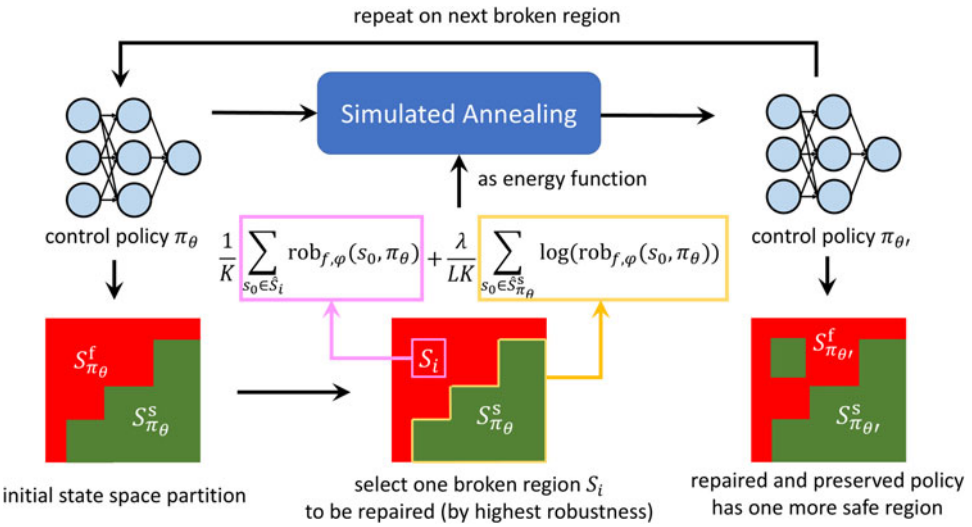

Figure 1. The workflow of incremental simulated annealing repair (ISAR).

function: a weighted sum between STL robustness of the counterexamples and the log-barriered STL robustness from the regions to be preserved.

In Section 2.4.1, we present an optimization objective function (energy function) to be maximized. This energy function aims to improve the STL robustness of one set of initial states while protecting the positive robustness of another set of initial states. In Section 2.4.2, we design a safeguarded version of simulated annealing to optimize the energy function. At the end, in Section 2.4.3, we propose the overall ISAR algorithm that uses the safeguarded simulated annealing as a subroutine to incrementally repair failed state regions.

Energy function design

We use a simulated annealing-based approach to optimize the parameters θ for an NN-based control policy π_θ, due to its ability to escape local optima.

To apply simulated annealing, our first step is to design the energy function. To repair a failed region of initial states $S_i \in \mathcal{S}_\pi^{\text{f}}$ (as defined in the main problem), we aim to improve the STL robustness of its generated trajectories. This results in the following energy function

$$e(\theta) := \frac{1}{V(S_i)}\int_{S_i} \text{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0. \tag{11}$$

Here, $V(\cdot)$ denotes a volumetric measure on a continuous set, $V(S) = \int_S ds_0$, i.e., the volume, if the set S is Euclidean ($S \subset \mathbb{R}^n$). That is, we want to maximize the average STL robustness of trajectories from all initial states $s_0 \in S_i$.

However, this energy function does not protect any currently successful initial states. Formally, we want to repair a region $S_i \in \mathcal{S}_\pi^{\text{f}}$ while safeguarding $S^{\text{s}} := \bigcup_{S\in\mathcal{S}_\pi^{\text{s}}} S$, as stated in the main problem. To do this, we employ a log-barrier function, as it is a smooth approximation of the constraint $\forall s_0 \in S^{\text{s}}$, $\text{rob}_{f,\phi}(s_0,\pi_\theta) \geq 0$ and is easy to compute (Hertog et al., 1992; Hauser and Saccon 2006; Polyak 1992). We have

$$\begin{aligned} e(\theta) := {} & \tfrac{1}{V(S_i)}\textstyle\int_{S_i} \text{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0 \\ & + \tfrac{\lambda}{V(S^s)}\textstyle\int_{S^s} \log\big(\text{rob}_{f,\varphi}(s_0,\pi_\theta)\big)ds_0 \end{aligned} \tag{12}$$

The hyperparameter $\lambda > 0$ is a balance factor. Larger values of λ favor protecting the successful states, while smaller values favor fixing the unsuccessful states. The energy function in Equation (12) aims to improve the robustness of all initial states in S_i, while safeguarding the robustness of all initial states in S^{s} with a log barrier. Note that if the STL robustness is negative, then the logarithm will be undefined. To avoid this, we set up a lower bound, e.g., -1000, that the log-robustness will take if the STL robustness is negative or if log robustness is lower than the bound.

One computational issue is that Equation (12) has no convenient closed-form solution, making it hard to evaluate directly. In response, we use Monte Carlo integration (Robert et al., 1999) to approximate the two integrals. So, we uniformly sample states from each region S_i to form finite sets $\hat{S}_i$ (as in Equation (10)). The union of all sampled states from successful regions is denoted as $\hat{S}^{\text{s}}$. The cardinality of $\hat{S}_i$ and $\hat{S}^{\text{s}}$ are K and LK (K samples per region as in the main problem, and $L = |\mathcal{S}_\pi^{\text{s}}|$ is the number of regions to protect). So the energy function now becomes

$$\begin{aligned} \hat{e}(\theta) := {} & \tfrac{1}{K}\textstyle\sum_{s_0\in\hat{S}_i} \text{rob}_{f,\varphi}(s_0,\pi_\theta) \\ & + \tfrac{\lambda}{LK}\textstyle\sum_{s_0\in\hat{S}^s} \log\big(\text{rob}_{f,\varphi}(s_0,\pi_\theta)\big). \end{aligned} \tag{13}$$

The following proposition shows that $\hat{e}(\theta)$ and $e(\theta)$ can be arbitrarily close.

Proposition 1 (Error in Monte Carlo Integrated Energy Function). The energy functions $\hat{e}(\theta)$ in Equation (13) and $e(\theta)$ *in Equation (12) satisfy*

$$\begin{aligned} & \mathbb{E}[\hat{e}(\theta)] = e(\theta) \text{and} \\ & \text{Var}[\hat{e}(\theta)] = \big(\tfrac{1}{K}\big)^2\text{Var}_{\hat{S}_i}\big[\text{rob}_{f,\varphi}\big] + \big(\tfrac{\lambda}{LK}\big)^2\text{Var}_{\hat{S}^s}\big[\log^\circ \text{rob}_{f,\varphi}\big]. \end{aligned} \tag{14}$$

Here, $\text{Var}_{\hat{X}}[g]$ denotes the variance of a function g on a finite set of inputs $\hat{X}$.

Proof. Monte Carlo integration has the following property (Robert et al., 1999): an integral $I = \int_X g(x)dx$ is approximated by

$\hat{I} = (V/K)\sum_{x\in\hat{X}} g(x)$. Here, $\hat{X}$ consists of K uniformly and independently sampled points in X and volume $V = \int_X dx$. Existing work (Robert et al., 1999) has shown this approximation satisfies (i) $\mathbb{E}[\hat{I}] = I$ and (ii) $\mathrm{Var}[\hat{I}] = V^2\mathrm{Var}_{\hat{X}}[g]/K^2$.

The above is an established result of Monte Carlo integration. In our case, the energy function is a linear combination of two Monte Carlo integrals. Also, recall that a region S_i has K uniformly sampled initial states, and the union of to-be-preserved regions $S^s = \bigcup_{S_i\in\mathcal{S}^s} S_i$ has LK uniformly sampled initial states.
We have

$$\begin{aligned}
&\mathbb{E}[\hat{e}(\theta)] = \tfrac{1}{K}\mathbb{E}\left[\sum_{s_0\in\hat{S}_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\right] + \tfrac{\lambda}{LK}\mathbb{E}\left[\sum_{s_0\in\hat{S}^s} \log\big(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\big)\right]\\
&= \tfrac{1}{V(S_i)}\underbrace{\mathbb{E}\left[\frac{V(S_i)}{K}\sum_{s_0\in\hat{S}_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\right]}_{\mathbb{E}[\hat{I}]} + \tfrac{\lambda}{V(S^s)}\underbrace{\mathbb{E}\left[\frac{V(S^s)}{LK}\sum_{s_0\in\hat{S}^s} \log\big(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\big)\right]}_{\mathbb{E}[\hat{I}]}\\
&= \tfrac{1}{V(S_i)}\int_{S_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)ds_0 + \tfrac{\lambda}{V(S^s)}\int_{S^s} \log\big(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\big)ds_0\\
&= e(\theta) \text{ by Equation (12).}
\end{aligned} \tag{15}$$

Likewise, because the initial states are independently sampled,

$$\begin{aligned}
&\mathrm{Var}[\hat{e}(\theta)] = \tfrac{1}{K^2}\mathrm{Var}\left[\sum_{s_0\in\hat{S}_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\right] + \tfrac{\lambda^2}{(LK)^2}\mathrm{Var}\left[\sum_{s_0\in\hat{S}^s} \log\big(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\big)\right]\\
&= \tfrac{1}{V(S_i)^2}\underbrace{\mathrm{Var}\left[\frac{V(S_i)}{K}\sum_{s_0\in\hat{S}_i} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\right]}_{\mathrm{Var}[\hat{I}]} + \tfrac{\lambda^2}{V(S^s)^2}\underbrace{\mathrm{Var}\left[\frac{V(S^s)}{LK}\sum_{s_0\in\hat{S}_2} \log\big(\mathrm{rob}_{f,\varphi}(s_0,\pi_\theta)\big)\right]}_{\mathrm{Var}[\hat{I}]}\\
&= \frac{1}{V(S_i)^2}\frac{V(S_i)^2\mathrm{Var}_{S_i}[\mathrm{rob}_{f,\varphi}]}{K^2} + \frac{\lambda^2}{V(S^s)^2}\frac{V(S^s)^2\mathrm{Var}_{\hat{S}^s}[\log\circ\mathrm{rob}_{f,\varphi}]}{(LK)^2}\\
&= \frac{1}{K^2}\mathrm{Var}_{S_i}[\mathrm{rob}_{f,\varphi}] + \frac{\lambda^2}{(LK)^2}\mathrm{Var}_{\hat{S}^s}[\log\circ\mathrm{rob}_{f,\varphi}].
\end{aligned} \tag{16}$$

Proposition 1 states that the Monte Carlo integrated energy function $\hat{e}(\theta)$ has an expected value of the energy function $\hat{e}$. Moreover, as we sample more initial states (larger K), the variance of $\hat{e}(\theta)$ decreases. Therefore, by increasing K, the approximation can be arbitrarily close to the original energy function. Specifically, we are able to compute confidence intervals based on the size of sampled state sets (Preacher and Selig 2012).

Safeguarded simulated annealing

The energy function in Equation (13) is a non-standard function with a potentially large number of local optima. Therefore, we utilize simulated annealing to maximize it, which has the advantage of escaping local optima as mentioned in Section 2.1.5. However, due to its stochastic nature, simulated annealing may compromise the protected initial states, lowering their trajectories' STL robustness. We tolerate such compromise unless the robustness drops below 0, which means the task is failed. Therefore, we propose the following safeguarded version of simulated annealing in Algorithm 1, which has an additional check on the STL robustness of the protected initial states.

Algorithm 1 Safeguarded Simulated Annealing

Input: Finite set of states to be repaired $\hat{S}_i$, finite set of states to be protected $\hat{S}^s$, control policy parameters θ, energy balance factor $\lambda > 0$, perturbation standard deviation $\sigma > 0$, initial temperature $\tau > 0$, cooling factor $\alpha \in (0,1)$, max iteration Q

Output: Intermediate repaired control policy parameters θ'

1: $\theta' \leftarrow \theta$ 1:
2: $\hat{e}, \rho_{\hat{S}^s} \leftarrow$ evaluate Energy($\hat{S}_i$, $\hat{S}^s$, θ', λ) 2:
3: **for** *iter* = 1, . . . , Q **do**
4: $\quad\theta'' \leftarrow \theta' + \mathcal{N}(0, \sigma^2)$
5: $\quad\hat{e}', \rho_{\hat{S}^s} \leftarrow$ evaluateEnergy($\hat{S}_i$,$\hat{S}^s$,θ'',λ) 5:
6: $\quad\Delta_{\hat{e}} \leftarrow \hat{e}' - \hat{e}$ 6:
7: $\quad$Draw $x \sim$ *Bernoulli*(exp($-\Delta\hat{e}/\tau$))
8: $\quad$**if** $\Delta_{\hat{e}} \geq 0$ or $x = 1$ **then**
9: $\quad\quad$**if** $\rho_{\hat{S}^s} \geq 0$ **then**
10: $\quad\quad\quad\theta' \leftarrow \theta''$, $\hat{e} \leftarrow \hat{e}'$
11: $\quad\quad$**end if**
12: $\quad$**end if**
13: $\quad\tau \leftarrow \tau \times \alpha$
14: **end for**

Algorithm 1 presents a safeguarded version of simulated annealing. Specifically, on a given S_i to be repaired, it aims to improve robustness on the finite sampled initial set $\hat{S}_i$ while protecting the robustness of $\hat{S}^s$. The key subroutine is evaluateEnergy at lines 2 and 5, which takes as input $\hat{S}_i$, $\hat{S}^s$, current control policy parameters θ and balance factor λ and outputs energy $\hat{e}$ as in Equation (13) and the minimum robustness $\rho_{\hat{S}^s}$ of protected initial states, i.e.,

$$\rho_{\hat{S}^s}(\theta) := \min_{s_0\in\hat{S}^s} \mathrm{rob}_{f,\varphi}(s_0,\pi_\theta). \tag{17}$$

From lines 3 to 15, we have the main loop of simulated annealing. The parameters are perturbed with Gaussian noise at line 4, and energy is evaluated again at line 5. Line 8 provides the standard criterion check of simulated annealing as explained in Section 2.1.5, i.e., Metropolis-Hastings criterion. However, this criterion check is insufficient to safeguard all initial states in $\hat{S}^s$, since a new parameter θ' may break them. Hence, we add an additional safeguard on the minimum robustness at line 9, to ensure that all $s_0 \in \hat{S}^s$ still produce trajectories that accomplish the task. Finally, the temperature cools down at line to gradually discourage exploration.

The ISAR algorithm

With the safeguarded simulated annealing defined in Algorithm 1, we use it as a subroutine in our main ISAR algorithm, detailed in Algorithm 2.

Algorithm 2 Incremental Simulated Annealing Repair (ISAR)

Input: Partitioned continuous initial state set $S_{init} = S_1 \cup \ldots \cup S_M$, control policy parameters θ, sample size K, energy balance factor $\lambda > 0$, perturbation std $\sigma > 0$, initial temperature $\tau > 0$, cooling factor $\alpha \in (0,1)$, max iteration Q

Output: Repaired control policy parameters θ and final verification outputs v_1', . . . , v_M'

1: **for** each S_i **do**
2: $\quad v_i \leftarrow \mathrm{ver}_{f,\phi}(S_i,\pi_\theta)$
3 **end for**
4: **for** each S_i **do**
5: $\quad$Uniformly sample K states in S_i to form finite set $\hat{S}_i$
6: $\quad$**for** each $s_0^j \in \hat{S}_i$ **do**
7: $\quad\quad\rho_i^j \leftarrow \mathrm{rob}_{f,\phi}(s_0^j,\pi_\theta)$
8: $\quad$**end for**
9: **end for**
10: $\mathcal{S}^s \leftarrow \{S_i : v_i = 1\}$ ▷ regions to be protected
11: $\hat{S}^s \leftarrow \bigcup_{S_i\in\mathcal{S}^s} \hat{S}_i$
12: $\mathcal{S}^f \leftarrow \{S_i : \exists j, \rho_i^j < 0\}$ ▷ regions to be repaired
13: Sort $\mathcal{S}^f$ in decreasing order of $\sum_j \rho_i^j$ of each S_i
14: **while** $\mathcal{S}^f$ is not empty **do**
15: $\quad S_i \leftarrow$ first region in $\mathcal{S}^f$ in sorted order

16: $\hat{\mathcal{S}}^{\mathrm{f}} \leftarrow \{s_0{}^j \in \hat{S}_i : \rho_i{}^j < 0\}$
17: $\theta' \leftarrow$ safeguardedSimulatedAnneling($\hat{\mathcal{S}}^{\mathrm{f}}$, $\hat{\mathcal{S}}^{\mathrm{s}}$, θ, λ, σ, τ, α, Q) ▷ Alg. 1
18: **if** $\theta' \neq \theta$ **then** ▷ new control policy identified
19: **for** each $S_k \in \mathcal{S}^{\mathrm{f}}$ **do** ▷ re-evaluate robustness
20: **for** each $s_0^j \in S_k$ **do**
21: $\rho_k{}^j \leftarrow \mathrm{rob}_{f,\phi}(s_0{}^j, \pi_{\theta'})$
22: **end for**
23: **end for**
24: Sort $\mathcal{S}^{\mathrm{f}}$ in decreasing order of $\sum_j \rho_k^j$ of each S_k
25: $\mathcal{S}' \leftarrow \{S_k \in \mathcal{S}^{\mathrm{f}} : \min_j \rho_k{}^j \geq 0\}$ ▷ newly repaired regions
26: $\mathcal{S}^{\mathrm{f}} \leftarrow \mathcal{S}^{\mathrm{f}} \setminus \mathcal{S}'$, $\mathcal{S}^{\mathrm{s}} \leftarrow \mathcal{S}^{\mathrm{s}} \cup \mathcal{S}'$
27: $\hat{S}^{\mathrm{s}} \leftarrow \bigcup_{S_k \in \mathcal{S}^{\mathrm{s}}} \hat{S}_k$, $\theta \leftarrow \theta'$
28: **end if**
29: **end while**
30: **for** each S_i **do**
31: $v_i' \leftarrow \mathrm{ver}_{f,\phi}(S_i, \pi_\theta)$
32: **end for**

Algorithm 2 can be divided into three parts: preparation (lines 1–13), main repair loop (lines 14–29) and final verification (lines 30–32). In the preparation phase, each initial state region S_i is verified under the current control policy π_θ by calling the sound but incomplete verifier $\mathrm{ver}_{f,\phi}$. This step can be parallelized by individual regions. At lines 4-9, we uniformly sample K initial states from each region S_i to form a finite set, to estimate the energy function via Monte Carlo integrals. The robustness of each sampled initial state is evaluated at line 7. This sampling and robustness computation can also be parallelized.

The regions and states to be protected are defined in lines 11 and 12, respectively, while the regions to be repaired are defined in line 12. As a final preparation step, we sort the to-be-repaired regions by decreasing order of average sampled robustness, since we expect regions with high robustness to be easier to repair.

In the main repair loop from lines 14 to 29, we first select the next failed region to repair (line 14) and identify which of its sampled states in $\hat{S}_i$ fail to accomplish the task. These states then get repaired by the safeguarded simulated annealing (Algorithm 1 called at line 17). After simulated annealing, we check whether a new policy is obtained.

If we have a new policy, the robustness of the sampled states in the remaining to-be-repaired regions is evaluated again (lines 19–23), and the regions are sorted based on the evaluation (line 24). Repairing a region S_i may also repair other regions, so we identify all the repaired regions as $\mathcal{S}'$ (line 25). These regions are taken away from the to-be-repaired regions and added to the to-be-protected ones (line 26), so that we safeguard both the previously verified regions and the newly repaired ones.

Finally, after the repair is done, we rerun the verifier at lines 30–32 to obtain the final verification results. The verifier is not called during the repair procedure because it is generally expensive compared to computing robustness on individual sampled states.

We identify two potential challenges in the execution of Algorithm 2. First, there may exist a protected region that does not pass verification at the end. We could increase the number of protected sampled states (larger K) in that region or adjust the balance factor λ. However, we did not encounter this issue during case studies. Second, the main repair loop can take a long time. In practice, this loop can be terminated early when the repairing speed is slow, i.e., very few to no additional regions are repaired and removed at line 26. This observation implies that θ has converged.

Scalability analysis

Here, we analyze the scalability of ISAR with respect to the complexities of the policy NN and dynamics equations. In Algorithm 2, the subroutine with dominant computational overhead is evaluating the STL robustness function $\mathrm{rob}_{f,\phi}$. Therefore, we start by analyzing the complexity of this evaluation.

Evaluating the STL robustness of specification ϕ requires computing a trajectory in the simulator. Let T denote the total time steps required for simulation, each time step consisting of two parts: (1) a forward pass on the policy NN π_θ to obtain action on the current state, i.e., $a_t = \pi_\theta(s_t)$, and (2) a state update via the dynamics, i.e., $s_{t+1} = f(s_t, a_t)$.

For the forward pass, we consider a NN-based policy $\pi: S \mapsto A$ implemented by a fully connected multi-layer perceptron (MLP), which is

$$\pi(s_t) := \alpha_P(w_P \cdot \alpha_{P-1}(w_{P-1} \cdot \alpha_{P-2}(\ldots \alpha_1(w_1 \cdot s_t + b_1)\ldots) + b_{P-1}) + b_P) \tag{18}$$

where P is the total number of layers, and α_i, w_i, b_i are the activation function, weight and bias of the i-th layer, respectively. If we denote the state space dimension as $N_0 = N_s$ and action space dimension as $N_P = N_a$, and the number of neurons at each hidden layer i as N_i, we are able to analyze complexity in number of scalar multiplications. At a hidden layer i, the $N_i \times N_{i-1}$-dimensional weight matrix multiplies with the N_{i-1}-dimensional embedding, resulting in $O(N_{i-1}N_i)$ scalar multiplications. Hence, the forward pass complexity is

$$O(N_0N_1) + O(N_1N_2) + \ldots + O(N_{P-1}N_P) = O\left(\sum_{i=1}^{P} N_{i-1}N_i\right). \tag{19}$$

Assuming the maximum number of neurons per hidden layer is N, we have

$$O\left(\sum_{i=1}^{P} N_{i-1}N_i\right) = O(N_0N + NN_P + (P-1)N^2) = O(PN^2) \tag{20}$$

The other half of the computation in one simulation step is the state update with dynamics $f: S \times A \mapsto S$. Here, we consider a polynomial dynamics f of degree D, as various nonlinear dynamics can be approximated by polynomials (Ivanov et al., 2021). Recall that every state s_t is a vector of N_s elements, and every action a_t is a vector of N_a elements. We use $s_{i,t}$ and $a_{i,t}$ to denote the i-th dimension of state s_t and action a_t, respectively. That is, $s_t = (s_{1,t}, \ldots, s_{N_s}, t)$ and $a_t = (a_{1,t}, \ldots, a_{N_a}, t)$. The general form of a D-th degree polynomial dynamics can be expressed as N_s scalar functions, $f_1, \ldots, f_{N_s}$, each computes one state dimension, as follows.

$$f_i(s_t, a_t) = \sum c_{\beta,\gamma} s_{1,t}^{\beta_1} \ldots s_{N_s,t}^{\beta_{N_s}} a_{1,t}^{\gamma_1} \ldots a_{N_a,t}^{\gamma_{N_a}}, \text{ for } i = 1, \ldots N_s. \tag{21}$$

Equation (21) is a summation of scalar monomials. In this equation, the superscripts are integer exponents, denoted as β_i and γ_i, satisfying $\sum_{i=1}^{N_s} \beta_i + \sum_{i=1}^{N_a} \gamma_i \leq D$, and $c_{\beta,\gamma}$ is some real-valued coefficient of a monomial.

To analyze the complexity of evaluating a D-th degree polynomial dynamics, we first find the number of monomials,

which is
$$\binom{N_s+N_a+D}{D} = \frac{(N_s+N_a+D)!}{D!(N_s+N_a)!}. \quad (22)$$

With binomial approximation, we have

$$\binom{N_s+N_a+D}{D} \approx (N_s+N_a)^D. \quad (23)$$

Recall that all these monomials compute f_i, which computes one scalar state variable in s_{t+1}. Therefore, the total time complexity, in number of scalar multiplications, of computing a state update in a general degree-d polynomial dynamics is

$$N_s\binom{N_s+N_a+D}{D} = O(N_s(N_s+N_a)^D). \quad (24)$$

Now we have the complexity of evaluating a forward pass in Equation (20) and dynamics in Equation (24). These computations have to run for each of the T steps. Therefore, running a simulation to generate a trajectory takes $O(T(PN^2+N_s(N_s+N_a)^D))$. When evaluating STL robustness for some specification ϕ, we compute max, min and other $O(1)$ arithmetic operations along the simulation. Therefore, evaluating STL robustness also has a time complexity of $O(T(PN^2+N_s(N_s+N_a)^D))$.

The analysis above results in the following proposition.

Proposition 2 (Complexity of ISAR). *Let the state space and action space dimensions be N_s and N_a, the dynamics be a D-th degree polynomial, the policy NN have N neurons at max per layer on the P layers, and evaluating the STL robustness requires running the simulation for T steps. We split the initial state space into M regions, sample K initial states per region, and run simulated annealing for Q iterations in Algorithm 1. Then, the computational complexity of ISAR is*

$$O(QKM^2T(PN^2+N_s(N_s+N_a)^D)). \quad (25)$$

Proof. In Algorithm 2, the dominant computation is simulated annealing at line 17. There are at most M regions to be repaired, so the total number of simulated annealing runs is $O(M)$.

Within each simulated annealing, the dominant computation is evaluating the energy function at line 5 in Algorithm 1. This energy function consists of two parts, referring to Equation (13): evaluating STL robustness from 1 to-be-repaired region, and from at most M to-be-preserved regions. In each region, this evaluation is done on K sampled states. Therefore, $O(KM)$ STL robustness evaluation needs to be done to compute the energy function, which is computed for Q times. Therefore, the total number of evaluating STL robustness in simulated annealing is $O(QKM)$.

As ISAR runs $O(M)$ simulated annealing in total, the overall number of evaluations of STL robustness is $O(QKM^2)$. From the complexity analysis above, we know each robustness evaluation costs $O(T(PN^2+N_s(N_s+N_a)^D))$. Consequently, the total complexity is $O(QKM^2T(PN^2+N_s(N_s+N_a)^D))$.

Based on Proposition 2, we can analyze the scalability of ISAR with respect to various scale factors. First, the time complexity scales linearly with the iterations in simulated annealing Q, the number of samples per region K, the total time steps to evaluate STL robustness T and the number of policy network layers P. Enlarging these factors would impose a linear impact on the total time. Next, we have the total number of regions M and the maximum number of neurons per layer N, which influence the total runtime quadratically. Therefore, the users of ISAR should be cautious when using fine region partitions (large M) or wide embeddings per NN layer (large N). Finally, notice that the complexity scales up with the degree of dynamics D exponentially, and this exponent directly affects the dimensions of state space and action space. Consequently, the best practice would be approximating the dynamics to lower degrees, such as keeping the first 1 to 2 terms of Taylor polynomials.

Results

Experiment setup

To evaluate ISAR, we compare it to the following four baselines: STLGym and F-MDP in Section 2.3.1, MIQP and minimally deviating repair in Section 2.3.2 and tube MPC shielding in Section 2.3.3. We implement these baselines as follows.

To repair with STLGym and F-MDP, the reinforcement learning updates the policy by exploring starting from the failed sampled initial states. Both STLGym and F-MDP are implemented based on three variants of state-of-the-art reinforcement learning algorithms: PPO (Schulman et al., 2017), A2C (Mnih et al., 2016) and SAC (Haarnoja et al., 2018), to evaluate the impact of learning algorithm selection. For F-MDP, we design a finite automaton for each case study, with the flag state as an additional integer dimension. This heuristic is the same as the original F-MDP paper (Venkataraman et al., 2020).

Tube MPC shielding does not modify the policy network. Instead, it performs reachability analysis from the sampled initial states and intervenes with an MPC solver if the specification cannot be fulfilled. Due to the non-convex nature of system dynamics, we solve the MPC optimization with a state-of-the-art non-convex MPC solver *IPOPT* (Andersson et al., 2019). Since the tube MPC shielding baseline does not modify the NN itself, the verification result of the NN-based policy shall remain the same. To evaluate the effectiveness of this baseline method, we check how many sampled successful initial states were broken, as well as the number of sampled failed initial states that were corrected.

We also run MPC on originally failed initial states and collect the trajectory data if they are repaired. This is the same way as collecting training data in the supervised learning baselines, MIQP and minimally deviating repair (Majd et al., 2021; Zhou et al., 2020). We then feed the collected data to train these two methods. For MIQP, the last layer of NN is selected to be repaired, the same as in the experiments of the MIQP paper.

The following three sets help us define our evaluation metrics:

1. $\mathcal{S}_\pi^{\mathrm{s}}$: the set of regions that pass verification under a control policy π (same as Equation (9)).
2. $\tilde{\mathcal{S}}_\pi^{\mathrm{s}}$: the set of regions that do not pass verification under a control policy π, but for which no failure is identified by sampling $K = 100$ initial states per region.
3. $\mathcal{S}_\pi^{\mathrm{f}}$: the set of regions with failure sampled under a control policy π (same as Equation (8)).

We track three key metrics according to the main problem. First, since our primary goal is to preserve the verifiability of regions, we check how many verified regions are lost, i.e., the number of regions in $\mathcal{S}_\pi^{\mathrm{s}}$ that no longer pass verification. Second, we check how many broken regions are repaired, i.e., the number

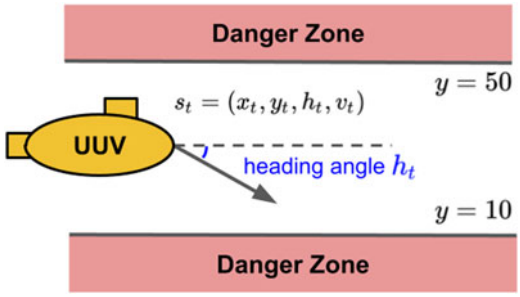

Figure 2. The unmanned underwater vehicle.

of regions in $\mathcal{S}_\pi^{\mathrm{f}}$ that no longer contain any failed sampled states. We also split the results into verified repair (previously in $\mathcal{S}_\pi^{\mathrm{f}}$, now in $\mathcal{S}_\pi^{\mathrm{s}}$) and sampled repair (previously in $\mathcal{S}_\pi^{\mathrm{f}}$, now in $\tilde{\mathcal{S}}_\pi^{\mathrm{s}}$). Third, we evaluate the minimal STL robustness of the sampled states in each region to estimate the worst-case scenarios of the regions. For a successful region, this worst-case STL robustness means how far it is from being broken, and for a failed region, this means how close it is to being repaired. We use Verisig (Ivanov et al., 2021; Ivanov et al., 2019) as our verifier.

In addition to the baseline comparisons, we also conduct two ablation studies: (1) replacing simulated annealing with gradient ascent in ISAR and (2) removing the log barrier function in ISAR.

We carry out the experiments on two case studies: an UUV and the OpenAI Gym mountain car (MC). The details of these two systems are explained in the sections below. The experiments are run on Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz. All parallel computations are distributed to CPUs of this type. The OS is Ubuntu 20.04.6 LTS, with kernel Linux 5.4.0-159-generic and architecture x86-64.

Case study 1: unmanned underwater vehicle (UUV)

The UUV control problem is based on a challenge problem from the DARPA Assured Autonomy program (Ivanov et al., 2021; Ruchkin et al., 2022). The UUV has a four-dimensional state space (x_t, y_t, h_t, v_t), where (x_t, y_t) are the two-dimensional coordinates, h_t is the heading angle and v_t is the velocity, as illustrated in Figure 2. The x-coordinate always starts at 0, and the velocity is fixed at 0.4855 m/s.

The NN-based control policy is implemented consistently with the previous literature (Ruchkin et al., 2022). Specifically, it is a multi-layer perceptron with 2 hidden layers, each with 32 neurons, and 1 output layer with 1 neuron, representing the turning angle at 0.5 Hz. Each hidden layer is activated by sigmoid and the output layer by tanh. Training is carried out using the TD3 algorithm (Fujimoto et al., 2018), the same as in a previous research (Ruchkin et al., 2022). At every time step t, the UUV receives two measurements: the increment in heading angle to the pipe Δh_t and the distance to the lower edge of the pipe r_t. We assume the measurement noise is negligible. The UUV then computes a one-dimensional turning angle action a_t based on the two measurements via the control policy.

The DARPA Assured Autonomy program posed the following challenge, specified as an STL task. The UUV must travel within a range of distances from a pipe that it is scanning. The upper and lower distance bounds are at $y = 10$ and $y = 50$, respectively. The control policy's goal is to keep the UUV within this safe range for the next 30 s. We formalize this task in STL as

$$\varphi = G_{t\in[0,30]}((y_t > 10) \wedge (y_t < 50)). \tag{26}$$

The dynamics of the UUV is as follows.

$$\begin{pmatrix} \Delta h_t \\ r_t \end{pmatrix} = \begin{pmatrix} 18.753 & 1.565 & 0.174 & -0.131 \\ -0.04 & -1.608 & -70.832 & -0.061 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \\ h_t \\ 0.4855 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_t$$
$$h_{t+1} = h_t + \Delta h_t$$
$$x_{t+1} = x_t + r_t \cos(h_t)$$
$$y_{t+1} = y_t - r_t \sin(h_t) \tag{27}$$

The state space of the UUV is $S = \{(y, h) | y \in [0, 50], h \in [-180, 180]\}$. The initial state space is $S_{init} = \{(y, h) | y \in [12, 22], h \in [10, 30]\}$. To repair the policy, we partition S_{init} into rectangular regions, with step size 0.1m for y and 1.0 degree for h. As a result, we obtained 2000 regions in total.

For fair comparison, we apply the same stopping criterion: when the average robustness of the failed sampled initial states can no longer be improved. We illustrate the repair results in Figure 3. Here, the solid green regions are the regions that pass Verisig ($\mathcal{S}_\pi^{\mathrm{s}}$), hatched green regions are the regions that does not pass Verisig, but with no failures sampled, i.e., Verisig may be conservative on those regions ($\tilde{\mathcal{S}}_\pi^{\mathrm{s}}$), and red regions are the ones with failures sampled ($\mathcal{S}_\pi^{\mathrm{f}}$). The quantitative results are presented in Table 1. Here, the original policy is denoted as π, and the alternative policy identified is π'. Since the verifier is conservative, we care about the total number of repaired regions, including both verified and sampled repairs. In practice, turning red regions into solid green (verified repair) is difficult due to the verifier's conservatism and a naive repair algorithm. We report both cases of repairs for clarity.

When collecting training data using MPC for MIQP and minimally deviating repair, we obtain 471 repaired initial states. The trajectory length is 30, so the training data has $471 \times 30 = 14130$ data points.

As mentioned above, tube MPC shielding does not alter the policy network, so it is not evaluated with the metrics above. Instead, we select 10 sampled states per region and check whether tube MPC shielding will successfully intervene. That is, we have 20000 initial states examined, consisting of 11090 successful and 8910 failed states before repair. We check how many of the initial states that previously produce successful trajectories but are broken afterwards and how many of the failed ones are repaired. The results are shown in Table 2.

Case study 2: mountain car (MC)

Mountain Car (MC) is a control problem in OpenAI Gym (Brockman et al., 2016). Here, we control a car with a two-dimensional state space (x_t, v_t), where x_t is the one-dimensional position (in the left-right direction) and v_t is the velocity. The policy model is trained in the same way with TD3 algorithm from a previous research (Fujimoto et al., 2018; Ruchkin et al., 2022) as in UUV. The OpenAI Gym task is to drive the car from the bottom of a valley to the top of a mountain to its right ($x \geq 0.45$) within 110 s, formalized in STL as

$$\varphi = F_{t\in[0,110]}(x_t \geq 0.45). \tag{28}$$

A visualization of MC is given in Figure 4. This problem's challenge is that the car must first accumulate momentum by backing to the left hill (waypoint 1 in the figure). Otherwise, it will not be able to reach to goal on the right hill (waypoint 2). In other

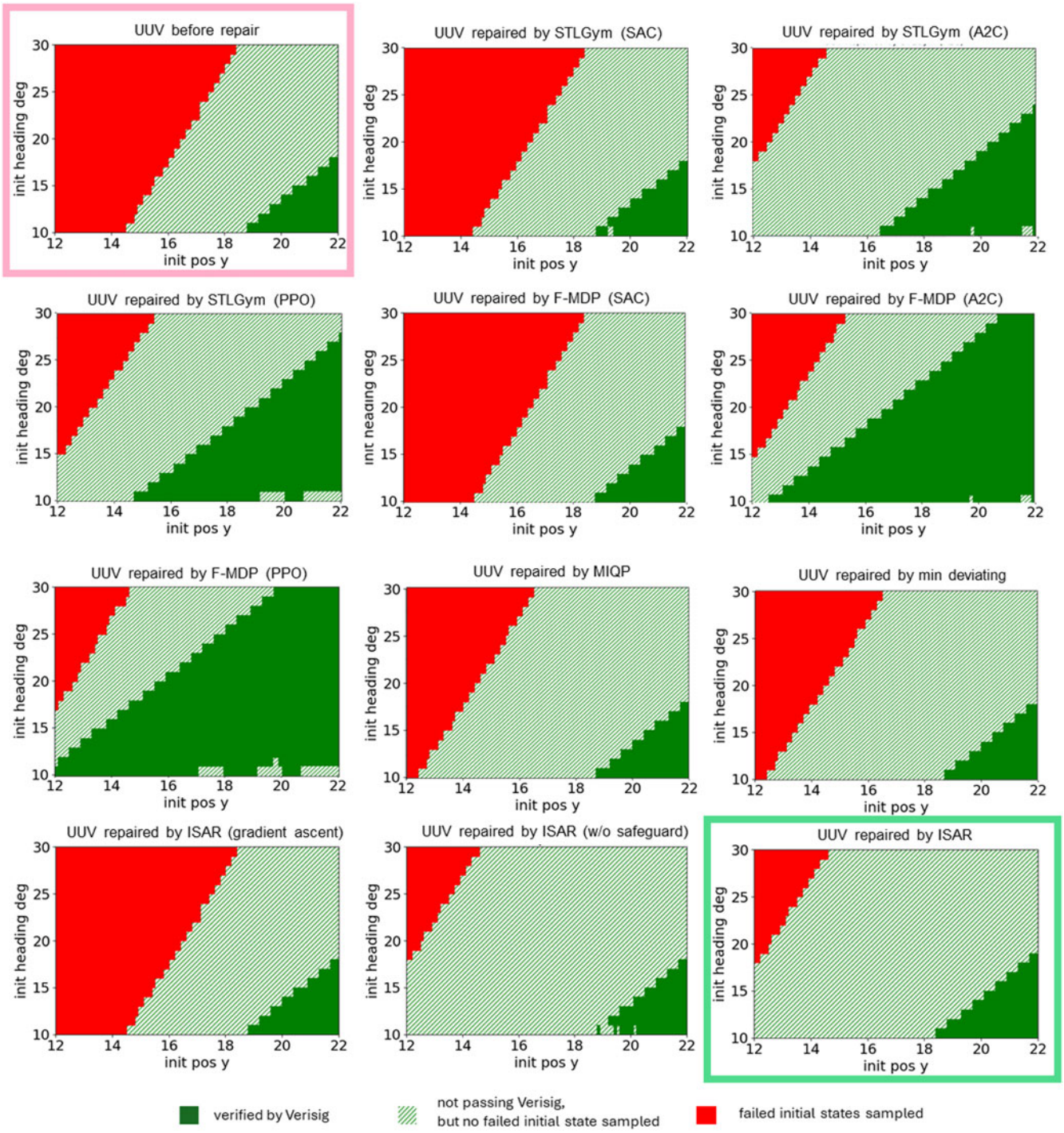

Figure 3. Repair results of the UUV.

words, to reach the goal, the car must first drive away from it. The control policy must learn this trick to succeed.

For MC, we repair a feed-forward neural network policy π (Ruchkin et al., 2022), with 2 hidden layers of 16 neurons each. The hidden layers are sigmoid-activated and the output layer is tanh-activated, with an output u_t representing the positive or negative throttle. The dynamics of MC are as follows:

$$\begin{aligned} v_{t+1} &= v_t + 0.0015u_t - 0.0025\cos(3x_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned} \tag{29}$$

The state space of MC is $S = \{(x,v) | x \in [-1.2, 0.6], v \in [-0.07, 0.07]\}$. The initial state space is $S_{init} = \{(x, v) | x \in [-0.505, 0.395], v \in [-0.055, 0.045]\}$, with partition step sizes of 0.01 and 0.01, respectively. There are 900 regions in total.

Table 1. Quantitative repair results of the UUV ($\pi' = \pi$ before repair)

	$\lvert\mathcal{S}^{\mathrm{s}}_{\pi'}\rvert : \lvert\tilde{\mathcal{S}}^{\mathrm{s}}_{\pi'}\rvert : \lvert\mathcal{S}^{\mathrm{f}}_{\pi'}\rvert$	# regions in $\mathcal{S}_{\pi}{}^{\mathrm{s}}$ broken	# regions in $\mathcal{S}_{\pi}{}^{\mathrm{f}}$ repaired	# verified repair	# sampled repair	Min rob per region in $\mathcal{S}_{\pi'}{}^{\mathrm{f}}$	Min rob per region in $\mathcal{S}^{\mathrm{s}}_{\pi'} \cup \tilde{\mathcal{S}}^{\mathrm{s}}_{\pi'}$	Min rob per region overall
Before repair	141:963:896	N/A	N/A	N/A	N/A	−0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
STLGym (SAC)	139:965:896	2 (1.4%)	0 (0%)	0 (0%)	0 (0%)	−0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
STLGym (A2C)	379:1453:168	5 (3.5%)	728 (81.3%)	0 (0%)	728 (81.3%)	−0.11 ± 0.04	4.91 ± 2.8	4.49 ± 3.01
STLGym (PPO)	632:1095:273	18 (12.8%)	623 (69.5%)	0 (0%)	623 (69.5%)	−0.13 ± 0.04	4.35 ± 2.56	3.74 ± 2.83
F-MDP (SAC)	141:963:896	**0 (0%)**	0 (0%)	0 (0%)	0 (0%)	−0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
F-MDP (A2C)	1047:689:264	5 (3.5%)	632 (70.5%)	48 (5.4%)	584 (65.2%)	−0.13 ± 0.04	4.44 ± 2.61	3.84 ± 2.88
F-MDP (PPO)	1236:587:177	22 (15.6%)	719 (80.2%)	**119 (13.3%)**	600 (66.9%)	−0.11 ± 0.04	4.87 ± 2.78	4.43 ± 3.01
MIQP	143:1458:499	**0 (0%)**	397 (44.3%)	0 (0%)	397 (44.3%)	−0.17 ± 0.05	3.91 ± 2.36	2.89 ± 2.7
Min deviating	146:1357:497	**0 (0%)**	399 (44.5%)	0 (0%)	399 (44.5%)	−0.17 ± 0.05	3.8 ± 2.31	2.82 ± 2.64
ISAR (gradient ascent)	141:963:896	**0 (0%)**	0 (0%)	0 (0%)	0 (0%)	−0.24 ± 0.06	2.79 ± 1.89	1.49 ± 2.05
ISAR (w/o safeguard)	132:1702:166	9 (6.4%)	**730 (81.4%)**	0 (0%)	**730 (81.4%)**	**−0.1 ± 0.03**	4.73 ± 2.79	4.27 ± 3.01
ISAR	173:1661:166	**0 (0%)**	**730 (81.4%)**	0 (0%)	**730 (81.4%)**	**−0.1 ± 0.04**	**4.92 ± 2.8**	**4.51 ± 3.02**

Table 2. Repair result of tube MPC shielding in the UUV

	# of successful states broken due to false intervention	# of failed states repaired
Tube MPC shielding	538/11090 (4.9%)	2370/8910 (26.6%)
ISAR	0/11090 (0%)	7301/8910 (81.9%)

When collecting training data using MPC, we run on 827 originally successful states and 8173 failed ones. Only 2 out of the 8173 failed ones are repaired by MPC, and these $2 \times 110 = 220$ data points are utilized to train MIQP and minimally deviating repair.

Like for the UUV, we illustrate the repair results of MC on all the methods, i.e., Figure 5, as well as the quantitative results, i.e., Table 3. We also have tube MPC shielding results in Table 4, on 10 sampled states per region, so there are 9000 sampled states evaluated in total. The reason for such a low quantity of repaired states is the same for running MPC when collecting training data: non-convex dynamics and a long horizon. Further analysis is in Section 3.4.2.

Finally, we present the repair computation time of all methods for both UUV and MC. This time excludes verification time, which is 360 ± 272 s per region for UUV and 1975 ± 681 s per region for MC. For MIQP and minimally deviating repair, although the training time is short, they require data generation from an MPC. Therefore, the data generation time is included.

Discussion

Comparison to reinforcement learning methods

We first compare the three variants of RL algorithms: SAC, A2C and PPO. For the UUV, based on Figure 3, the three algorithms all repair a certain subset of the failed regions $\mathcal{S}^{\mathrm{f}}_{\pi}$ but break the verifiability of some regions in $\mathcal{S}^{\mathrm{s}}_{\pi}$. Notice that the broken regions are in the bottom right corner, the farthest from the failed states in the top left. This is due to the training data being from the failed sampled states in the far left and without considering preservation. Therefore, the learning overfits to the failed regions and downgrades its performance farther away from these regions. Among the three algorithms, A2C and PPO repair 81.3% and 69.5% of the regions with failures when being used by STLGym and 70.5% and 80.2% by F-MDP. These repair percentages are close to but not quite matching 81.4% of ISAR based on Table 1. Differently from A2C and PPO, SAC produces an output that barely changes anything. This is because, in contrast to A2C and PPO, SAC (1) is off-policy and (2) maximizes entropy in exploration. These features make SAC more exploration-focused than the other methods, instead of focusing on specific improvement directions that are likely to be correct. As the relationship between NN parameters and STL robustness is a complicated function, such exploration tends to be unsuccessful if the algorithm is unlucky. From Table 5, we can see that the training time of SAC in UUV takes significantly more time than A2C and PPO.

For the MC, the situation becomes different. Different from UUV, MC requires the policy network to learn the trick that first

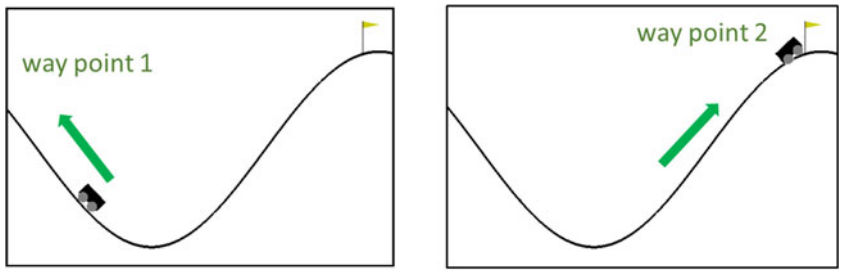

Figure 4. The openAI Gym mountain car.

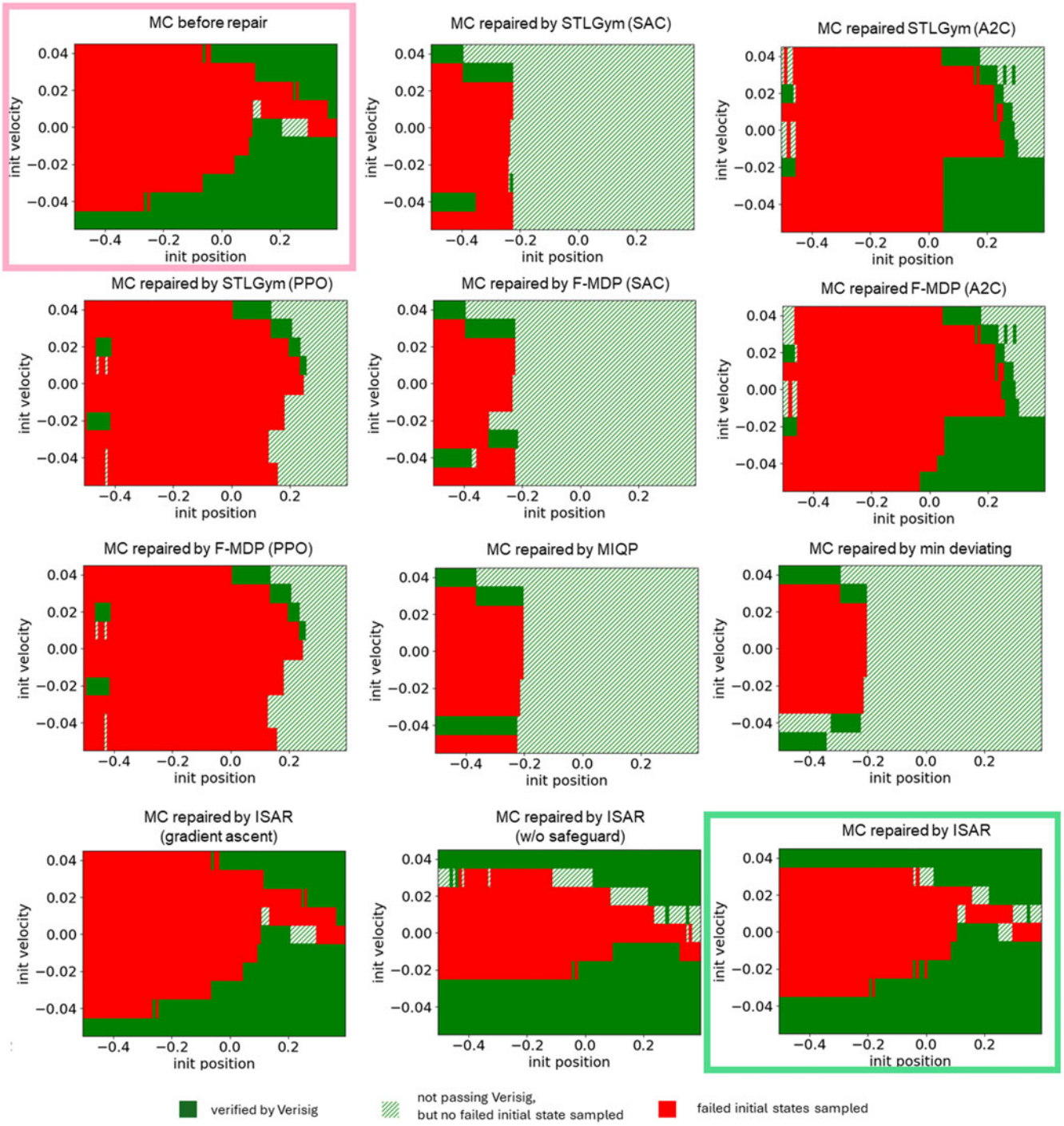

Figure 5. Repair results of MC.

Table 3. Quantitative repair results of MC ($\pi' = \pi$ before repair)

	$\lvert\mathcal{S}^{s}_{\pi'}\rvert : \lvert\tilde{\mathcal{S}}^{s}_{\pi'}\rvert : \lvert\mathcal{S}^{f}_{\pi'}\rvert$	# regions in $\mathcal{S}^{s}_{\pi}$ broken	# regions in $\mathcal{S}^{f}_{\pi}$ repaired	# verified repair	# sampled repair	Min rob per region in $\mathcal{S}^{f}_{\pi'}$	Min rob per region in $\mathcal{S}^{s}_{\pi'} \cup \tilde{\mathcal{S}}^{s}_{\pi'}$	Min rob per region overall
Before repair	366:11:523	N/A	N/A	N/A	N/A	−0.37 ± 0.22	0.14 ± 0.02	−0.12 ± 0.3
STLGym (SAC)	64:691:145	326 (89.1%)	125 (23.9%)	64 (12.2%)	61 (11.7%)	−0.64 ± 0.15	0.09 ± 0.09	−0.03 ± 0.22
STLGym (A2C)	208:129:563	168 (45.9%)	58 (11.1%)	31 (5.9%)	27 (5.2%)	−0.54 ± 0.18	0.11 ± 0.06	−0.45 ± 0.25
STLGym (PPO)	56:240:604	302 (82.5%)	34 (6.5%)	2 (0.4%)	32 (6.1%)	−0.55 ± 0.17	0.12 ± 0.05	−0.44 ± 0.25
F-MDP (SAC)	72:701:127	326 (89.1%)	143 (27.3%)	71 (13.6%)	**72 (13.7%)**	−0.64 ± 0.15	0.09 ± 0.09	−0.03 ± 0.22
F-MDP (A2C)	208:131:561	168 (45.9%)	60 (11.5%)	31 (5.9%)	29 (5.6%)	−0.54 ± 0.18	0.11 ± 0.06	−0.45 ± 0.25
F-MDP (PPO)	56:240:604	302 (82.5%)	34 (6.5%)	2 (0.4%)	32 (6.1%)	−0.55 ± 0.17	0.12 ± 0.05	−0.44 ± 0.25
MIQP	62:691:147	326 (89.1%)	25 (4.8%)	8 (1.5%)	17 (3.3%)	−0.61 ± 0.18	0.12 ± 0.09	−0.26 ± 0.24
Min deviating	66:707:127	334 (91.3%)	35 (6.7%)	18 (3.4%)	17 (3.3%)	−0.63 ± 0.14	0.08 ± 0.06	−0.17 ± 0.31
ISAR (gradient ascent)	366:11:523	**0 (0%)**	0 (0%)	0 (0%)	0 (0%)	**−0.37 ± 0.22**	0.14 ± 0.02	−0.12 ± 0.3
ISAR (w/o safeguard)	485:52:363	29 (7.9%)	**189 (36.1%)**	**132 (25.2%)**	57 (10.9%)	−0.39 ± 0.19	0.14 ± 0.02	−0.05 ± 0.29
ISAR	469:21:410	**0 (0%)**	123 (23.5%)	104 (19.9%)	19 (3.6%)	−0.39 ± 0.17	**0.15** ± 0.01	**−0.03 ± 0.27**

Table 4. Repair result of tube MPC shielding in MC

	# of successful states broken due to false intervention	# of failed states repaired
Tube MPC shielding	346/827 (41.8%)	248/8173 (3.1%)
ISAR	0/827 (0%)	1235/8173 (15.1%)

Table 5. Repair time taken for each method in each case study

	Repair time (sec)			Repair time (sec)	
	UUV	MC		UUV	MC
STLGym (SAC)	1006	19	Tube MPC Shield	17687	15580
STLGym (A2C)	107	61	MIQP	37 (+3103 data generation)	0.2 (+2513 data generation)
STLGym (PPO)	236	31			
F-MDP (SAC)	1230	20	Min deviating	51 (+3103 data generation)	0.6 (+2513 data generation)
F-MDP (A2C)	186	66			
F-MDP (PPO)	302	44	ISAR (w/o) safeguard	27	15
ISAR (gradient ascent)	1040	302	ISAR	1560	360

backs up to accumulate momentum. In this case, vast exploration would show benefits, as improving in a single direction may fall for incorrect local optima. Per Figure 5 and Table 3, STLGym with SAC repairs a larger percentage of $\mathcal{S}^{f}_{\pi}$ (23.9% with STLGym and 27.3% with F-MDP), when compared to A2C (11.1% and 11.5%) and PPO (6.5% and 6.5%). Still, this repair is at the cost of losing verifiability of 89.1% of $\mathcal{S}^{s}_{\pi}$. The correct regions after repair also suffer from the lowest worst-case STL robustness, at 0.09 ± 0.09. Still, SAC can finish the repair early in this case, as A2C and PPO have to try different improvement directions for a long time, according to Table 5.

Next, we compare STLGym and F-MDP. For STLGym, we directly use the NN policy before repair to initialize the policy network and train on trajectories from the failed initial states sampled. For F-MDP, we first design the two finite-state automata of flag states for UUV and MC shown in Figure 6. The flag states, as in the original paper (Venkataraman et al., 2020), encodes additional information about the environment, such as whether the UUV is in a near-danger zone or if the MC is in the valley or has accumulated sufficient momentum. Next, we augment the to-be-repaired policy network with one additional input neuron, which takes the integer-encoded flag state. This new neuron has randomly initialized weights connecting to the first hidden layer. After training, we remove this neuron and its connected weights to produce the repaired policy network. In UUV, as shown in Figure 3

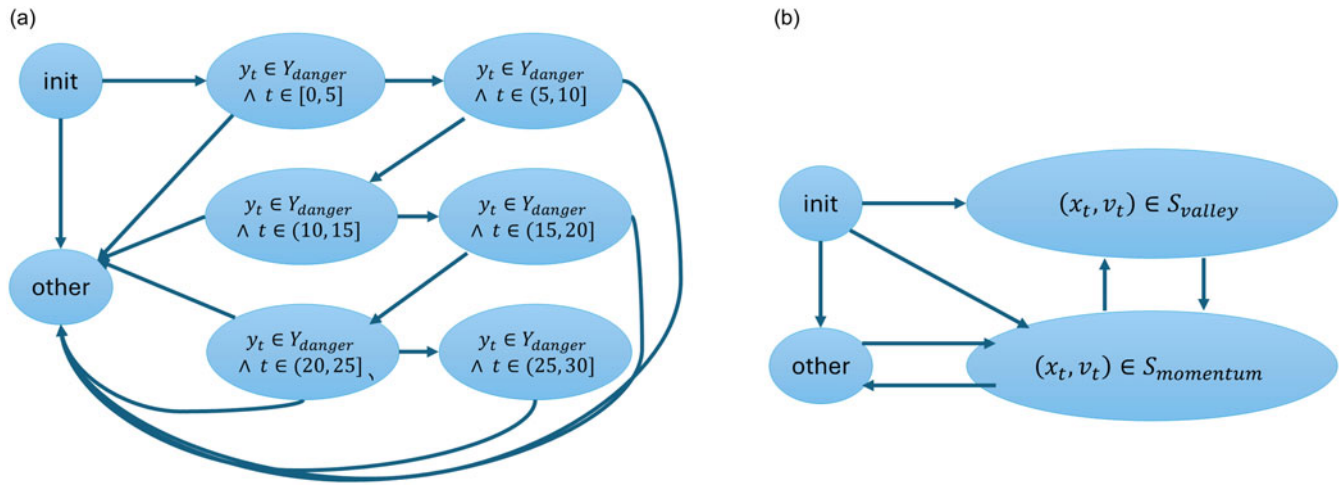

Figure 6. Flag states automata designed for (a) UUV and (b) MC. Here, $Y_{danger} = [10, 13]$, $S_{valley} = [-0.7, 0.3] \times [-0.07, 0.07]$ and $S_{momentum} = [-1.2, -0.7) \times [0.01, 0.07]$.

and Table 1, F-MDP significantly enlarges the regions being verified when equipped with A2C and PPO, thanks to the additional information provided by flag states. However, as useful as it is, enlarging verifiable regions is not our objective, as the primary goal is to preserve the existing verified ones. For the MC, as shown in Figure 5 and Table 3, the flag states do not help much, as the results are almost the same as STLGym. We point out that the performance of F-MDP depends on the design of flag states, which requires human knowledge of the system.

Finally, comparing ISAR to STLGym and F-MDP, the most significant advantage is that ISAR preserves all verified regions due to its safeguarding mechanism. Moreover, ISAR achieves the most number of regions repaired, as well as the highest worst-case robustness in all regions for UUV. In MC, the repair task is more difficult as the trick of accumulating momentum needs to be learned, and ISAR shows conservatism in its repair to preserve all verified regions. Nonetheless, ISAR achieves 23.9% of regions repaired, as well as higher worst-case robustness than the RL baselines.

Comparison to online tube MPC shielding

Computing tube MPC shielding requires first performing a reachability analysis at each initial state and then solving for MPC control sequences if the target state is not reachable. Both steps suffer from computational overheads and errors if (1) the horizon is long and (2) the dynamics are non-convex. As shown in Table 2, 4.9% of the successful sampled states of UUV are broken due to erroneous reachability analysis, which rolls forward the reachable set from these initial states. This situation is worsened in MC, as Table 4 shows that the reachability analysis falsely intervenes 41.8% of the successful initial states. We remind the reader that both dynamics are non-convex, with UUV having a horizon of 30 time steps and MC having 110. Moreover, the target subset of states in MC is much smaller than that of UUV, easily being missed by reachability analysis over a long horizon.

Solving for the control sequences via MPC is also difficult. With the dynamics being non-convex, solvers using convex optimization cannot be used. As described in the experiment setup in Section 3.1, we leverage a state-of-the-art non-convex solver, IPOPT, which implements a numerical method to search for interior points that satisfy the constraints. It is difficult for this procedure to find an optimum (or a dual optimum) on long horizons. As shown in Table 2 and Table 4, only 26.6% and 3.1% of the sampled failed states in the two benchmarks are repaired. In addition, this search is time-consuming, as it takes 17687 and 15580 s, respectively, to finish all computations.

In contrast, ISAR is not challenged by long horizons or non-convex dynamics. As it can preserve all the correct sampled initial states and repair 81.9% and 15.1% of the failed sampled initial states. Its computational time is also shorter than MPC computation, taking 1560 and 360 s, respectively.

Comparison to imitation learning methods

Like all supervised learning methods, imitation learning depends on the quality of training data. Both baselines require MPC to generate these data. When collecting data, we leverage the same method as in baselines and run MPC on failed initial states in a receding horizon to generate trajectories (this is different from tube MPC shielding, which uses MPC to alter actions of a given network). We exclude the trajectories that lead to failures, taking only the repaired trajectories for training. That is, we have $471 \times 30 = 14130$ data points for UUV training, where 471 is the number of failed sampled states repaired as shown in Table 2, and 30 is the horizon, i.e., the trajectory length. Likewise, for MC training, we have $2 \times 110 = 220$ data points. Same as all methods under comparison, the learning repeats in multiple epochs until the average STL robustness of failed sampled states can no longer be improved.

The major difference between MIQP and minimally deviating repair is that the former freezes the parameters except for one hidden layer, while the latter retrains all parameters. We mimic the original MIQP paper to freeze everything except for the last layer (Majd et al., 2021). This means minimally deviating repair has a larger degree of freedom compared to MIQP. As shown in Figure 3 and Table 1, both methods end up with similar results. However, Figure 5 and Table 3 show that minimally deviating repair has repaired more regions (6.7%) than MIQP (4.8%). These results show that imitation learning on a single hidden layer suffices when the task is as simple as UUV, but a larger degree of freedom would be beneficial as the task becomes more difficult as MC. For MC, adjusting only the last hidden layer tends to miss optima with more regions repaired.

Both imitation baselines do not break any verified regions in UUV, but they repair fewer regions compared to ISAR. This shows

that they perform conservatively. However, for MC, where repairing requires parameter improvement in a very specific direction, both break a large percentage of regions (89.1% and 91.3%) as they tend to improve in the wrong ways by treating local optima as global optima. This is the same disadvantage suffered by A2C and PPO as discussed in Section 3.4.1. Defeating these baselines, ISAR breaks none of the verified regions in MC, while repairing more failed ones. The relatively poorer performance of imitation learning in MC is affected by the training data, as only 220 data points are collected.

Another notable observation is that imitation learning itself is computationally fast. Based on Table 5, both learning methods take less than 1 s to finish. Unfortunately, they depend on data generation, which takes 3103 and 2513 seconds on the two benchmarks, respectively.

Analysis of ablation studies

We hereby analyze the results of the two ablation studies. Note that we originally chose simulated annealing instead of gradient ascent to avoid getting stuck at local optima, as the relationship between policy network parameters and STL robustness is complex and highly non-convex. Our experiment results match our expectation that the gradient ascent version of ISAR does not change the outcome. This is due to the objective function having a very uneven landscape, forming numerous local optima with low STL robustness, which traps gradient ascent.

When the safeguarding mechanism is removed, one significant improvement is the computational time. As shown in Table 5, ISAR without safeguarding shortens the computational time from 1560 and 360 s to 27 and 15 s, respectively. This is because (1) no log barrier function is computed, as the log barrier needs to check STL robustness on all sampled states to be preserved, and (2) the parameters do not roll back under a safety check. Compared to all the baselines and ablated versions, ISAR is more computationally expensive due to these two major overheads. It is a promising future research direction to improve ISAR by reducing its computation time. However, without the safeguards, the algorithm breaks verified regions in both benchmarks.

Additional discussion

Complexity due to the safeguarding constraint. As we analyzed in Section 2.4.4, the overall complexity of the computation is $O(QKM^2T(PN^2+N_s(N_s+N_a)^D))$. This complexity mostly depends on the STL robustness constraint on safeguarding the verified regions. That is, at every iteration of simulated annealing, we need to compute the STL robustness of $O(M)$ safeguarded regions in order to evaluate the energy function. If the safeguarding constraint is removed, only 1 region (the to-be-repaired region) needs to have its sampled states' robustness computed. This will reduce the overall complexity by $O(M)$ times, i.e., the complexity will become $O(QKMT(PN^2+N_s(N_s+N_a)^D))$, a boost especially when the partition is fine and M is large.

However, we observed the negative impact of removing the safeguarding constraint in the experiments. Tables 1 and 3 demonstrate the ablation study of removing the safeguard. In the UUV study, 9 regions are broken, while the number of repaired regions remains the same. In the MC study, 29 regions are broken, while more regions are repaired compared to safeguarding. We can see a trade-off between computational time and the effectiveness of preservation. Since MC is a more difficult STL task than UUV, this trade-off becomes more obvious. This is because achieving a more difficult task from different initial states requires more dissimilar control policies. As our primary goal is to preserve the verified correctness in regions, removing the safeguards is an unacceptable compromise.

Limitations due to conservatism of the verifier. The consequence is that it leads to false negatives in verification. In our case, regions classified as $\tilde{\mathcal{S}}_\pi^s$ (green hatches in the figures) could actually be $\mathcal{S}_\pi{}^s$ (solid green). This conversion could affect the decision of whether a region possesses verified correctness.

Reducing approximation error when verifying nonlinear functions has been a long-standing research problem. State-of-the-art NN verifiers leverage interval propagation across layers. This technique is able to achieve both soundness and completeness when the NN is a piecewise linear function but would compromise either for networks with nonlinear operations. This is because interval propagation estimates affine maps from one layer to the next, leaving over-approximation, under-approximation or both when a nonlinear activation function is present. We want to show that our repair technique is not limited to piecewise linear policy networks – the NNs we choose for UUV and MC both contain tanh and sigmoid activations. Therefore, we have to sacrifice either soundness or completeness in verification. Among all the verifiers, we pick Verisig (Ivanov et al., 2019) as it only induces under-approximation in safe interval propagation. No false positives could occur, and soundness is guaranteed.

Conclusions

Neural network-based control policies have been widely adopted in cyber-physical systems, and one yet-to-be-addressed issue is repairing them while preserving safety. That is, when a policy produces behaviors that violate a safety specification, we need to fix such erroneous behaviors while keeping the existing correct behaviors still correct. This paper formalizes the repair with preservation (RwP) problem, which requires preserving the verifiable safety of initial state regions while improving the STL robustness of the incorrect ones. To address this problem, we propose ISAR, which repairs regions with failures one by one via simulated annealing with a safeguarding mechanism. We compare ISAR to state-of-the-art baselines of (1) reinforcement learning guided by formal specifications, (2) imitation learning with MPC-generated data and (3) online tube MPC shielding. Results show that ISAR performs significantly better than the baselines by preserving all regions with verified safety while making repairs to the policy by correcting 81.4% and 23.5% of the underwater vehicle and mountain car benchmarks, respectively.

Note

1 Due to the incompleteness of the verifier, there may exist some regions that do not pass verification, but no failures can be found. We do not know their safety for sure, and thus they are neither protected nor repaired.

Data availability statement. Our experiment source code is available at and will be maintained on this GitHub repository: https://github.com/ericlupy/isar_rep. Please note that the data is randomly generated in this paper, and no fixed dataset is used.

Author contribution. Pengyuan Lu has led the method design, analysis, implementation, experiments and the writing of the initial draft. Matthew Cleaveland has contributed to editing, conceptualization and formal analysis. Oleg Sokolsky and Insup Lee have contributed to advising, editing and funding acquisition. Ivan Ruchkin led the administration and has contributed to the advising, conceptualization and editing of this paper.

Financial support. This research was supported in part by NSF awards CNS 2143274 and CCF 2403616 and by ARO grant W911NF-20-1-0080. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF), the Army Research Office (ARO) or the United States Government.

Competing interests. The authors do not have any conflicts of interest to report for this paper.

Ethics statement. There are no ethical concerns in this paper. The experiments are run on simulations without any human subjects. All literature mentioned in this paper is properly cited.

Connections references

Nicola Paoletti and Jim Woodcock. How to ensure safety of learning-enabled cyber-physical systems? In *Research Directions: Cyber-Physical Systems*, volume 1, pages e2. Cambridge University Press, 2023. https://doi.org/10.1017/cbp.2023.2.

References

Aksaray D, Jones A, Kong Z, Schwager M, and Belta C (2016) Q-learning for robust satisfaction of signal temporal logic specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, IEEE, pp. 6565–6570.

Andersson JAE , Gillis J, Horn G, Rawlings JB, and Diehl M (2019) Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* **11**, 1–36.

Argall BD, Chernova S, Veloso M, and Browning B (2009) A survey of robot learning from demonstration. *Robotics and Autonomous Systems* **57**(5), 469–483.

Arulkumaran K, Deisenroth MP, Brundage M, and Anthony Bharath A (2017) Deep reinforcement learning: a brief survey. *IEEE Signal Processing Magazine* **34**(6), 26–38, .

Bastani O (2021) Safe reinforcement learning with nonlinear dynamics via model predictive shielding. In *2021 American control conference (ACC)*, IEEE, pp. 3488–3494.

Bauer-Marquart F, Boetius D, Leue S, and Schilling C (2022) Specrepair: Counter-example guided safety repair of deep neural networks. In *International Symposium on Model Checking Software*, Springer, pp. 79–96.

Bozkurt AK, Wang Y, Zavlanos MM, and Pajic M (2020) Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 10349–10355.

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, and Zaremba W (2016) Openai gym. arXiv preprint arXiv:1606.01540.

Carbonell JG, Michalski RS, and Mitchell TM (1983) An overview of machine learning. *Machine learning*, pp. 3–23.

Cohen D and Strichman O (2022) Automated repair of neural networks. arXiv preprint arXiv:2207.08157.

Hertog DD, Roos C, and Terlaky T (1992) On the classical logarithmic barrier function method for a class of smooth convex programming problems. *Journal of Optimization Theory and Applications* **73**(1), 1–25, .

Ding X, Smith SL, Belta C, and Rus D (2014) Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control* **59**(5), 1244–1257.

Fainekos GE and Pappas GJ (2009) Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* **410**(42), 4262–4291.

Fritz C (2003) Constructing büchi automata from linear temporal logic using simulation relations for alternating büchi automata. *In International Conference on Implementation and Application of Automata*, Springer, pp. 35–48.

Fu F and Li W (2021) Sound and complete neural network repair with minimality and locality guarantees. arXiv preprint arXiv:2110.07682.

Fu F, Wang Z, Fan J, Wang Y, Huang C, Chen X, Zhu Q, and Li W (2022) Reglo: Provable neural network repair for global robustness properties. In *Workshop on Trustworthy and Socially Responsible Machine Learning*, NeurIPS 2022.

Fujimoto S, Hoof H, and Meger D (2018) Addressing function approximation error in actor-critic methods. In *International conference on Machine Learning*, PMLR, pp. 1587–1596.

Gilpin Y, Kurtz V, and Lin H (2020) A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters* **5**(1), 241–246.

Goldberger B, Katz G, Adi Y, and Keshet J (2020) Minimal modifications of deep neural networks using verification. In *LPAR*, volume 2020, p 23rd.

Haarnoja T, Zhou A, Abbeel P, and Levine S (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, PMLR, pp. 1861–1870.

Hamilton N, Robinette PK, and Johnson TT (2022) Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning. In *International Conference on Software Engineering and Formal Methods*, Springer, pp. 190–206.

Hasan O and Tahar S (2015) Formal verification methods. In *Encyclopedia of Information Science and Technology*, Third Edition, IGI global, pp. 7162–7170.

Hauser J and Saccon A (2006) A barrier function method for the optimization of trajectory functionals with constraints. In *Proceedings of the 45th IEEE Conference on Decision and Control*, IEEE, pp. 864–869.

Hessel M, Soyer H, Espeholt L, Czarnecki W, Schmitt S, and Van Hasselt H (2019) Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume **33**, pp. 3796–3803.

Holland J, Kingston L, McCarthy C, Armstrong E, O'Dwyer P, Merz F, and McConnell M (2021) Service robots in the healthcare sector. *Robotics* **10**(1), 47.

Hussein A, Gaber MM, Elyan E, and Jayne C (2017) Imitation learning: a survey of learning methods. *ACM Computing Surveys (CSUR)* **50**(2), 1–35.

Ivanov R, Carpenter T, Weimer J, Alur R , Pappas G, and Lee I (2021) Verisig 2.0: Verification of neural network controllers using Taylor model preconditioning. In *International Conference on Computer Aided Verification*, Springer, pp. 249–262.

Ivanov R, Weimer J, Alur R, Pappas GJ, and Lee I (2019) Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 169–178.

Jazdi N (2014) Cyber physical systems in the context of industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, IEEE, pp. 1–4.

Kaelbling LP, Littman ML, and Moore AW (1996) Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* **4**, 237–285.

Kendall A, Gal Y, and Cipolla R (2018) Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7482–7491.

Kirkpatrick S, Gelatt CD Jr, and Vecchi MP (1983) Optimization by simulated annealing. *Science* **220**(4598), 671–680.

Li S and Bastani O (2020) Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 7166–7172.

Liu B (2017) Lifelong machine learning: a paradigm for continuous learning. *Frontier of Computer Science* **11**(3), 359–361.

Lu P, Cleaveland M, Sokolsky O, Lee I, and Ruchkin I (2024) Repairing learning-enabled controllers while preserving what works. In *2024 ACM/IEEE 15th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, pp. 1–11.

Lu P, Ruchkin I, Cleaveland M, Sokolsky O, and Lee I (2023) Causal repair of learning-enabled cyber-physical systems. In *2023 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 1–10.

Lyu D, Song J, Zhang Z, Wang Zhijie, Zhang T, Ma L, and Zhao J (2023) Autorepair: Automated repair for ai-enabled cyber-physical systems under safety-critical conditions. arXiv preprint arXiv:2304.05617.

Majd K, Clark G, Khandait T, Zhou S, Sankaranarayanan S, Fainekos G, and Ben Amor H (2023) Safe robot learning in assistive devices through neural network repair. arXiv preprint arXiv:2303.04431.

Majd K, Zhou S, Amor HB, Fainekos G, and Sankaranarayanan S (2021) Local repair of neural networks using optimization. arXiv preprint arXiv:2109.14041.

Maler O and Nickovic D (2004) Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer, pp. 152–166.

Mitchell TM (1997) Machine learning.

Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, and Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, PMLR, pp. 1928–1937.

Mukund M (1997) Linear-time temporal logic and büchi automata. Tutorial talk, Winter School on Logic and Computer Science, Indian Statistical Institute, Calcutta, page 8.

Parisotto E, Ba JL, and Salakhutdinov R (2016) Actor-mimic: Deep multitask and transfer reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Pnueli A (1977) The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE, pp. 46–57.

Polyak R (1992) Modified barrier functions (theory and methods). *Mathematical Programming* **54**, 177–222.

Preacher KJ and Selig JP (2012) Advantages of Monte Carlo confidence intervals for indirect effects. *Communication Methods and Measures* **6**(2), 77–98.

Ravichandar H, Polydoros AS, Chernova S, and Billard A (2020) Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems* **3**(1), 297–330.

Robert CP, Casella G, Robert CP, and Casella G (1999) Monte Carlo integration. *Monte Carlo Statistical Methods* 71–138.

Ruchkin I, Cleaveland M, Ivanov R, Lu P, Carpenter T, Sokolsky O, and Lee I (2022) Confidence composition for monitors of verification assumptions. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, pp. 1–12.

Rusu AA, Colmenarejo SG, Gulcehre C, Desjardins G, Kirkpatrick J, Pascanu R, Mnih V, Kavukcuoglu K, and Hadsell R (2016) Policy distillation. In *International Conference on Learning Representations (ICLR)*.

Ruvolo P and Eaton E (2013) Active task selection for lifelong machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence* **27**(1), 862–868.

Ruvolo P and Eaton E (2013) Active task selection for lifelong machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 862–868.

Sadigh D, Kim ES, Coogan S, Sastry SS , and Seshia SA (2014) A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, IEEE, pp. 1091–1096.

Sammut C, Hurst S, Kedzier D, and Michie D (1992) Learning to fly. In *Machine Learning Proceedings 1992*, Elsevier, pp. 385–393.

Schaal S. (1996) Learning from demonstration. *Advances in Neural Information Processing Systems* **9**, 1040–1046.

Schulman J, Wolski F, Dhariwal P, Radford A , and Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Serafini P. (1994) Simulated annealing for multi objective optimization problems. In *Multiple Criteria Decision Making: Proceedings of the Tenth International Conference: Expand and Enrich the Domains of Thinking and Application*, Springer, pp. 283–292.

Sickert S (2015) Converting linear temporal logic to deterministic (generalised) Rabin automata. *Archive of Formal Proofs*.

Sickert S, Esparza J, Jaax S, and Křetínský J (2016) Limit-deterministic büchi automata for linear temporal logic. In *International Conference on Computer Aided Verification*, Springer, pp. 312–332.

Sohn J, Kang S, and Yoo S (2019) Search based repair of deep neural networks. arXiv preprint arXiv:1912.12463.

Sotoudeh M and Thakur AV (2021) Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 588–603.

Suman B and Kumar P (2006) A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society* **57**, 1143–1160.

Thrun S (1998) Lifelong learning algorithms. *Learning to Learn* **8**, 181–209.

Tokui S, Tokumoto S, Yoshii A, Ishikawa F , Nakagawa T, Munakata K, and Kikuchi S (2022) Neurecover: Regression-controlled repair of deep neural networks with training history. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, pp. 1111–1121.

Usman M, Gopinath D, Sun Y, Noller Y, and Pasareanu CS (2021) Nn repair: Constraint-based repair of neural network classifiers. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*, Springer, pp. 3–25.

Van de Ven GM and Tolias AS (2018) Three continual learning scenarios. NeurIPS Continual Learning Workshop. Page 4. Vol. **1**. No. 9.

Vardi MY. (2005) An automata-theoretic approach to linear temporal logic. *Logics for Concurrency: Structure Versus Automata*, pp. 238–266.

Venkataraman H, Aksaray D, and Seiler P (2020) Tractable reinforcement learning of signal temporal logic objectives. In *Learning for Dynamics and Control*, PMLR, pp. 308–317.

Wabersich KP, Hewing L, Carron A, and Zeilinger MN (2021) Probabilistic model predictive safety certification for learning-based control. *IEEE Transactions on Automatic Control* **67**(1), 176–188.

Wabersich KP and Zeilinger MN (2018) Linear model predictive safety certification for learning-based control. In *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, pp. 7130–7135.

Wiering MA and Van Otterlo M (eds) (2012) *Reinforcement Learning: State-of-the-Art. Series: Adaptation, Learning, and Optimization*, Vol. **12**. Berlin/Heidelberg: Springer.

Yang X, Yamaguchi T, Tran H-D, Hoxha B, Johnson TT, and Prokhorov D (2022) Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*, Springer, pp. 221–236.

Zhou W, Gao R, Kim B, Kang E, and Li W. (2020) Runtime-safety-guided policy repair. In *Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings 20*, Springer, pp. 131–150.

Zhou X, Ahmed B, Aylor JH, Asare P, and Alemzadeh H (2023) Hybrid knowledge and data driven synthesis of runtime monitors for cyber-physical systems. *IEEE Transactions on Dependable and Secure Computing* **21**(1), 12–30.