Aligning Neural Network Controllers of CPS with System-Level Specifications via Optimization-Based Repair

Anonymous submission

Abstract

Cyber-Physical Systems (CPSs) have been widely deployed in safety-critical domains such as transportation, power and energy. Recently, there comes an increasing demand in employing neural networks (NNs) in CPSs for more intelligent control, giving birth to AI-enabled CPSs (AI-CPSs). Due to the uninterpretable decision logic of NNs, concerns about the safety misalignment between NN controllers and system-level specifications are also surging. In this work, we propose a framework OPERA that repairs the NN controllers to correct their unsafe behaviors. OPERA first diagnoses the unsafe system behaviors by monitoring the system executions. Then, for each unsafe system execution, OPERA searches in the control decision space for a *patch* of the control decisions that can ensure the safe behavior of the system under the same condition. To repair the system, OPERA retrains the NN controllers using a dataset with the control patches incorporated. We conduct a comprehensive evaluation for OPERA on 6 instances of AI-CPSs from different safety-critical domains. Evaluation results show that OPERA can effectively repair the critical safety issues in NN controllers, and significantly improve the reliability of the systems.

1 Introduction

Ensuring that AI systems are aligned with human safety requirements and ethical standards is increasingly important, given the trend of integrating AI products into systems in safety-critical domains (Ji et al. 2023). Modern Cyber-Physical Systems (CPSs) are a noteworthy example, which are increasingly adopting neural networks (NNs) as controllers, giving birth to AI-enabled CPSs (AI-CPSs). Examples of such CPSs include *adaptive cruise control systems*, *wind turbine systems*, *smart grids*, etc. Typically, an AI-CPS consists of a physical plant and an NN controller—the NN controller makes control decisions depending on the state of the plant and the external environment, and the plant executes the control decisions to trigger a state transition.

Despite the trend of adopting NN controllers in CPSs, a surge of concerns arise due to the inherent opacity of NNs. As CPSs are deployed in safety-critical domains, misalignment between NN controllers and system safety requirements can lead to improper control decisions and catastrophic system failures, potentially causing intolerable losses. Most of the existing research efforts have been paid to the testing of CPS (e.g., *optimization-based falsification* (Annpureddy et al. 2011; Donzé 2010; Ernst et al. 2022)), which aims to detect those dangerous cases. However, to reduce the safety risks caused by improper control decisions, it is necessary to diagnose and repair the unsafe system behaviors after finding them out, which has not received much attention.

There have been established works (Sohn, Kang, and Yoo 2023; Wang, Ustun, and Calmon 2019; Zhang and Chan 2019) in the repair of NNs, typically, in the domain of image classification. These works often exploit the ground truths (i.e., the true labels) of the wrongly classified images to retrain or tune the parameters of the NNs. However, these techniques are not directly applicable to the repair of AI-CPS, because, firstly, there is no such ground truth that specifies the *desired* value of a control decision. Instead, the correctness of control decisions can be assessed by safety requirements at the system level that concern with the states of the physical plant under the control of an NN controller. Moreover, while system-level safety requirements can be used to assess the correctness of NN controller inferences, the satisfaction to them is decided jointly by a series of NN controller inferences, rather than a single inference in the case of image classifiers; this poses a great challenge to the diagnosis of wrong NN inferences that lead to safety violation.

Contributions. To cope with the challenges, in this paper, we propose OPERA, a framework for automated repair of AI-CPSs guided by system-level specifications. Targeting the problem of the lack of ground truth for control decisions in AI-CPSs, OPERA exploits a search-based algorithm to find a *control patch* that corrects the behavior of those safety-violated execution traces for retraining the NN controllers. Given a set of system-level specifications, OPERA first identifies unsafe system behaviors by monitoring system executions. Then, for an unsafe system execution, OPERA iteratively explores the control decision space and searches for an optimal *control patch* that corrects a segment of control decisions and enables the system to satisfy the system-level specifications. Finally, OPERA includes those repaired control inputs and outputs into a training dataset and performs robust retraining of the NN controller. In this way, an enhanced NN controller is obtained that corrects the unsafe system behavior and improves the reliability of the system.

In summary, we make the following contributions:

- First, we propose a safety alignment approach called OPERA (*Optimization-based, sPEcification-guided Repair for Ai controllers*) for repairing NN controllers of CPS. We formulate the problem into a multi-objective optimization, and employ optimization solvers to search for the optimal patches for the control decisions, which ensures that the system execution after repair satisfies the safety requirement;
- Then, we implement OPERA as a prototype tool that can handle AI-CPSs in *Simulink*, the *de facto* standard of CPS modeling environment;
- We perform a comprehensive experimental evaluation on 6 instances of AI-CPSs. The evaluation results demonstrate the effectiveness of our approach.

2 Related Work

NN repair. Different repair mechanisms have been adopted to improve the performance and reliability of NN models. Existing work on NN repair can be roughly categorized into *adversarial training* and *parameter alteration*.

Adversarial training first collects adversarial examples and then uses these examples with corrected labels to retrain the model. It has been extensively studied in repairing image classifiers, e.g., (Goodfellow, Shlens, and Szegedy 2014; Wong and Kolter 2018; Shafahi et al. 2019; Ma et al. 2018; Yu et al. 2021; Gao et al. 2020). Although our work can be seen as a specific application of adversarial training, we differ in how we obtain labels for adversarial examples. Due to the complexity of AI-CPSs, we use optimization to find control signal corrections that ensure system-level safety.

Parameter alteration refers to the approaches that directly modify NNs' parameters to change the inference of adversarial examples (Sohn, Kang, and Yoo 2023; Wang, Ustun, and Calmon 2019; Zhang and Chan 2019; Calsi et al. 2023). For instances, (Sohn, Kang, and Yoo 2023; Zhang and Chan 2019) attempt to localize the suspicious NN parameters and then optimize them for NN repair.

Repair of NN controllers. There have been several attempts to repair NN controllers. Yang et al. (Yang et al. 2022) propose to first attack an NN controller and then retrain with the identified adversarial examples. Majd et al. (Majd et al. 2022) presents a controller repair technique based on Mixed Integer Quadratic Program that encodes temporal logic requirements. However, compared to (Yang et al. 2022) and (Majd et al. 2022) in which an isolated controller is considered, we target a combined system including a physical system controlled by an NN controller and moreover, our safety requirement is defined based on states of the physical plant, rather than the NN outputs. Another related work (Zhou et al. 2020) repairs a learning-based controller using an assumptive *safe* oracle. Compared to (Zhou et al. 2020), our approach automates the controller repair process, by searching corrections in the control decision space.

Safe reinforcement learning. *Reinforcement learning (RL)* is a common approach for training controllers. In RL, there is a line of work (Alshiekh et al. 2018; Anderson et al. 2020; Zhu et al. 2019; Berkenkamp et al. 2017) that aims to ensure the safety of the trained controllers; in particular, *shielding* (Alshiekh et al. 2018) offers a mechanism similar to our framework for repairing unsafe actions. However, there are key differences. First, many of the problem settings in RL are bounded to *Markov decision processes (MDP)*, which involve finitely many states and transitions; in contrast, we consider the control of non-linear continuous plants with dense-time real-valued dynamics, which are intrinsically hard to verify. Moreover, the safety mechanisms like shielding can be done at runtime—when an unsafe action is detected, a shield can be activated to stop it; in our case, banning a single action at runtime does not prevent unsafety from happening, due to the systems' *dense-time* nature. This is why we analyze and repair complete signals offline.

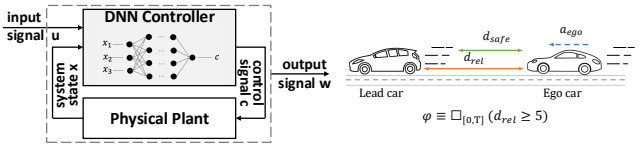

Figure 1: The architecture of a typical AI-CPS and an Adaptive Cruise Control (ACC) system

3 Preliminaries

In this section, we overview AI-CPSs (Section 3.1), NN controllers (Section 3.2), and system-level specifications expressed in *Signal Temporal Logic (STL)* (Section 3.3).

3.1 AI-Enabled Cyber-Physical Systems

Fig. 1 shows a typical system architecture of AI-CPSs. The system consists of a physical *plant* and an *NN controller*. The plant may involve complex non-linear continuous dynamics, or even confidential modules, and thus is deemed as a black box. The NN controller, at each moment t, takes as input the system state $\mathbf{x}(t)$ and an external input $\mathbf{u}(t)$; then, it issues a control decision $\mathbf{c}(t)$ to actuate the state evolution of the plant. During the execution of the system in $[0, T]$ ($T \in \mathbb{R}_{\geq 0}$), $\mathbf{u}, \mathbf{w}, \mathbf{x}, \mathbf{c}$ are all *time-variant* traces (called *signals*), which maps a time instant $t \in [0, T]$ to a vector. In particular, the control signal $\mathbf{c} : [0, T] \to \mathbb{R}$ is a one-dimensional trace, bounded by a closed interval $\mathcal{B}_{\mathbf{c}} \subseteq \mathbb{R}$, i.e., $\forall t \in [0, T] \,.\, \mathbf{c}(t) \in \mathcal{B}_{\mathbf{c}}$.

3.2 NN Controllers and Their Training

Traditional CPS controllers, such as those based on *PID* (Johnson and Moradi 2005) and *MPC* (Camacho and Alba 2013), employ human-crafted heuristics or linear models to make control decisions. Recently, NN has been increasingly investigated in industrial contexts as an alternative to traditional control algorithms, in handling sophisticated conditions in the real world (Liu, Mehdipour, and Belta 2021; Deshmukh et al. 2019; Yaghoubi, Fainekos, and Sankaranarayanan 2020; Nivison and Khargonekar 2018; Gilra and Gerstner 2018; Ferlez, Sun, and Shoukry 2020). Currently, two approaches are usually adopted to

train NN controllers, namely, *Deep Reinforcement Learning (DRL)* (Arulkumaran et al. 2017; Song et al. 2022) and *Supervised Learning* (Hertneck et al. 2018; Liu, Mehdipour, and Belta 2021):

- DRL allows a trainee to explore in the action space; the trainee will get rewarded if it selects "good" actions, and get penalized if it selects "bad" actions. Finally the optimal policy will be obtained by the trainee, in the form of an NN, by iteratively performing *"trial and error"*.
- In supervised learning, the trainee learns from the historical or experience data collected in the CPS, which involve quantities of pairs of the controller inputs $\langle \mathbf{x}, \mathbf{u} \rangle$ and the corresponding expected controller outputs $\mathbf{c}$. The training phase consists in optimizing the NN parameters in order to maximize the accuracy of predicting $\mathbf{c}$ given $\langle \mathbf{x}, \mathbf{u} \rangle$.

As NN controllers are the main component for decision making in AI-CPSs, we attribute the unsafe behavior of the system to the improper control decisions made by NN controllers. In this paper, we consider the NN controller repair in the CPS development and maintenance contexts under the assumption of the awareness of NN internals, in order to assist the engineers to identify and repair the potential safety issues at an early stage. In other words, the engineers can access the internals of an NN controller, so that with more repaired signal data collected either by our techniques or from practical CPS physical environment, it is possible to repair and enhance the safety behavior of CPS, e.g., by retraining or fine-tuning the NN controller, for continuous delivery. Moreover, we assume that the signals of $\mathbf{u}, \mathbf{w}, \mathbf{x}, \mathbf{c}$ are all observable.

3.3 System-Level Specification

We adopt *Signal Temporal Logic (STL)* (Maler and Nickovic 2004) to describe system specifications of AI-CPSs. STL has been a widely-accepted formalism (Ernst et al. 2020, 2021) for expressing the users' desired properties. It extends the classic *linear temporal logic (LTL)* (Pnueli 1977) in the discrete time setting to continuous time and space domain, and is equipped with a quantitative semantics, called *robustness*, that not only tells *whether* a property is satisfied, but also how much it is satisfied. The STL robust semantics has become the technical fundamentals of many other safety assurance techniques, such as *runtime verification* (Donzé, Ferrere, and Maler 2013; Deshmukh et al. 2017; Jakšić et al. 2018; Zhang et al. 2023) and *falsification* (Annpureddy et al. 2011; Donzé 2010). The detailed syntax and robust semantics of STL are introduced in Appendix A.

4 The OPERA Framework

In this paper, we propose a framework, named OPERA, to enhance NN controllers for system safety. Specifically, OPERA (i) identifies unsafe system behaviors in a system execution, (ii) repairs them by searching for a patch of control decisions to ensure that the system satisfies safety requirements during the given execution, and (iii) repairs the NN controller with the safe execution trace data. We first illustrate the intuition of system repair in Example 4.1, and then introduce the workflow of OPERA.

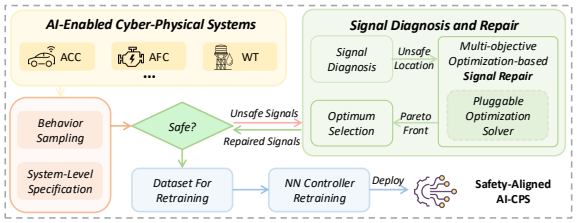

Figure 2: The workflow of the proposed framework OPERA

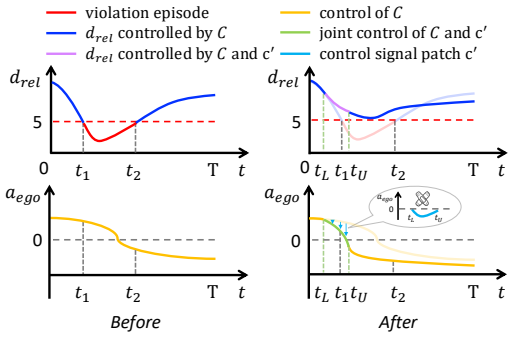

Figure 3: An illustrative example of signal repair for ACC.

Example 4.1. Consider the *Adaptive Cruise Control (ACC)* scenario in the right half of Fig. 1, in which an ego car tries to follow a lead car within a safe distance. We treat the ego car as the AI-enabled system under consideration, and the lead car as the external environment. The controller of the ego car decides its acceleration a_{ego} based on the sensor data, including the states of the ego car and the lead car. The system-level specification φ requires that, the relative distance d_{rel} between the two cars should always be greater than a safe distance of 5.

$$\varphi \equiv \square_{[0,T]}(d_{rel} > 5)$$

Fig. 3 shows an example of how OPERA works to repair a system execution that violates the safety requirement φ. Before repair, during $[t_1, t_2]$, d_{rel} is smaller than 5, thus violating φ. This is because the ego car did not manage to make a timely deceleration, which could have prevented the safety violation from happening. The task of repair thus consists in correcting the control signal a_{ego} to bring it down earlier before t_1 such that the safety violation can be avoided.

OPERA achieves this by using a *patch* of control decisions (as indicated by the blue segment in Fig. 3) applied as an intervention to the control signal of the original NN controller, which results in a joint control by the original NN controller and the patch. After repair, the ego car decelerates timely and avoids the safety violation. Now, this new execution trace, including the new control signals, will be used as training data to repair the NN controller. □

Workflow. Given a CPS with an NN controller, OPERA first runs the system and performs behavior sampling to obtain execution traces, each of which includes a system input signal $\mathbf{u}$, a system output signal $\mathbf{w}$, a control decision signal

Algorithm 1 The optimization-based signal repair

Require: Original system with an NN controller $\mathcal{C}$, a safety requirement φ

1: **for** $\mathbf{u}$ sampled from the input space **do**
2: $\quad \mathbf{w}, \mathbf{c}, \mathbf{x} \leftarrow \textsc{SysExec}(\mathbf{u}, \mathcal{C})$ ▷ system execution under $\mathcal{C}$
3: $\quad \textsc{DiagnoseAndRepair}(\mathbf{u}, \mathbf{w}, \mathbf{c}, \mathbf{x})$ ▷ repair violating signals
4: $\quad$ add $(\langle \mathbf{x}, \mathbf{u} \rangle, \mathbf{c})$ to the retraining dataset ▷ retraining

5: **procedure** $\textsc{DiagnoseAndRepair}(\mathbf{u}, \mathbf{w}, \mathbf{c}, \mathbf{x})$
6: $\quad$ **if** $\texttt{rob}(\mathbf{w}, \varphi) < 0$ **then** ▷ the case that $\mathbf{w}$ is unsafe
7: $\quad\quad \begin{bmatrix} \tau_{\mathtt{S}} \leftarrow \text{the instant when violation episode starts} \\ \tau_{\mathtt{E}} \leftarrow \text{the instant when violation episode ends} \end{bmatrix}$ ▷ signal diagnosis
8: $\quad\quad \mathbf{c}' \leftarrow \arg \begin{bmatrix} \max\limits_{\mathbf{c}'} & \texttt{rob}\left(\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}, \varphi\right) \\ \min\limits_{\mathbf{c}'} & \|\mathbf{c}'\|_1 \\ \text{s.t.} & t^{\mathtt{L}}_{\mathbf{c}'} \in [0, \tau_{\mathtt{S}}] \\ & t^{\mathtt{U}}_{\mathbf{c}'} \in (t^{\mathtt{L}}_{\mathbf{c}'}, \tau_{\mathtt{E}}] \\ & \forall t \in \left[t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}\right] . \mathbf{c}'(t) \in \delta \cdot \mathcal{B}_{\mathbf{c}} \end{bmatrix}$ ▷ signal repair by search of control signal patch
9: $\quad\quad \mathbf{w}, \mathbf{c}, \mathbf{x} \leftarrow$ system exec. by joint control of $\mathcal{C}$ and $\mathbf{c}'$
10: $\quad\quad \textsc{DiagnoseAndRepair}(\mathbf{u}, \mathbf{w}, \mathbf{c}, \mathbf{x})$ ▷ continus repair

$\mathbf{c}$, and a system state signal $\mathbf{x}$. Then, given a safety requirement φ, OPERA checks whether the system output signal $\mathbf{w}$ in each execution trace satisfies the safety requirement. If unsafe signals are found, the signal diagnosis and repair procedure will be triggered to repair unsafe signals and produce a new safety-assured execution trace. This procedure will be detailed later in Section 5.

With the safety-assured execution traces, we can repair the original NN controller that issued improper control signals and led to unsafe system behavior. To be general, as an early step for NN controller repair, the NN repair module in OPERA is designed to be extensible (to diverse repairing methods). The basic mode is to retrain the NN controller with the new safety-assured execution traces to enhance its performance, while keep the remaining training configurations, e.g., the architecture of the NN controller, unchanged. In addition to the basic mode, other options include *Neural Architecture Search (NAS)* (Elsken, Metzen, and Hutter 2019), which can search over different NN configurations and identify the best model with the new execution traces. In this work, we experimentally evaluate the performance of the repaired NN controller obtained by retraining, and we leave the investigation of other options as future work.

5 Signal Diagnosis and Repair

In this section, we introduce the details of the key component of OPERA, i.e., signal diagnosis and repair, which aims to repair the control signals that lead to safety violation.

5.1 Algorithm Overview

Alg. 1 describes the signal repair process. It requires a CPS with an NN controller $\mathcal{C}$ and an STL safety requirement φ. At a high level, the algorithm samples in the input space of the system (Line 1) to check the system behavior (Line 2) under different conditions, and feed the system execution traces to the procedure of signal diagnosis and repair to correct the potential unsafe behaviors (Line 3). Then, the safety-assured execution traces will be added into a dataset for the retraining of the NN controllers (Line 4).

The procedure of signal diagnosis and repair starts with checking whether the output signal $\mathbf{w}$ satisfies the safety requirement φ (Line 6). If $\mathbf{w}$ is safe, then there is no need to repair it. Otherwise, the algorithm first localizes the starting and ending time instants, $\tau_{\mathtt{S}}$ and $\tau_{\mathtt{E}}$, that *cause* the safety violation of the output signal $\mathbf{w}$ (Line 7), by using a runtime monitoring technique called *signal diagnostics* (Zhang, Arcaini, and Xie 2022). Then, it searches for an optimal control signal patch $\mathbf{c}'$ under which the system output signal $\mathbf{w}$ until $\tau_{\mathtt{E}}$ is corrected to satisfy the safety requirement (Line 8). Formally, $\mathbf{c}'$ is a partial signal defined in the time interval $[t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}]$, where $t^{\mathtt{L}}_{\mathbf{c}'}$ is the *lower time bound* at which the control signal starts to be rectified, and $t^{\mathtt{U}}_{\mathbf{c}'}$ is the *upper time bound* at which the rectification can end. The details of deciding $\mathbf{c}'$ will be given in Section 5.2.

Next, we re-execute the system under the joint control of the original NN controller and the control signal patch $\mathbf{c}'$. Concretely, this execution is controlled as described below:

- Before $t^{\mathtt{L}}_{\mathbf{c}'}$, the control is given by the original NN controller $\mathcal{C}$;
- During $[t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}]$, the control is jointly performed by $\mathbf{c}'$ and the original controller $\mathcal{C}$;
- After $t^{\mathtt{U}}_{\mathbf{c}'}$, the control is switched back to the original NN controller $\mathcal{C}$.

By re-executing the system, we can obtain new $\mathbf{w}$, $\mathbf{c}$ and $\mathbf{x}$ (Line 9). In case that the new execution still violates the safety requirement (due to some new causes), this procedure is invoked recursively with the new $\mathbf{w}, \mathbf{c}$ and $\mathbf{x}$ to diagnose and repair the new unsafe behavior, until the entire system execution is safe. Finally, this new execution is returned to the OPERA framework and fed as input into the NN repair module to enhance the NN controller (Line 4).

5.2 Signal Diagnosis

A naive selection of the time range, in which we search for the control signal patch $\mathbf{c}'$ that repairs the unsafe system execution, is the entire time range $[0, T]$. However, it can lead to a search space that is too large, thus making the search of $\mathbf{c}'$ too expensive. In this section, we employ an STL monitoring technique called *signal diagnosis* (Zhang, Arcaini, and Xie 2022; Bartocci et al. 2018) to identify the *violation episode* in the system output signal, which can be seen as the *cause* of the safety violation. Intuitively, since the safety violation is caused by the violation episode, it suffices to repair the system behavior only in the violation episode. As the violation episode is only a segment of the whole signal, our search space of $\mathbf{c}'$ is significantly reduced.

Signal diagnosis is used to localize the *violation episode* in a signal. In general, signal diagnosis collects a set of

sufficient conditions for the safety violation of a signal. It achieves that by recursively collecting the time instants at which the Boolean verdicts of the atomic propositions lead to the violation of the given STL formula, based on the semantics of different STL operators. For instance, if $\varphi \equiv \Box_I \varphi'$ is violated, it must be due to that, at some instant $t \in I$, φ' is violated. In this case, signal diagnosis collects all the time instants, at which φ' is violated, as the violation episode. A more concrete example is given in Example 5.1. For more details, we refer the readers to (Zhang, Arcaini, and Xie 2022).

Example 5.1. In Example 4.1, the segment in $[t_1, t_2]$ could be considered as the cause of the violation, because d_{rel} keeps being less than 5 in this segment, which violates the atomic proposition $d_{rel} > 5$ of φ. As a result, the violation of the atomic proposition leads to the violation of φ, and therefore, this segment will be diagnosed as a violation episode. □

In Line 7 of Alg. 1, $\tau_{\mathtt{S}}$ is set as the timestamp when the violation episode starts, i.e., when the system output starts to violate the safety requirement; and $\tau_{\mathtt{E}}$ is set as the timestamp when the violation episode ends, i.e., when the system resumes back to a safe status. The correctness of using signal diagnosis to derive the time constraints of controller repair is based on the fact that the unsafe status caused by improper control decisions only propagate backward in the time domain, and therefore, only the time interval that precedes or matches the safety violation episode of the output signal needs to be considered for repair. See Lemma 5.2 for an explanation.

Lemma 5.2. *Let $\mathbf{c}'$ be a control signal patch defined in the interval $[t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}]$, and $\tau_{\mathtt{S}}$ and $\tau_{\mathtt{E}}$ be respectively the starting and the ending point of the violation episode. Then, the lower time bound $t^{\mathtt{L}}_{\mathbf{c}'}$ of $\mathbf{c}'$ should be not later than $\tau_{\mathtt{S}}$; and the upper time bound $t^{\mathtt{U}}_{\mathbf{c}'}$ of $\mathbf{c}'$ should be not later than $\tau_{\mathtt{E}}$.*

The proof of Lemma 5.2 is based on the definition of STL signal diagnosis, and the fact that safety violations only propagate backward in the time range.

If $t^{\mathtt{L}}_{\mathbf{c}'} > \tau_{\mathtt{S}}$, there will be an unsafe segment in $[\tau_{\mathtt{S}}, t^{\mathtt{L}}_{\mathbf{c}'}]$ that is not repaired by the control signal patch $\mathbf{c}'$. Therefore, the lower time bound $t^{\mathtt{L}}_{\mathbf{c}'}$ of $\mathbf{c}'$ should not be later than $\tau_{\mathtt{S}}$. Similarly, we can derive that the upper time bound $t^{\mathtt{U}}_{\mathbf{c}'}$ of $\mathbf{c}'$ should not be later than $\tau_{\mathtt{E}}$. □

Example 5.3. In Example 4.1, $\tau_{\mathtt{S}}$ and $\tau_{\mathtt{E}}$ are respectively computed as $\tau_{\mathtt{S}} = t_1$ and $\tau_{\mathtt{E}} = t_2$. Therefore, when we search for the control signal patch $\mathbf{c}'$, its lower time bound $t^{\mathtt{L}}_{\mathbf{c}'}$ should be not later than t_1, and its upper time bound $t^{\mathtt{U}}_{\mathbf{c}'}$ should be not later than t_2.

Intuitively, this is correct, because if $t^{\mathtt{L}}_{\mathbf{c}'}$ is later than t_1, then the safety violated interval $[\tau_{\mathtt{S}}, t^{\mathtt{L}}_{\mathbf{c}'}]$ cannot be repaired by $\mathbf{c}'$; if $t^{\mathtt{U}}_{\mathbf{c}'}$ is later than $\tau_{\mathtt{E}}$, then in $[\tau_{\mathtt{E}}, t^{\mathtt{U}}_{\mathbf{c}'}]$ there is no safety violation. □

5.3 Multi-Objective Optimization-Based Repair

There remains the problem of how to construct the control signal patch $\mathbf{c}'$, given the time constraint $[t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}]$. The desired properties of $\mathbf{c}'$ involve the following two aspects:

- First, by the intervention of $\mathbf{c}'$, the safety violation in $[\tau_{\mathtt{S}}, \tau_{\mathtt{E}}]$ should be resolved;
- Second, the intervention of $\mathbf{c}'$ should be minimal, in order to minimize the behavior change of the system and thus minimize the performance losses in other aspects than safety. Moreover, this can also avoid producing severe outlier control signals that harm the quality of training data.

In this paper, we search for such a control signal patch $\mathbf{c}'$ using *multi-objective optimization*. In general, multi-objective optimization consists of multiple objective functions; these functions may conflict with each other, i.e., optimizing one function may hinder another function. As a result, the outcome of multi-objective optimization is a set of optima, called *Pareto front*, rather than a single optimum. Intuitively, a Pareto front involves all the incomparable optimal solutions—it is often not the case that one solution is better than another one in terms of all the objective functions.

We formulate our problem of searching for $\mathbf{c}'$ as a multi-objective optimization problem, in which the two objective functions embody the two desired properties mentioned above. The multi-objective optimization problem is shown in Line 8 of Alg. 1:

- The first objective function is to maximize the satisfaction of the repaired output signal $\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}$ in $[0, \tau_{\mathtt{E}}]$ to the safety requirement. Note that $\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}$ is produced by the joint control of the original NN controller $\mathcal{C}$ and the control signal patch $\mathbf{c}'$, as stated in Section 5.1. The goal of this objective function is to find $\mathbf{c}'$ such that the satisfaction $\mathtt{rob}\left(\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}, \varphi\right)$ of the repaired signal $\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}$ is positive, and so $\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}$ satisfies the safety requirement φ.
- The second objective function is to minimize the intervention of the control signal patch $\mathbf{c}'$, which is measured by the L1-norm of $\mathbf{c}'$;
- The constraints include the time constraint in Section 5.2 and the space constraint in Section 3.1. Here, $\mathbf{c}'$ is spatially bounded by $\delta \cdot \mathcal{B}_{\mathbf{c}}$, where $\delta \in [0, 1]$ is a small value such that $\mathbf{c}'$ does not bring a big intervention. In our experiments, we set δ as 0.1.

Representation of $\mathbf{c}'$. An issue arises in handling the infinite search space for $\mathbf{c}'$ due to its continuity in time and space. Following the convention of CPS community (Ernst et al. 2020, 2021, 2022), we adopt a *parameterized representation* to denote $\mathbf{c}'$, so the continuous signal $\mathbf{c}'$ can be identified by a finite number of variables. Specifically, we select *piecewise linear* signal, i.e., given a hyperparameter k, $\mathbf{c}'$ is linear in each time interval $\left[t^{\mathtt{L}}_{\mathbf{c}'} + \frac{i(t^{\mathtt{U}}_{\mathbf{c}'} - t^{\mathtt{L}}_{\mathbf{c}'})}{k}, t^{\mathtt{L}}_{\mathbf{c}'} + \frac{(i+1)(t^{\mathtt{U}}_{\mathbf{c}'} - t^{\mathtt{L}}_{\mathbf{c}'})}{k}\right]$ ($i \in \{0, \ldots, k-1\}$). In that case, $\mathbf{c}'$ is identified by the variables $t^{\mathtt{L}}_{\mathbf{c}'}, t^{\mathtt{U}}_{\mathbf{c}'}$, and $\mathbf{c}'\left(t^{\mathtt{L}}_{\mathbf{c}'} + \frac{i(t^{\mathtt{U}}_{\mathbf{c}'} - t^{\mathtt{L}}_{\mathbf{c}'})}{k}\right)$ ($i \in \{0, \ldots, k-1\}$).

Solving the Optimization Problem. An efficient optimization solver is needed to solve the optimization problem formulated in Line 8. In our context, the evaluation of the objective $\mathtt{rob}\left(\mathbf{w}^*_{[0,\tau_{\mathtt{E}}]}, \varphi\right)$ requires system execution of the AI-

CPSs, whose dynamics could be hard to model. To handle this issue, we select *NSGA-II* (Deb et al. 2002), a genetic algorithm (GA)-based multi-objective optimization algorithm, as our solver, since NSGA-II does not require the awareness of the internal dynamics of the objective function. It computes a Pareto front by iteratively performing *mutation* and *crossover*, as other genetic algorithms do.

The Pareto front returned by the optimization solver contains a set of optimum solutions; having such a Pareto front, we need to further perform an *optimum selection* to select the solution used for signal repair. In OPERA, we make this selection process configurable, and here we provide five possible strategies, as follows. In RQ2 of Section 6, we experimentally compare their performances.

- `MinSat` selects the optimum that has a minimum satisfaction to the system specification;
- `MaxSat` selects the optimum that maximizes the satisfaction to the system specification;
- $\texttt{Similar}_{\texttt{T}}$ selects the solution with with the shortest intervention time;
- $\texttt{Similar}_{\texttt{I}}$ selects the solution with the minimal intervention in terms of L1-norm;
- `Balance` does not introduce any bias, but selects a solution randomly from the Pareto front, under the premise that it satisfies the system specification.

After the error-triggering signals are repaired, the next step is to leverage such signal data for the NN controller repair, as described in Section 4. In general, OPERA can be easily integrated into the development and maintenance of AI-CPSs for NN repair and enhancement, in order to improve their safety.

6 Experimental Evaluation

In this section, we provide the experimental evaluation of OPERA, which is publicly available (Anonymous Github 2025).

6.1 Research Questions

We identified the following research questions to assess OPERA:

- **RQ1: Does OPERA effectively improve the safety of AI-CPSs?** To answer RQ1, we first create the retraining dataset by sampling the execution traces of these systems, and apply OPERA to obtain the repaired systems with retrained NN controllers. We train each NN controller 10 times to reduce the effect of randomness and report the average results with standard deviation. For the testing purpose, we also create a test set for each system. Then, we execute each AI-CPS with the original controllers and with the repaired controllers respectively, by feeding them the input signals from the retraining dataset and the test set. We compare SR of these systems, which are measured as the ratio of the number of safe executions over the number of all the executions in the retraining dataset and the test set.
- **RQ2: How does the optimum selection strategy affect the effectiveness of OPERA?** To answer RQ1 and RQ2, we performed pairwise comparisons between SR of the five optimum selection strategies across all the benchmarks and all the dataset (i.e., training set and test set), using the Wilcoxon signed-rank test (Wilcoxon et al. 1970) and the Cohen's d effect size (Cohen 1969).
- **RQ3: Does OPERA affect the performance of AI-CPSs in terms of other metrics than safety?** To answer RQ3, we investigate the influence of OPERA on the performance of the AI-CPSs w.r.t. various system performance metrics. Specifically, we compare the performance of the repaired systems, by OPERA with the default optimum selection strategy `Balance`, with the original systems in terms of these metrics. Note that these system performance metrics are system-specific, since different systems are constrained by different requirements in their development. A detailed introduction to these metrics is given in Appendix C.4.

6.2 Benchmarks

We selected six AI-CPSs from three industrial domains as the subject systems for our evaluation, namely, *Adaptive Cruise Control* (ACC) (Tran et al. 2020), *Abstract Fuel Control* (AFC) (Jin et al. 2014), and *Water Tank* (WT) (Song et al. 2022). These systems span over different safety-critical industrial domains, such as autonomous driving and powertrain control. All these systems are built in Simulink, the *de facto* CPS modeling standard in industry. Table 3 in the Appendix C provides their functionalities, controller details and plant details.

NN controllers. For each subject CPS, we trained 2 NN controllers with different structures, and thus, we have 6 instances in total as the systems under evaluation. Our training dataset and approach follows the literature (Tran et al. 2020; Zhang et al. 2022). Table 3 provides their types, structures and training algorithms.

Retraining set and test set. For each instance of the subject systems, we randomly sample 10000 execution traces as the dataset for retraining, and 1000 traces as the test set. We list the safety rates (SR) of the systems with original NN controllers on the retraining dataset in Table 3, which uses $\frac{\#safe\ trials}{\#total\ trials}$ to represent how safe a system is. As illustrated in Table 3, the SR of the 6 instances to be repaired range from 82.0% to 98.1%.

6.3 Experimental Results

In this section, we present our evaluation results for OPERA. The more detailed experimental data and results are left in Appendix E.

RQ1: This RQ aims to evaluate whether OPERA can improve the safety of the subject AI-CPSs. The comparison on SR of the systems before and after repair is shown in Table 1. We notice that, compared to the original systems, the systems with the repaired controllers (under different optimum selection strategies) can improve SR significantly, ranging from 0.26% (WT#2) to 14.6 % (AFC#1) on the retraining set and from 0.26% (WT#2) to 12.6 % (AFC#1)

Table 1: The SR of the original systems, and the systems after repair under five optimum selection strategies (average $\pm$ standard deviation, %). The best performer for each benchmark is highlighted. (*Train*: the dataset for retraining; *Test*: the test set.)

		ACC#1		ACC#2		AFC#1		AFC#2		WT#1		WT#2	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Original		96.82	97.40	82.00	84.10	82.87	85.40	86.61	85.50	84.04	84.00	98.08	97.70
Repaired	`MinSat`	99.51 ± 0.85	99.52 ± 0.65	92.09 ± 7.71	94.67 ± 5.97	88.56 ± 3.68	89.71 ± 2.66	96.55 ± 2.26	96.04 ± 4.53	88.91 ± 6.29	88.48 ± 6.47	98.39 ± 0.65	98.07 ± 0.74
	`MaxSat`	98.76 ± 0.80	98.85 ± 0.77	92.85 ± 3.59	93.35 ± 3.35	96.99 ± 2.15	97.26 ± 1.95	95.53 ± 4.52	95.84 ± 4.59	95.41 ± 3.12	94.85 ± 3.41	98.55 ±0.12	98.22 ± 0.19
	$\texttt{Similar}_\texttt{T}$	99.45 ± 0.76	99.46 ± 0.56	90.81 ± 3.53	92.20 ± 2.79	96.37 ± 4.89	98.00 ± 1.43	98.60 ± 1.80	97.69 ± 3.57	89.90 ± 6.48	91.22 ± 6.88	98.34 ± 0.25	97.96 ± 0.31
	$\texttt{Similar}_\texttt{I}$	99.20 ± 1.01	99.29 ± 0.86	89.22 ± 3.57	90.66 ± 3.12	90.98 ± 4.88	92.40 ± 4.15	94.44 ± 4.11	95.16 ± 3.96	90.96 ± 7.01	90.58 ± 7.29	98.40 ± 0.65	97.99 ± 0.71
	`Balance`	99.06 ± 0.90	99.22 ± 0.74	91.08 ± 5.33	91.94 ± 4.91	97.47 ± 2.10	97.38 ± 2.16	97.68 ± 2.19	97.78 ± 2.14	91.67 ± 0.92	90.96 ± 0.89	98.44 ± 0.05	98.05 ± 0.08

Table 2: RQ1, RQ2 – Statistical comparison between each pair of approaches in terms of SR. (Legend. $\equiv$: there is no difference between the two approaches. ✓, ✓✓, ✓✓✓: the row approach is significantly better than the column approach with strength *small*, *medium*, *large*. ✗, ✗✗, ✗✗✗: the row approach is significantly worse than the column approach with strength *small*, *medium*, *large*)

	`MinSat`	`MaxSat`	$\texttt{Similar}_\texttt{T}$	$\texttt{Similar}_\texttt{I}$	`Balance`	Original
`MinSat`	$\equiv$	✗✗	✗✗	✓	✗✗	✓✓✓
`MaxSat`	✓✓	$\equiv$	✓✓	✓✓✓	✓✓	✓✓✓
$\texttt{Similar}_\texttt{T}$	✓✓	✗✗	$\equiv$	✓✓	✗	✓✓✓
$\texttt{Similar}_\texttt{I}$	✗	✗✗✗	✗✗	$\equiv$	✗✗✗	✓✓✓
`Balance`	✓✓	✗✗	✓	✓✓✓	$\equiv$	✓✓✓
Original	✗✗✗	✗✗✗	✗✗✗	✗✗✗	✗✗✗	$\equiv$

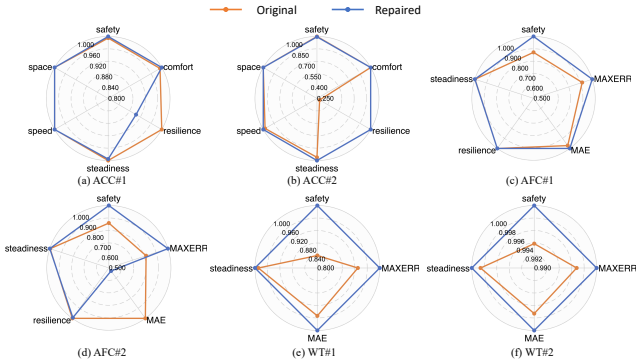

Figure 4: Affection of OPERA to system performances.

on the test set, respectively. These results exhibit the effectiveness of OPERA in repairing the unsafe behaviors, and moreover, the safety improvements on the test sets demonstrate the generalization ability of OPERA.

RQ2: In this RQ, we compare the performances of OPERA under the five optimum selection strategies introduced in Section 5.3.

Table 2 shows that `MaxSat` is the best performer, due to its selection to the optimum with the maximum satisfaction of system specifications. `Balance` is the second best, and this is reasonable, because `Balance` does not introduce any bias when selecting the optimum. $\texttt{Similar}_\texttt{I}$ has a similar performance with $\texttt{Similar}_\texttt{T}$. These results show that $\texttt{Similar}_\texttt{T}$ and $\texttt{Similar}_\texttt{I}$ are relatively weak in changing the characteristics of NN controllers, and this could be due to that they prefer the control signal patches that have the least deviation from the original control signals. Note that the repaired controllers by all of the five strategies are better than the original controller, which exhibits the effectiveness of OPERA.

RQ3: The affection of OPERA to other performances than safety of the systems is depicted in Fig. 4. Specifically, we normalize the performance data in terms of different metrics, and compare the performances of the original systems (orange lines) and the repaired systems (blue lines).

As illustrated in Fig. 4, compared to the original systems, the repaired systems do not suffer much from performance degradation. For example, in ACC#1, while OPERA improves the safety of the original system, it barely harms the *space* performance (i.e., how far the ego travels) or the *speed* performance (i.e., the speed should not be too low in a safe status). This is due to that, OPERA minimizes the intervention of the control signal patch $\mathbf{c}'$ to the original control signal $\mathbf{c}$. It also happens that the safety improvement comes with performance degradation in some aspects (e.g., *MAE* performance for AFC#2). The severity of such degradation depends on the importance of these metrics.

Moreover, we observe that OPERA can also improve other performance metrics together with safety, e.g., *steadiness* for WT#1. This is because safety and other performance metrics can be aligned in some cases. For example, improving safety can also enhance *steadiness*, as the *steadiness* metric requires the system to remain safe for a period.

7 Conclusion and Future Work

While AI-CPSs have received increasing attention, how to align the behavior of NN controllers of CPS with system-level specifications is an ongoing challenge. In this work, we propose OPERA, an optimization-based repair framework that accounts for system-level specifications during repairing. OPERA adopts a novel signal diagnosis and repair method that, given a system execution trace, identifies the minimal intervention to the control signals of the original NN controllers, to avoid unsafe system outputs. By applying the optimal intervention, OPERA obtains a new safety-assured execution trace, which can be used to repair the initial NN controller. Our experiments demostrate the effectiveness of OPERA.

In future, we plan to extend our framework to support the safety requirements specified by more diverse advanced temporal logic formalism towards providing better safety assurance and enhancement to accelerate the NN adoption in diverse industrial CPS across domains.

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Anderson, G.; Verma, A.; Dillig, I.; and Chaudhuri, S. 2020. Neurosymbolic reinforcement learning with formally verified exploration. *Advances in neural information processing systems*, 33: 6172–6183.

Annpureddy, Y.; Liu, C.; Fainekos, G.; and Sankaranarayanan, S. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In Abdulla, P. A.; and Leino, K. R. M., eds., *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2011*, volume 6605 of *Lecture Notes in Computer Science*, 254–257. Springer.

Anonymous Github. 2025. Supplementary Material for the paper "Aligning Neural Network Controllers of CPS with System-Level Specifications via Optimization-Based Repair". https://anonymous.4open.science/r/OPERA-B02D/.

Arcuri, A.; Iqbal, M. Z.; and Briand, L. 2012. Random Testing: Theoretical Results and Practical Implications. *IEEE Transactions on Software Engineering*, 38(2): 258–277.

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

Bartocci, E.; Ferrère, T.; Manjunath, N.; and Ničković, D. 2018. Localizing faults in Simulink/Stateflow models with STL. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, 197–206.

Berkenkamp, F.; Turchetta, M.; Schoellig, A.; and Krause, A. 2017. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30.

Calsi, D. L.; Duran, M.; Zhang, X.-Y.; Arcaini, P.; and Ishikawa, F. 2023. Distributed Repair of Deep Neural Networks. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 83–94. IEEE.

Camacho, E. F.; and Alba, C. B. 2013. *Model predictive control*. Springer science & business media.

Cohen, J. 1969. *Statistical power analysis for the behavioral sciences*. New York: Academic Press.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2): 182–197.

Deshmukh, J. V.; Donzé, A.; Ghosh, S.; Jin, X.; Juniwal, G.; and Seshia, S. A. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1): 5–30.

Deshmukh, J. V.; Kapinski, J. P.; Yamaguchi, T.; and Prokhorov, D. 2019. Learning deep neural network controllers for dynamical systems with safety guarantees. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–7. IEEE.

Donzé, A. 2010. Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, CAV'10, 167–170. Berlin, Heidelberg: Springer-Verlag. ISBN 364214294X.

Donzé, A.; Ferrere, T.; and Maler, O. 2013. Efficient robust monitoring for STL. In *International Conference on Computer Aided Verification*, 264–279. Springer.

Donzé, A.; and Maler, O. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 92–106. Springer.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1): 1997–2017.

Ernst, G.; Arcaini, P.; Bennani, I.; Chandratre, A.; Donzé, A.; Fainekos, G.; Frehse, G.; Gaaloul, K.; Inoue, J.; Khandait, T.; Mathesen, L.; Menghi, C.; Pedrielli, G.; Pouzet, M.; Waga, M.; Yaghoubi, S.; Yamagata, Y.; and Zhang, Z. 2021. ARCH-COMP 2021 Category Report: Falsification with Validation of Results. In Frehse, G.; and Althoff, M., eds., *Proceedings of the 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, 133–152. EasyChair.

Ernst, G.; Arcaini, P.; Bennani, I.; Donzé, A.; Fainekos, G.; Frehse, G.; Mathesen, L.; Menghi, C.; Pedrielli, G.; Pouzet, M.; Yaghoubi, S.; Yamagata, Y.; and Zhang, Z. 2020. ARCH-COMP 2020 Category Report: Falsification. In Frehse, G.; and Althoff, M., eds., *Proceedings of the 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, 140–152. EasyChair.

Ernst, G.; Arcaini, P.; Fainekos, G.; Formica, F.; Inoue, J.; Khandait, T.; Mahboob, M. M.; Menghi, C.; Pedrielli, G.; Waga, M.; Yamagata, Y.; and Zhang, Z. 2022. ARCH-COMP 2022 Category Report: Falsification with Ubounded Resources. In Frehse, G.; Althoff, M.; Schoitsch, E.; and Guiochet, J., eds., *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90 of *EPiC Series in Computing*, 204–221. EasyChair.

Ernst, G.; Hasuo, I.; Zhang, Z.; and Sedwards, S. 2018. Time-Staging Enhancement of Hybrid System Falsification. *CoRR*, abs/1803.03866.

Ferlez, J.; Sun, X.; and Shoukry, Y. 2020. Two-level lattice neural network architectures for control of nonlinear systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, 2198–2203. IEEE.

Gao, X.; Saha, R. K.; Prasad, M. R.; and Roychoudhury, A. 2020. Fuzz Testing Based Data Augmentation to Improve Robustness of Deep Neural Networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, 1147–1158. New York, NY, USA: Association for Computing Machinery. ISBN 9781450371216.

Gill, P. E.; Murray, W.; and Wright, M. H. 2019. *Practical optimization*. SIAM.

Gilra, A.; and Gerstner, W. 2018. Non-linear motor control by local learning in spiking neural networks. In *International Conference on Machine Learning*, 1773–1782. PMLR.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572.

Hertneck, M.; Köhler, J.; Trimpe, S.; and Allgöwer, F. 2018. Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3): 543–548.

Jakšić, S.; Bartocci, E.; Grosu, R.; Nguyen, T.; and Ničković, D. 2018. Quantitative monitoring of STL with edit distance. *Formal methods in system design*, 53(1): 83–112.

Ji, J.; Qiu, T.; Chen, B.; Zhang, B.; Lou, H.; Wang, K.; Duan, Y.; He, Z.; Zhou, J.; Zhang, Z.; et al. 2023. Ai alignment: A comprehensive survey. *arXiv preprint arXiv:2310.19852*.

Jin, X.; Deshmukh, J. V.; Kapinski, J.; Ueda, K.; and Butts, K. 2014. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, 253–262.

Johnson, M. A.; and Moradi, M. H. 2005. *PID control*. Springer.

Liu, W.; Mehdipour, N.; and Belta, C. 2021. Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints. *IEEE Control Systems Letters*.

Lv, C.; Xing, Y.; Zhang, J.; Na, X.; Li, Y.; Liu, T.; Cao, D.; and Wang, F.-Y. 2018. Levenberg–Marquardt Backpropagation Training of Multilayer Neural Networks for State Estimation of a Safety-Critical Cyber-Physical System. *IEEE Transactions on Industrial Informatics*, 14(8): 3436–3446.

Ma, S.; Liu, Y.; Lee, W.-C.; Zhang, X.; and Grama, A. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 175–186.

Majd, K.; Clark, G. M.; Khandait, T.; Zhou, S.; Sankaranarayanan, S.; Fainekos, G.; and Amor, H. 2022. Safe Robot Learning in Assistive Devices through Neural Network Repair. In *6th Annual Conference on Robot Learning*.

Maler, O.; and Nickovic, D. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, 152–166. Springer.

Mathworks. 2023. Adaptive Cruise Control System Using Model Predictive Control.

Nivison, S. A.; and Khargonekar, P. P. 2018. Development of a robust, sparsely-activated, and deep recurrent neural network controller for flight applications. In *2018 IEEE Conference on Decision and Control (CDC)*, 384–390. IEEE.

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

Shafahi, A.; Najibi, M.; Ghiasi, A.; Xu, Z.; Dickerson, J.; Studer, C.; Davis, L. S.; Taylor, G.; and Goldstein, T. 2019. Adversarial training for free! In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 3358–3369.

Sohn, J.; Kang, S.; and Yoo, S. 2023. Arachne: Search-based repair of deep neural networks. *ACM Transactions on Software Engineering and Methodology*, 32(4): 1–26.

Song, J.; Lyu, D.; Zhang, Z.; Wang, Z.; Zhang, T.; and Ma, L. 2022. When cyber-physical systems meet AI: a benchmark, an evaluation, and a way forward. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 343–352.

Tran, H.-D.; Yang, X.; Manzanas Lopez, D.; Musau, P.; Nguyen, L. V.; Xiang, W.; Bak, S.; and Johnson, T. T. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, 3–17. Springer.

Wang, H.; Ustun, B.; and Calmon, F. 2019. Repairing without retraining: Avoiding disparate impact with counterfactual distributions. In *International Conference on Machine Learning*, 6618–6627. PMLR.

website of this paper, S. 2021. "AutoRepair: Automated Repair for AI-Enabled Cyber-Physical Systems under Safety-Critical Conditions". https://sites.google.com/view/autorepair4aicps. Accessed: 2022-08-15.

Wilcoxon, F.; Katti, S.; Wilcox, R. A.; et al. 1970. Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1: 171–259.

Wohlin, C.; Runeson, P.; Hst, M.; Ohlsson, M. C.; Regnell, B.; and Wessln, A. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated. ISBN 3642290434, 9783642290435.

Wong, E.; and Kolter, Z. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, 5286–5295. PMLR.

Yaghoubi, S.; Fainekos, G.; and Sankaranarayanan, S. 2020. Training neural network controllers using control barrier functions in the presence of disturbances. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 1–6. IEEE.

Yang, X.; Yamaguchi, T.; Tran, H.-D.; Hoxha, B.; Johnson, T. T.; and Prokhorov, D. 2022. Neural network repair with reachability analysis. In *Formal Modeling and Analysis of Timed Systems: 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13–15, 2022, Proceedings*, 221–236. Springer.

Yu, B.; Qi, H.; Guo, Q.; Juefei-Xu, F.; Xie, X.; Ma, L.; and Zhao, J. 2021. DeepRepair: Style-Guided Repairing for Deep Neural Networks in the Real-World Operational Environment. *IEEE Transactions on Reliability*, 1–16.

Zhang, H.; and Chan, W. 2019. Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 376–387.

Zhang, Z.; An, J.; Arcaini, P.; and Hasuo, I. 2023. Online Causation Monitoring of Signal Temporal Logic. In Enea, C.; and Lal, A., eds., *Computer Aided Verification*, 62–84. Cham: Springer Nature Switzerland. ISBN 978-3-031-37706-8.

Zhang, Z.; Arcaini, P.; and Xie, X. 2022. Online Reset for Signal Temporal Logic Monitoring. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11): 4421–4432.

Zhang, Z.; Lyu, D.; Arcaini, P.; Ma, L.; Hasuo, I.; and Zhao, J. 2022. FalsifAI: Falsification of AI-Enabled Hybrid Control Systems Guided by Time-Aware Coverage Criteria. *IEEE Transactions on Software Engineering*.

Zhou, W.; Gao, R.; Kim, B.; Kang, E.; and Li, W. 2020. Runtime-safety-guided policy repair. In *International Conference on Runtime Verification*, 131–150. Springer.

Zhu, H.; Xiong, Z.; Magill, S.; and Jagannathan, S. 2019. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 686–701.

8 Reproducibility Checklist

Unless specified otherwise, please answer "yes" to each question if the relevant information is described either in the paper itself or in a technical appendix with an explicit reference from the main paper. If you wish to explain an answer further, please do so in a section titled "Reproducibility Checklist" at the end of the technical appendix.

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced. (yes/partial/no/NA) [yes]
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results. (yes/no) [yes]
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper. (yes/no) [yes]

1. Does this paper make theoretical contributions? (yes/no) [yes]

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no) [yes]
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no) [yes]
- Proofs of all novel claims are included. (yes/partial/no) [yes]
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no) [yes]
- Appropriate citations to theoretical tools used are given. (yes/partial/no) [yes]
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA) [yes]
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA) [yes]

2. **Does this paper rely on one or more datasets?** (yes/no) [yes]

If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets. (yes/partial/no/NA) [yes]
- All novel datasets introduced in this paper are included in a data appendix. (yes/partial/no/NA) [yes]
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no/NA) [yes]
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes/no/NA) [yes]
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes/partial/no/NA) [yes]
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing. (yes/partial/no/NA) [NA]

3. Does this paper include computational experiments? (yes/no) [yes]

- Any code required for pre-processing data is included in the appendix. (yes/partial/no) [yes]
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes/partial/no) [yes]
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no) [yes]
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from. (yes/partial/no) [yes]
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (yes/partial/no/NA) [yes]
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes/partial/no) [yes]
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes/partial/no) [yes]
- This paper states the number of algorithm runs used to compute each reported result. (yes/no) [yes]
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes/no) [yes]
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes/partial/no) [yes]
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (yes/partial/no/NA) [yes]
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (yes/partial/no/NA) [yes]