AUTOREPAIR: Automated Repair for AI-Enabled Cyber-Physical Systems under Safety-Critical Conditions

Deyun Lyu, Jiayang Song, Zhenya Zhang, Zhijie Wang, Tianyi Zhang, Lei Ma and Jianjun Zhao

Abstract—Cyber-Physical Systems (CPS) have been widely deployed in safety-critical domains such as transportation, power and energy. Recently, there comes an increasing demand in employing deep neural networks (DNNs) in CPS for more intelligent control and decision making in sophisticated industrial safetycritical conditions, giving birth to the class of DNN controllers. However, due to the inherent uncertainty and opaqueness of DNNs, concerns about the safety of DNN-enabled CPS are also surging. In this work, we propose an automated framework named AUTOREPAIR that, given a safety requirement, identifies unsafe control behavior in a DNN controller and repairs them through an optimization-based method. Having an unsafe signal of system execution, AUTOREPAIR iteratively explores the control decision space and searches for the optimal corrections for the DNN controller in order to satisfy the safety requirements. We conduct a comprehensive evaluation of AUTOREPAIR on 6 instances of industry-level DNN-enabled CPS from different safetycritical domains. Evaluation results show that AUTOREPAIR successfully repairs critical safety issues in the DNN controllers, and significantly improves the reliability of CPS.

Index Terms—Cyber-physical systems, Neural network controllers, System repair.

I. Introduction

Cyber-Physical Systems (CPS) employ computer technologies to monitor and control physical components for various real-world tasks. Examples of CPS include adaptive cruise control systems, wind turbine systems, smart grid systems, etc. Typically, a CPS consists of a physical plant and a digital controller—the 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.

Traditional controllers, such as those based on *proportional* integral derivative (PID) [1] and model predictive control (MPC) [2], often adopt human-crafted heuristics or linear models to make control decisions. Developing these controllers is costly, which requires huge investment and numerous iterations to ensure their robustness and reliability to handle sophisticated conditions in the real world. Recent studies [3]–[5] have shown that DNN controllers can outperform traditional controllers in terms of robustness and runtime performance

Deyun Lyu, Zhenya Zhang and Jianjun Zhao are with the Faculty and Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan (e-mail: zhang@ait.kyushu-u.ac.jp).

Jiayang Song, Zhijie Wang and Lei Ma are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada. Tianyi Zhang is with the Department of Computer Science, Purdue University, West Lafayette, USA.

in several CPS domains, demonstrating the promise of using DNN controllers as an alternative for traditional controllers.

Despite an increasing trend of adopting DNN controllers in CPS, a surge of concerns also arise due to the inherent uncertainty and opacity nature of DNNs. As CPS are often deployed in safety-critical domains, improper control decisions may lead to catastrophic system failures, which may further bring intolerable losses and severe social impacts. Most of the existing research efforts have been paid to the testing of CPS (e.g., optimization-based falsification [6]–[13]), which aims to detect those dangerous cases. However, to enhance the reliability of DNN-enabled CPS and reduce the risks caused by improper control decisions, it is necessary to diagnose and repair the unsafe system behavior after finding them out, which has not received much attention.

In this paper, we make an early attempt along this direction and propose a framework, AUTOREPAIR, for automated repair of DNN-enabled CPS under safety-critical conditions. Given a set of safety requirements, AUTOREPAIR first identifies unsafe system behavior by checking the safety satisfaction of system outputs. Then, AUTOREPAIR repairs the control decisions through an optimization-based method. Given an unsafe system execution, AUTOREPAIR iteratively explores the control decision space and searches for optimal control corrections that enable the system to satisfy the safety requirements. Finally, AUTOREPAIR includes those repaired control inputs and outputs into a training dataset and performs robust retraining of the DNN controller. As a result, an enhanced DNN controller is obtained which corrects the unsafe system behavior and improves the reliability of the system under safety-critical conditions.

In summary, we make the following contributions:

- First, we propose a novel technique for automated repair
 of DNN-enabled CPS under safety-critical conditions. We
 formulate the problem into a multi-objective optimization,
 and employ optimization solvers to search for the optimal
 corrections for the control signals, which ensure that the system execution after repair satisfies the safety requirement;
- Then, we implement our proposed technique as a general framework called AUTOREPAIR that can handle industrial CPS in *Simulink*, which is the *de facto* standard for CPS modeling environment used in industry;
- We perform a comprehensive experimental evaluation on 6 instances of industry-level DNN-enabled CPS. The evaluation results demonstrate the effectiveness of our approach.

To the best of our knowledge, this is the first work that aims

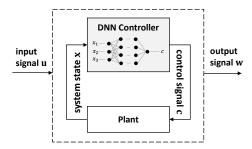


Fig. 1. The architecture of a typical DNN-enabled CPS

to repair DNN controllers with respect to system-level safety requirements. Together with the trending demand of adopting DNN in industrial CPS, this work paves the path to further research and real-world safe adoption of DNN in complex industrial CPS.

II. PRELIMINARIES

In this section, we give an overview of DNN-enabled CPS (Section II-A), DNN controllers (Section II-B), and the system requirements expressed in *Signal Temporal Logic (STL)* (Section II-C) as the preliminaries of our proposed techniques.

A. DNN-Enabled CPS

Fig. 1 shows the typical overall system architecture of DNN-enabled CPS. In particular, the system consists of a *plant* and a *DNN controller*. The plant is a physical component with continuous state transitions, which are highly non-linear and complex. The DNN controller takes a system state \mathbf{x} and an external input signal \mathbf{u} as input. Then, it issues a control signal \mathbf{c} at each timestamp to control the execution of the plant. This in turn produces an output signal \mathbf{w} that is observable from the outside. To model the dynamics of signals and system states (i.e., $\mathbf{u}, \mathbf{w}, \mathbf{x}, \mathbf{c}$), we define them as *time series*, which maps a time instant $t \in [0, T]$ ($T \in \mathbb{R}_{\geq 0}$) to a vector $\mathbf{s}(t) \in \mathbb{R}^d$; Here, d is the dimension of the signal or state, and [0, T] is the *time range* of the signal or state.

Specifically, \mathbf{c} is one-dimensional, and is bounded by a closed interval $\mathcal{B}_{\mathbf{c}}$ of real numbers, i.e., $\forall t \in [0, T] \cdot \mathbf{c}(t) \in \mathcal{B}_{\mathbf{c}}$.

B. DNN Controllers and Their Training

Traditional CPS controllers, such as those based on *PID* [1] and *MPC* [2], employ human-crafted heuristics or linear models to make control decisions. Recently, DNN has been increasingly investigated in industrial contexts as an alternative to traditional control algorithms, in handling sophisticated conditions in the real world. Currently, two approaches are usually adopted to train DNN controllers, namely, *Deep Reinforcement Learning (DRL)* and *Supervised Learning*:

- 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 a DNN, by iteratively performing "trial and error".
- In supervised learning, the trainee learns from the historical or experience data collected in the CPS as a training dataset

 $\{\langle \mathbf{x}, \mathbf{c} \rangle\}$, in which each element is a pair of a system state \mathbf{x} and the corresponding control decision \mathbf{c} . The training phase is to optimize the DNN parameters that maximize the accuracy of predicting \mathbf{c} given \mathbf{x} .

In this paper, we consider the DNN controller repair in the CPS development and maintenance contexts under the white-box settings, 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 a DNN 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 DNN controller, for continuous delivery. Moreover, we assume that $\mathbf{u}, \mathbf{w}, \mathbf{x}, \mathbf{c}$ (i.e., the signals between system components and between the system and external environment) are all observable. Furthermore, we assume that the control decisions made by the DNN controller account for the unsafe behavior of CPS.

C. Safety Requirement of CPS

In this work, we adopt Signal Temporal Logic (STL) [14] to specify the safety requirements of the DNN-enabled CPS. STL has been a widely-accepted formalism [15], [16] for expressing the users' desired properties. It extends the classic linear temporal logic (LTL) [17] 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 quantitative STL semantics has become the technical fundamental of many other safety assurance techniques, such as runtime monitoring [18]–[20] and falsification [6]–[11]. Below we introduce the syntax and robustness of STL.

STL syntax In STL, the atomic proposition α is defined as $\alpha \equiv (f(\mathbf{w}) > 0)$, where f is a function that maps an output signal to a real number. An STL formula φ is defined recursively as follows:

$$\varphi :\equiv \alpha \mid \bot \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where I represents a time interval. Here, $\Box_I \varphi$, $\Diamond_I \varphi$ and $\varphi_1 \mathcal{U}_I \varphi_2$ are the temporal propositions, where $\Box_I \varphi$ requires that *something always happens*, $\Diamond_I \varphi$ requires that *something eventually happens*, and $\varphi_1 \mathcal{U}_I \varphi_2$ requires that φ_1 keeps happening until φ_2 . Other common operators, such as \vee , can be derived by the existing operators: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg \varphi_1 \wedge \neg \varphi_2)$.

STL robustness Given the system output \mathbf{w} and an STL formula φ , the robustness $\mathsf{rob}(\mathbf{w}, \varphi)$ returns a real number that indicates *how robustly* \mathbf{w} satisfies φ . The definition of $\mathsf{rob}(\mathbf{w}, \varphi)$ is described as follows, by induction on the structure of the STL formula:

$$\begin{split} \operatorname{rob}\left(\mathbf{w},\alpha\right) &:= f(\mathbf{w}) & \operatorname{rob}\left(\mathbf{w},\bot\right) := -\infty \\ \operatorname{rob}\left(\mathbf{w},\neg\varphi\right) &:= -\operatorname{rob}\left(\mathbf{w},\varphi\right) \\ \operatorname{rob}\left(\mathbf{w},\varphi_1 \land \varphi_2\right) &:= \min\left(\operatorname{rob}\left(\mathbf{w},\varphi_1\right),\operatorname{rob}\left(\mathbf{w},\varphi_2\right)\right) \\ \operatorname{rob}\left(\mathbf{w},\Box_I\varphi\right) &:= \inf_{t \in I}(\operatorname{rob}\left(\mathbf{w}^t,\varphi\right)\right) \\ \operatorname{rob}\left(\mathbf{w},\varphi_1 \ \mathcal{U}_I \ \varphi_2\right) &:= \sup_{t \in I} \min\left(\begin{array}{c} \operatorname{rob}\left(\mathbf{w},\varphi_2\right), \\ \inf_{t' \in [0,t)} \operatorname{rob}\left(\mathbf{w}^{t'},\varphi_1\right) \end{array}\right) \end{split}$$

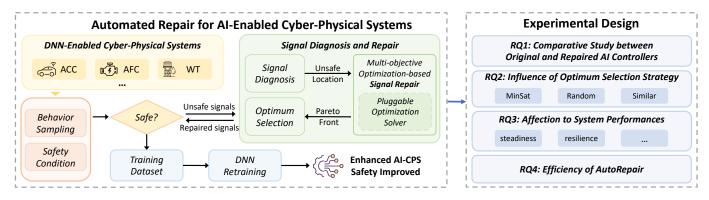


Fig. 2. The workflow and study design of our proposed framework AUTOREPAIR for automated repair of DNN-enabled CPS.

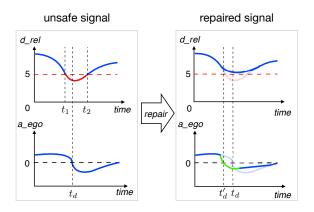


Fig. 3. An illustrative example of signal repair for an adaptive cruise control (ACC) system.

Here \mathbf{w}^t denotes a signal obtained by *shifting* the starting point of \mathbf{w} for t time units. Intuitively, the larger $\mathsf{rob}(\mathbf{w}, \varphi)$ is, the more robustly \mathbf{w} satisfies φ ; once $\mathsf{rob}(\mathbf{w}, \varphi)$ is negative, then it indicates that \mathbf{w} violates φ .

III. THE AUTOREPAIR FRAMEWORK

In this paper, we propose a framework, named AUTORE-PAIR, to enhance DNN controllers for system safety. Specifically, AUTOREPAIR (i) identifies unsafe output signals in a system execution, (ii) repairs them by searching for a control signal patch to ensure that the system satisfies safety requirements during the given execution, and (iii) repairs the DNN controller with the safe execution trace data. We first illustrate how to repair an unsafe system execution in Example III.1, and then introduce the workflow of AUTOREPAIR.

Example III.1. Consider the *adaptive cruise control (ACC)* scenario, in which an ego car tries to follow a lead car within a safe distance. We treat the ego car as the DNN-enabled system under consideration, and the lead car as the external environment. The controller of the ego car decides its acceleration or deceleration a_{ego} based on the sensor data, including the states of the ego car and the lead car. The system requirement φ in this scenario can be expressed as follows,

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

which requires that, the relative distance d_{rel} between the two cars should always be larger than a safe distance of 5.

The left part of Fig. 3 shows an example of system execution that violates the safety requirement. During $[t_1,t_2]$, d_{rel} is smaller than 5. This is because, t_d (i.e., the time when the ego car starts to decelerate) is too late to avoid a close relative distance. The right part of Fig. 3 shows an example of signal repair, in which a *control signal patch* (the green segment) replaces the original control signal. The signal patch brings the deceleration moment t_d forward to t_d' . As a result, the ego car decelerates timely and avoids violating the safety requirement. Now, with the control signal patch, we can rerun the system to obtain a new execution trace that avoids the unsafe behavior. This new execution trace will be used to repair the DNN controller to enhance the system reliability.

This example explains the main task of AUTOREPAIR, namely, for an unsafe system execution, it requires to synthesize a control signal patch, which enables the system execution to satisfy the safety requirement. Before the technical details, we first overview the high-level workflow of AUTOREPAIR.

Workflow. Given a CPS with a DNN controller, AUTOREPAIR 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 \mathbf{c} , and a system state signal \mathbf{x} . Then, given a safety requirement φ , AUTOREPAIR checks whether each system output signal \mathbf{w} in the 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 IV.

With the safety-assured execution trace, we can repair the original DNN controller that issued improper control signals and led to unsafe system behavior. To be general, as an early step for DNN controller repair, the DNN repair module in AUTOREPAIR is designed to be extensible (to diverse repairing methods). The basic mode is to retrain the original DNN controller with the new safety-assured execution trace to enhance its performance, while keep the remaining training configurations unchanged. Although this mode has the lowest cost, it sometimes produces a less robust DNN controller since the model architecture may not be optimal. In addition to the basic mode, other options include *Neural Architecture Search* (*NAS*) [21], which can search over different DNN configura-

Algorithm 1 The optimization-based signal repair

Require: The original system with a DNN controller C, a safety requirement φ , and an execution trace including input signal \mathbf{u} , output signal \mathbf{w} , control signal \mathbf{c} , and state signal \mathbf{x}

1: **procedure** DIAGANDREPAIR(
$$\mathbf{u}, \mathbf{w}, \mathbf{c}, \mathbf{x}$$
)
2: **if** $\mathbf{rob}(\mathbf{w}, \varphi) < 0$ **then** \triangleright **w** is unsafe

3:
$$\begin{bmatrix} \tau_{\mathbf{S}} \leftarrow \text{ when the violation episode starts} \\ \tau_{\mathbf{E}} \leftarrow \text{ when the violation episode ends} \end{bmatrix} \quad \triangleright \text{ diagnosis}$$

4:
$$\mathbf{c}' \leftarrow \text{arg} \begin{bmatrix} \max_{\mathbf{c}'} \mathbf{rob}\left(\mathbf{w}_{[0,\tau_{\mathbf{E}}]}^*, \varphi\right) \\ \min_{\mathbf{c}'} \mathbf{dist}\left(\mathbf{c}', \mathbf{c}_{[t_{\mathbf{c}'}^L, t_{\mathbf{c}'}^U]}\right) \\ \text{s.t.} \quad t_{\mathbf{c}'}^L \in [0, \tau_{\mathbf{S}}] \\ t_{\mathbf{c}'}^U \in (t_{\mathbf{c}'}^L, \tau_{\mathbf{E}}] \\ \forall t \in [t_{\mathbf{c}'}^L, t_{\mathbf{c}'}^U] \cdot \mathbf{c}'(t) \in \mathcal{B}_{\mathbf{c}} \end{bmatrix} \quad \triangleright \text{ repair}$$

5:
$$\mathbf{w}, \mathbf{c}, \mathbf{x} \leftarrow \text{SYSEXE}(\mathbf{u}, \mathbf{c}^*) \text{ with } \mathbf{c}^* = \langle \mathcal{C}, \mathbf{c}' \rangle \quad \triangleright \text{ system rerun}$$

6: DIAGANDREPAIR($\mathbf{u}, \mathbf{w}, \mathbf{c}, \mathbf{x}$) $\triangleright \text{ recursive call}$

7: add the repaired $\langle \mathbf{x}, \mathbf{c} \rangle$ to the retraining dataset

tions and identify the best model with the new execution trace. In this work, we experimentally evaluate the performance of the repaired DNN controller obtained by retraining, and we leave the investigation on other options as future work.

As signal diagnosis and repair are the major technical contributions of the AUTOREPAIR framework, we elaborate on their details in the next section.

IV. SIGNAL DIAGNOSIS AND REPAIR

In this section, we introduce the details of the key component of AUTOREPAIR, i.e., signal diagnosis and repair, which aims to repair the control signals that lead to unsafe system output in a system execution.

A. Algorithm Overview

Alg. 1 describes the signal diagnosis and repair algorithm. It takes as input a CPS with a DNN controller C, an STL safety requirement φ , and an execution trace including input signal u, output signal w, control signal c, and state signal x. Note that all these signals are defined as time series to model their dynamics over a time range [0,T], as defined in Section II-A. This procedure starts with checking whether the output signal w satisfies the safety requirement φ (Line 2). If w is safe, then there is no need to repair it. Otherwise, the algorithm first localizes the starting and ending time instants of the unsafe output signal (Line 3). Then, it searches for an optimal control signal patch \mathbf{c}' that corrects the system output signal to satisfy the safety requirement (Line 4). Formally, c' is a partial signal defined in the time interval $[t_{\mathbf{c}'}^{\mathsf{L}}, t_{\mathbf{c}'}^{\mathsf{U}}]$, where $t_{\mathbf{c}'}^{\mathsf{L}}$ is the lower time bound at which the control must start being rectified to avoid the unsafe system behavior, and $t_{\mathbf{c}'}^{\mathtt{U}}$ is the upper time bound at which the control rectification can end. The details of deciding $t_{\mathbf{c}'}^{\mathbf{L}}$ and $t_{\mathbf{c}'}^{\mathbf{U}}$ will be given in Section IV-B.

With the control signal patch \mathbf{c}' , we employ an alternate control between the original controller \mathcal{C} and the patch \mathbf{c}' to re-execute the system, described as below:

- in $[0, t_{\mathbf{c}'}^{\mathsf{L}}]$, the control is given by the original controller;
- in $[t_{\mathbf{c}'}^{\mathsf{L}}, t_{\mathbf{c}'}^{\mathsf{U}}]$, the control is switched to \mathbf{c}' ;

• in $[t_{\mathbf{c}'}^{\mathsf{U}}, T]$, the control is switched back to the original controller, until the end of the system execution.

Using this control switch, we can obtain new \mathbf{w} , \mathbf{c} and \mathbf{x} by re-executing the system (Line 5). This procedure is invoked recursively with the new \mathbf{w} , \mathbf{c} and \mathbf{x} to diagnose and repair new unsafe behavior, until the entire system execution is safe. Finally, this new execution is returned to the AUTOREPAIR framework and fed as input into the DNN repair module to enhance the DNN controller (Line 7).

B. Signal Diagnosis

A naive selection of the time range $[t_{\mathbf{c}'}^{\mathbf{L}}, t_{\mathbf{c}'}^{\mathbf{U}}]$, in which we search for the control signal patch \mathbf{c}' to repair the unsafe system execution, is the entire time range [0, T]. However, the search in that case will be too expensive, due to the large search space. In this section, we leverage the recent progress of STL signal diagnosis [22], [23] to reduce the search space.

Signal diagnosis is a technique that can localize the *violation episode* in a signal. Intuitively, a violation episode is a segment of a signal that could be considered as the *cause* of the violation of the whole signal to a given safety property. In general, signal diagnosis is achieved by recursively collecting the time instants at which the evaluations of the atomic proposition of the safety property lead to the violation of the given STL formula, based on the semantics of different STL operators. We illustrate this process using an example in Example IV.1. For more details, we refer the readers to [22].

Example IV.1. In Example III.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 3 of Alg. 1, $\tau_{\rm S}$ is set as the timestamp when the violation episode starts, i.e., when the system output starts to violate the safety requirement; and $\tau_{\rm E}$ 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 IV.2 for an explanation.

Lemma IV.2. Let \mathbf{c}' be a control signal patch defined in the interval $[t_{\mathbf{c}'}^{\mathsf{L}}, t_{\mathbf{c}'}^{\mathsf{U}}]$, and τ_{S} and τ_{E} be respectively the starting and the ending point of the violation episode. Then, the lower time bound $t_{\mathbf{c}'}^{\mathsf{L}}$ of \mathbf{c}' should be not later than τ_{S} ; and the upper time bound $t_{\mathbf{c}'}^{\mathsf{U}}$ of \mathbf{c}' should be not later than τ_{E} .

The proof of Lemma IV.2 is based on the definition of STL signal diagnosis, and the fact that safety violations only propagate backward in the time range.

If $t_{\mathbf{c}'}^{\mathrm{L}} > \tau_{\mathrm{S}}$, there will be an unsafe segment in $[\tau_{\mathrm{S}}, t_{\mathbf{c}'}^{\mathrm{L}}]$ that is not repaired by the control signal patch \mathbf{c}' . Therefore, the lower time bound $t_{\mathbf{c}'}^{\mathrm{L}}$ of \mathbf{c}' should not be later than τ_{S} .

Similarly, we can derive that the upper time bound $t_{\mathbf{c}'}^{\mathbf{U}}$ of \mathbf{c}' should not be later than $\tau_{\mathbf{E}}$.

Example IV.3. In Example III.1, $\tau_{\rm S}$ and $\tau_{\rm E}$ are respectively computed as $\tau_{\rm S}=t_1$ and $\tau_{\rm E}=t_2$. Therefore, when we search for the control signal patch ${\bf c}'$, its lower time bound $t_{{\bf c}'}^{\rm L}$ should be not later than t_1 , and its upper time bound $t_{{\bf c}'}^{\rm U}$ should be not later than t_2 .

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

C. Multi-Objective Optimization-Based Repair

There remains the problem of how to construct the control signal patch \mathbf{c}' , given the time constraint $[t_{\mathbf{c}'}^{\mathrm{L}}, t_{\mathbf{c}'}^{\mathrm{U}}]$. The desired properties of \mathbf{c}' involve the following two aspects:

- First, by substituting the original control signal with \mathbf{c}' , the safety violation in $[\tau_S, \tau_E]$ should be resolved;
- Second, the change between the original control signal c and the control signal patch c' should be minimal, in order to minimize the performance losses in other aspects than safety and 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 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 c' as multi-objective optimization, as in Line 4 of Alg. 1:

- The first objective function is to maximize the satisfaction of the repaired output signal $\mathbf{w}_{[0,\tau_{\rm E}]}^*$ in $[0,\tau_{\rm E}]$ to the safety requirement. Initially, $\mathbf{w}_{[0,\tau_{\rm E}]}^* = \mathbf{w}_{[0,\tau_{\rm E}]}$ and so ${\bf rob}\left(\mathbf{w}_{[0,\tau_{\rm E}]}^*,\varphi\right)$ is negative. The goal of this objective function is to find a ${\bf c}'$ such that the satisfaction ${\bf rob}\left(\mathbf{w}_{[0,\tau_{\rm E}]}^*,\varphi\right)$ of the repaired signal $\mathbf{w}_{[0,\tau_{\rm E}]}^*$ is positive, and so $\mathbf{w}_{[0,\tau_{\rm E}]}^*$ satisfies the safety requirement φ .
- The second objective function is to minimize the change of \mathbf{c}' compared to the original control signal in $[t_{\mathbf{c}'}^{\mathrm{L}}, t_{\mathbf{c}'}^{\mathrm{U}}]$, in terms of a specific distance measure.
- The constraints include the time constraint in Section IV-B and the space constraint in Section II-A.

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 [15], [16], [24], 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 interval $\left[t_{\mathbf{c}'}^{\mathbf{L}} + \frac{i(t_{\mathbf{c}'}^{\mathbf{U}} - t_{\mathbf{c}'}^{\mathbf{L}})}{k}, t_{\mathbf{c}'}^{\mathbf{L}} + \frac{(i+1)(t_{\mathbf{c}'}^{\mathbf{U}} - t_{\mathbf{c}'}^{\mathbf{L}})}{k}\right]$ $(i \in \{1, \ldots, k-1\})$. In that case, \mathbf{c}' is identified by $t_{\mathbf{c}'}^{\mathbf{L}}$, $t_{\mathbf{c}'}^{\mathbf{U}}$, and the k variables that characterize the shape of \mathbf{c}' .

An efficient optimization solver is needed to solve the optimization problem formulated in Line 4. In our context, the evaluation of the objective $\operatorname{rob}\left(\mathbf{w}_{[0,\tau_E]}^*,\varphi\right)$ requires system execution of CPS, whose dynamics is often hard to model. To that end, we select NSGA-II [25], 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 AUTOREPAIR, we make this selection process configurable, and here we provide three possible strategies, as follows. In RQ2 of Section VI, we experimentally compare their performances.

- MinSat selects the optimum which triggers a minimum satisfaction to the safety requirement, to accommodate the other objective function;
- Random refers to a naïve selection strategy that randomly selects a solution from the Pareto front, as long as it satisfies the safety requirement;
- Similar selects the solution with the minimal change w.r.t. the original control decisions, under the premise of satisfying the safety requirement.

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

V. EXPERIMENT DESIGN AND SETUP

In this section, we introduce the design and setup of the experiments used to evaluate the usefulness and performance of AUTOREPAIR. All the source code and benchmarks are publicly available: https://github.com/lyudeyun/AutoRepair.

A. Research Questions

We design our experiments in order to investigate the following research questions:

RQ1: Does AUTOREPAIR effectively improve the safety of DNN-enabled CPS?

In this RQ, we aim to evaluate whether the subject CPS with the DNN controller repaired by AUTOREPAIR can indeed improve the safety of the system. To answer this question, we first create the retraining dataset by sampling the execution traces of these systems, and apply AUTOREPAIR to obtain the repaired systems with retrained DNN controllers. For the testing purpose, we also create a test set for each system. Then, we execute the original systems and the repaired systems, by feeding them the input signals from the retraining dataset and the test set. We compare the safety rates 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.

TABLE I
SUBJECT CPS ADOPTED TO EVALUATE AUTOREPAIR. THE COLUMN **BLOCKS** REPORTS THE NUMBER OF THE COMPONENT BLOCKS IN A SYSTEM, IN
ORDER TO MEASURE THE COMPLEXITY OF THE MODEL.

Subject CPS	Description	Input signals	Control signals	Blocks
Adaptive Cruise Control	Maintain a safe distance from a lead car	Lead car acceleration	Ego car acceleration	297
Abstract Fuel Control	Maintain the reference air-to-fuel ratio	Pedal angle & engine speed	Fuel command	281
Water Tank	Keep the water level at a reference value	Reference water level	Water flowrate	919

• RQ2: How does the optimum selection strategy influence the effectiveness of AUTOREPAIR?

A key phase in AUTOREPAIR is the signal diagnosis and repair procedure introduced in Section IV. Given an unsafe execution trace, we search for an optimal control signal patch which ensures the system safety. However, since the output given by the optimization solver is a Pareto front, we need to select a specific optimum and apply it as the control signal patch to repair the system. In Section IV-C, we have introduced three optimum selection strategies, namely, MinSat, Random, Similar. In this RQ, we compare the safety rates of the systems under different optimum selection strategies to study the impact of these strategies.

• RQ3: Does AUTOREPAIR affect the performance of DNN-enabled CPS in terms of other metrics than safety? Although AUTOREPAIR aims to improve the system safety, a concern arises that it may diminish the system performances in other aspects, such as the efficiency of the system execution. In this RQ, we investigate the influence of AUTOREPAIR on the performance of the DNN-enabled CPS w.r.t. various system performance metrics. Specifically, we compare the performance of the repaired systems, by AUTOREPAIR with the default optimum selection strategy MinSat, 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 Section V-B.

• RQ4: How is the efficiency of AUTOREPAIR?

In addition to the effectiveness of AUTOREPAIR, the efficiency of AUTOREPAIR is another important factor for its real-world application. The most time consuming part in AUTOREPAIR is the procedure of signal repair for each unsafe system execution. Since this procedure relies on the multi-objective optimization algorithm NSGA-II, the time cost is dependent on the configuration of NSGA-II. Moreover, different systems have different execution time, and this also affects the time cost of applying AUTOREPAIR. In this RQ, we investigate this aspect of AUTOREPAIR, and report how efficient AUTOREPAIR is for repairing the unsafe behavior of DNN-enabled CPS.

B. Benchmarks

We selected DNN-enabled CPS from three industrial domains as the subject systems for our evaluation, namely, *Adaptive Cruise Control* (ACC) [26], *Abstract Fuel Control* (AFC) [27], and *Water Tank* (WT) [28]. These systems span

over different safety-critical industrial domains, such as autonomous driving and powertrain control. Table I summarizes these systems with a short description of their functionalities. All these systems are built in Simulink, the *de facto* CPS modeling standard in industry. The rest of this section contains the detailed information of each subject CPS, which involves their functionality, safety requirements and other performance metrics.

Adaptive Cruise Control (ACC). As an important driving assistance function, ACC has been popularized in the automotive field after decades of development. The ACC system mentioned in this work is originally from MathWorks [26], and it aims to maintain the safety distance of an ego car from a lead car by adjusting the acceleration of ego car. When the relative distance is larger than the safe distance, the speed of ego car will instead maintain at the driver-set velocity. To this end, the whole system takes the acceleration of the lead car as input and outputs the velocity and the moving distance of the lead car and the ego car. The safety requirement of ACC is formulated as follows, saying that during the simulation, the relative distance, d_{rel} , between the two cars should always be larger than a safety distance, defined by the sum of a constant d_{safe} and the braking distance of the ego car. Meanwhile, the speed of the ego car v_{eqo} should be lower than v_{set} . Here, we set d_{safe} and v_{set} as 10 and 30, respectively.

$$S_{\text{ACC}} \equiv \Box_{[0.50]} (d_{rel} \geq d_{safe} \wedge v_{eqo} \leq v_{set})$$

The performance metrics of ACC are described as follows.

- space refers to the longitudinal distance traveled by the ego car. The greater the space is, the higher the driving efficiency is;
- speed denotes the average deviation between the actual speed of the ego car and the expected speed when the car travels in a safe status. A smaller speed means higher cruise efficiency;
- *steadiness* aims to measure the system stability, which is reflected by the proportion of the time period during which the system satisfies the safety requirement over the total time interval [0, 50];
- resilience aims to assess whether a controller acts rapidly to return to a steady state, when the system is not steady;
- comfort indicates how comfortable the ego car is, by measuring the maximum changing rate of the acceleration during the system execution.

Abstract Fuel Control (AFC). AFC is a complex air-fuel control system released by Toyota [27]. The whole system takes two input signals from the outside environment, namely,

pedal angle and engine speed, and outputs $\mu = \frac{|AF - AF_{ref}|}{AF_{ref}}$, which is the deviation of the air-to-fuel ratio AF from a reference value AF_{ref} . By changing the system inputs, the fuel controller should adjust the intake gas rate to the cylinder to maintain the optimal air-to-fuel ratio. The goal of this system is to control the deviation μ such that it is within a predefined threshold $\mu_{set} = 0.15$ during the time interval [0,30].

$$S_{\mathrm{AFC}} \equiv \Box_{[0,30]} \left(\frac{|AF - AF_{ref}|}{AF_{ref}} \le \mu_{set} \right)$$

The performance metrics of AFC are described as follows.

- error indicates the absolute value of the actual deviation rate μ. A small error indicates better performance, and ideally, error should be 0;
- *steadiness* is characterized by the proportion of the time period during which the system satisfies a safety requirement over the total time interval [0, 30].

Water Tank (WT). As a container for controlling the inflow and outflow of water, WT has been applied in the domains such as chemical industry. This system is collected from MathWorks [28]. Water can inflow or outflow the water tank, until the height of water in the tank reaches the reference value. After a certain time, the height of water should be the same as the reference height. The system takes the reference water level h_{ref} as input signal and outputs the actual water level h_{out} in real time. The absolute deviation error between h_{ref} and h_{out} should always be less than a setting value $error_{set}$ within the time intervals [4, 5], [9, 10] and [14, 15]. Here $error_{set}$ is 0.86.

$$\begin{split} S_{\mathrm{WT}} & \equiv \Box_{\mathrm{I}}(|\mathit{error}| \leq \mathit{error}_\mathit{set}) \\ & \text{where } \mathrm{I} = [4, 5] \cup [9, 10] \cup [14, 15] \end{split}$$

The performance metrics of WT are formulated as follows,

- error indicates the absolute error value between the reference value and the real value. For WT, this is indicated by the absolute error value between h_{ref} and h_{out}. A smaller error represents a higher control accuracy;
- *steadiness* is characterized by the proportion of the time period during which the system satisfies a safety requirement, over the time intervals [4, 5], [9, 10] and [14, 15].

Neural network controllers. To demonstrate the effectiveness of AUTOREPAIR, for each subject CPS, we trained 2 neural network controllers with different structures, and thus, we have 6 instances in total as the systems under evaluation. In Table II, we make a detailed introduction to the original controllers deployed on these systems, including their types, structures and training algorithms. Taking ACC#2 as an example, the original DNN controller constructed by *Feed-Forward Neural Network (FFNN)* consists of 3 hidden layers, and each hidden layer has 30 neurons. This controller is trained by the built-in BFG algorithm of *Deep Learning Toolbox* in MATLAB.

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 controllers in the retraining dataset in Table II, which uses $\frac{\#safe\ trials}{\#total\ trials}$ to represent how safe a system is. As illustrated in Table II,

TABLE II
THE DETAILS OF SIX SUBJECT CPS INSTANCES

Instance	Original DNN Controller	SR
	Type: FFNN	
ACC#1	Structure: [10, 10, 10]	9682/10000
	Training Algorithm: LMBP	*
	Type: FFNN	
ACC#2	Structure: [30, 30, 30]	8200/10000
	Training Algorithm: BFG*	*
	Type: FFNN	
AFC#1	Structure: [15, 15, 15]	8287/10000
	Training Algorithm: LMBP	
	Type: FFNN	
AFC#2	Structure: [30, 30, 30]	8661/10000
	Training Algorithm: LMBP	
	Type: FFNN	
WT#1	Structure: [5, 5, 5]	8404/10000
	Training Algorithm: BFG	
	Type: FFNN	
WT#2	Structure: [15, 15, 15]	9808/10000
	Training Algorithm: BFG	
* LMBP:	Levenberg-Marquardt	Backpropagation

^{*} LMBP: Levenberg-Marquardt Backpropagation (LMBP) Algorithm [29]

TABLE III PARAMETERS OF NSGA-II ALGORITHM

Parameter of NSGA-II	Value
Population Size	40
Maximum Generation	10
Number of Objective	3
Number of Decision Variable	6

the safety rates of the 6 instances to be repaired range from 82.0% to 98.1%.

C. Software and Hardware Dependencies

All the subject CPS are implemented in Simulink with dependencies on MATLAB toolboxes *Model Predictive Control, Control System*, and *Deep Learning*. The experiments were conducted on AWS cloud instances of type c4.2xlarge (2.9 GHz Intel Xeon E5-2666 v3, 15G RAM). For DNN controller retraining, we employ the features of the *Deep Learning Toolbox* of MATLAB, and run it on a server with a 3.3GHz Intel i9-10940X CPU, 64G RAM, and two NVIDIA RTX A6000 GPUs. The evaluation took around 480 hours (20 days * 24 hours) on 20 AWS instances in total.

To conduct multi-objective optimization-based repair in Section IV-C, we select NSGA-II as our solver. The main parameters of NSGA-II employed in our evaluation are listed in Table III. Here we set the population size and the maximum generation of NSGA-II as 40 and 10, respectively. The number of objective functions is 3, with the second one (regarding the distance between the original control signal and the control patch signal) in Section IV-C instantiated to a spatial constraint and a temporal constraint. The number of decision variables is the dimension of a single optimal solution, namely 6, which

^{**} BFG: BFGS quasi-Newton (BFG) Algorithm [30]

TABLE IV
THE SAFETY RATE OF THE ORIGINAL SYSTEMS, AND THE SYSTEMS AFTER REPAIR UNDER THREE OPTIMUM SELECTION STRATEGIES. THE BEST PERFORMER FOR EACH SYSTEM INSTANCE 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										
Orig	ginal	96.8%	97.4%	82.0%	84.1%	82.9%	85.4%	86.6%	85.5%	84.0%	84.0%	98.1%	97.7%
Repaired	MinSat	75.6%	76.0%	95.4%	96.7%	82.3%	83.7%	91.5%	91.9%	86.1%	85.8%	98.2%	97.8%
	Similar	97.8%	97.9%	81.0%	81.9%	96.0%	96.3%	81.1%	81.5%	86.8%	85.9%	98.2%	97.8%
	Random	96.3%	97.1%	71.9%	72.3%	99.4%	99.6%	90.5%	90.3%	93.1%	92.5%	98.5%	98.1%

consists of 2 timestamps $t_{\mathbf{c}'}^{L}$ and $t_{\mathbf{c}'}^{U}$, and 4 interpolation points used to generate the control signal patch \mathbf{c}' .

VI. EVALUATION RESULTS

In this section, we present our evaluation results used to evaluate our proposed framework AUTOREPAIR.

RQ1: Does AUTOREPAIR effectively improve the safety of DNN-enabled CPS?

This RQ aims to evaluate whether AUTOREPAIR can truly improve the safety of these subject CPS. We run these subject CPS over the retraining dataset and the test set, with the original controller and with the repaired controller respectively, by feeding the input signals from the executions in the retraining dataset and the test set into the systems.

The experimental results for RQ1 are shown in Table IV, in which the safety rates of these systems with the original DNN controllers and with the repaired DNN controllers are compared. We notice that, in most of the cases, compared to the relatively low safety rates of the original systems, the systems with the repaired controllers, under different configurations, can achieve significantly higher safety rates. For instance, the safety rates of these systems on the retraining sets increase by at least 0.4% (for WT#2) and at most 16.5% (for AFC#1); the safety rates of these systems on the test sets increase by at least 0.4% (for WT#2) and at most 14.2% (for AFC#1). These results evidently exhibit the advantages of AUTOREPAIR in terms of safety, in that it indeed repairs the vulnerable characteristics of the original systems successfully. Moreover, the safety improvement on the test set also demonstrates the generalization ability of AUTOREPAIR.

We also notice the cases when the safety rate decreases after repair, e.g., ACC#1 under MinSat. This issue can be caused by several factors in the approach, including the quality of retraining. In practice, we can mitigate this issue by tuning the hyperparameters in the controller retraining and trying AUTOREPAIR with other strategies.

Answer to RQ1: AUTOREPAIR is able to repair unsafe DNN controllers effectively, and thus increase the safety and reliability of the entire system.

RQ2: How does the optimum selection strategy influence the effectiveness of AUTOREPAIR?

In RQ2, we study how different optimum selection strategies of the Pareto front in the signal repair phase (Section IV-C) influence the effectiveness of AUTOREPAIR. In this experiment,

we compare the performances of AUTOREPAIR under the three optimum selection strategies introduced in Section IV-C.

As shown in Table IV, MinSat outperforms the original controller in 4/6 cases, but is not effective for ACC#1; Random outperforms the original controller in 4/6 cases, and outperforms the other two strategies in 3/6 cases, but is not effective for ACC#2. These results show that MinSat and Random are effective in general, but they may also be not effective in some cases. As we analyzed in RQ1, in practice, we can improve the effectiveness of AUTOREPAIR by trying to improve the quality of retraining and trying different strategies, such that the repaired systems can achieve higher safety rates.

Moreover, Similar outperforms the original controller in 4/6 cases, and in the remaining 2 cases, it performs similarly to the original controller; however, only in 1/6 case it outperforms other two strategies. These results show that Similar is relatively weak in changing the characteristics of DNN controllers, because it always prefers the control signal patchs that have the least deviation from the original control signals.

Answer to RQ2: Among the three optimum selection strategies, MinSat and Random are effective in most cases; Similar is also effective in many cases, but it is weak in changing the characteristics of DNN controllers.

RQ3: Does AUTOREPAIR affect the performance of DNN-enabled CPS in terms of other metrics than safety?

The affection of AUTOREPAIR to the performances in different aspects of the systems is shown in Fig. 4. Specifically, we normalize the performance data in terms of different metrics, and show the performance differences between the original systems and the repaired systems. The blue line and the orange line represent the normalized performances of the original systems and the repaired systems, respectively.

As illustrated in Fig. 4, compared to the original systems, the repaired systems do not suffer much from performance degradation. For example, in ACC#2, while AUTOREPAIR 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 in our repair algorithm, we minimize the deviation of the control signal patch from the original control signal. Of course it also happens that the safety improvement comes with performance degradation in some aspects (e.g., error performance for AFC#2). The severity of such degradation depends on the importance of these metrics.

TABLE V
THE TIME COST OF AUTOREPAIR (TIME IN SECONDS)

	sim.	avg. cost	vio. ratio	total cost
ACC#1	1.42	393.81	318/10000	1.25×10^{5}
ACC#2	2.08	558.92	1800/10000	1.01×10^{6}
AFC#1	2.03	480.69	1713/10000	8.23×10^{5}
AFC#2	2.54	631.66	1339/10000	8.46×10^{5}
WT#1	1.07	171.77	1596/10000	2.74×10^{5}
WT#2	1.36	262.83	192/10000	5.05×10^4

Moreover, we observe that AUTOREPAIR can also improve the performance in some other aspects, together with safety, e.g., the steadiness for WT#1. That is because safety and the other performance metrics are consistent in some cases. For instance, the steadiness metric requires the system to stay in a safe status for a certain period, so improving safety can also improve the system performance in steadiness.

Answer to RQ3: AUTOREPAIR can improve the safety of the systems with little sacrifice in performance of other aspects. Sometimes, it can even improve the performance in the aspect that is relevant to safety.

RQ4: How is the efficiency of AUTOREPAIR?

Table V lists the details of the time costs of AUTOREPAIR, including the time cost of one system execution, the average cost of repairing one unsafe execution, the ratio of the unsafe executions in the retraining set, and the total time cost for repairing all the unsafe traces in the retraining set.

First, as shown in Table V, the cost of repairing a single system execution depends on the time cost of a single system execution, and moreover, it depends on the configuration of the multi-objective optimization algorithm. The parameter settings of NSGA-II are given in Table III; by tuning those parameters, users can balance the trade-off between efficiency and repair effectiveness. Second, the total time cost of AUTOREPAIR also depends on the ratio of the violating system executions in the collected dataset for retraining. If the system is relatively safe, less repair efforts will be paid. To sum up, as our approach can effectively improve the safety of the DNN-enabled CPS, paying such repair efforts is worthwhile.

Answer to RQ4: The time cost of AUTOREPAIR depends on the system execution time, the hyperparameters of optimization algorithms, and the safety status of the systems. For safety-critical systems, it is worth paying such repair efforts to achieve higher safety rates.

VII. RELATED WORK

DNN controllers. With the rapid development of AI, practitioners consider replacing traditional controllers with DNN controllers [4], [5], [31]–[34]. A DNN controller can be trained in different ways, including *supervised learning* [35] and *deep reinforcement learning (DRL)* [36]. Supervised learning requires to collect system executions of an original CPS to train new DNN controllers. For example, Hertneck et al. [35]

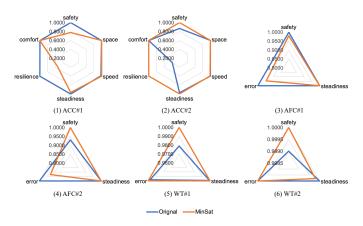


Fig. 4. Affection of AUTOREPAIR to system performances. (RQ3)

use DNNs to approximate a robust MPC controller. In contrast, DRL follows the "*trial-and-error*" paradigm, learning the best strategy via interactions with environment. A recent work [37] applies DRL for DNN controller training.

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

Adversarial training refers to the approaches that first collect adversarial examples and then use these examples with corrected labels to retrain the model. It has been extensively studied in repairing image classifiers, e.g., [38]–[43]. While our work could be treated as a specific practice of adversarial training, we differ from existing works in the way we obtain the labels for the adversarial examples, namely, due to the high complexity of DNN-enabled CPS, we search for the corrections for the control signals by optimization, such that safety is achieved at the system level.

Parameter alteration refers to the approaches that directly modify DNNs' parameters to change the inference of adversarial examples [44]–[46]. For instances, [44], [46] are the early attempts that localize the suspicious DNN parameters and then optimize them for DNN repair. Our work is orthogonal to these works since we do not manipulate DNNs' weights directly.

Repair of DNN controllers. There have been several attempts to repair DNN controllers. Yang et al. [47] propose to first attack a DNN controller and then retrain with the identified adversarial examples. However, compared to [47] in which an isolated controller is considered, we target at a physical system controlled by DNN controller; also, our safety requirement is defined on the system level. Another related work is [48], in which Zhou et al. repair a learning-based controller using an assumptive *safe* oracle. Compared to [48], our approach automates the controller repair process, by searching in the control decision space for corrections.

Safe reinforcement learning. Reinforcement learning (RL) is a typical approach for training controllers. In RL, there is a line of work [49]–[52] that aims to ensure the safety of the trained controllers; in particular, *shielding* [49] provides a similar mechanism to our framework, which repairs the actions

that trigger unsafe system behavior. However, there are several major differences between safe RL and our framework. 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 nonlinear 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, and this can also explain why we analyze and repair complete signals in an offline manner.

VIII. CONCLUSION AND FUTURE WORK

While DNN-enabled CPS have received increasing attention, how to enhance DNN controllers remains an unaddressed challenge. In this work, we propose AUTOREPAIR, a general automated repair framework that accounts for system-level safety requirements during repairing. AUTOREPAIR adopts a novel signal diagnosis and repair method that, given a system execution trace, identifies the minimal changes to the initial sequence of control signals to avoid unsafe system outputs. By applying the optimal changes to the control sequence, AUTOREPAIR obtains a new safety-assured execution trace, which can be used to repair the initial DNN controller. Our experiments show that AUTOREPAIR can effectively repair unsafe DNN controllers and increase the safety of the entire system. 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 DNN adoption in diverse industrial CPS across domains.

ACKNOWLEDGMENTS

This work was supported in part by funding from the Canada First Research Excellence Fund as part of the University of Alberta's Future Energy Systems research initiative, Canada CIFAR AI Chairs Program, Amii RAP program, the Natural Sciences and Engineering Research Council of Canada (NSERC No.RGPIN-2021-02549, No.RGPAS-2021-00034, No.DGECR-2021-00019), as well as JSPS KAKENHI Grant No.JP19H04086, No.JP20H04168, No.JP21H04877, JST-Mirai Program Grant No.JPMJMI20B8. D. Lyu is also supported by JST SPRING Grant No. JPMJSP2136.

REFERENCES

- [1] M. A. Johnson and M. H. Moradi, PID control. Springer, 2005.
- [2] E. F. Camacho and C. B. Alba, Model predictive control. Springer science & business media, 2013.
- [3] S. Alsalehi, N. Mehdipour, E. Bartocci, and C. Belta, "Neural network-based control for multi-agent systems from spatio-temporal specifications," 2021.
- [4] W. Liu, N. Mehdipour, and C. Belta, "Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints," *IEEE Control Systems Letters*, 2021.
- [5] S. Yaghoubi, G. Fainekos, and S. Sankaranarayanan, "Training neural network controllers using control barrier functions in the presence of disturbances," in 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2020, pp. 1–6.

- [6] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2011*, ser. Lecture Notes in Computer Science, P. A. Abdulla and K. R. M. Leino, Eds., vol. 6605. Springer, 2011, pp. 254–257.
- [7] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proceedings of the 22nd International Conference* on Computer Aided Verification, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 167–170.
- [8] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini, and I. Hasuo, "Two-layered falsification of hybrid systems guided by Monte Carlo Tree Search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits* and Systems, vol. 37, no. 11, pp. 2894–2905, Nov 2018.
- [9] Z. Zhang, I. Hasuo, and P. Arcaini, "Multi-armed bandits for boolean connectives in hybrid system falsification," in *Proceedings of the 31st International Conference on Computer Aided Verification, CAV 2019*, Part I, ser. Lecture Notes in Computer Science, I. Dillig and S. Tasiran, Eds., vol. 11561. Springer, 2019, pp. 401–420.
- [10] Z. Zhang, D. Lyu, P. Arcaini, L. Ma, I. Hasuo, and J. Zhao, "Effective hybrid system falsification using monte carlo tree search guided by QBrobustness," in *Proceedings of the 33rd International Conference on Computer Aided Verification, CAV 2021*, ser. Lecture Notes in Computer Science, A. Silva and K. R. M. Leino, Eds., vol. 12759. Springer, 2021, pp. 595–618.
- [11] Z. Zhang, P. Arcaini, and I. Hasuo, "Hybrid system falsification under (in)equality constraints via search space transformation," *IEEE Trans*actions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3674–3685, 2020.
- [12] Z. Zhang and P. Arcaini, "Gaussian process-based confidence estimation for hybrid system falsification," in *Proceedings of the 24th International Symposium on Formal Methods, FM 2021*, ser. Lecture Notes in Computer Science, M. Huisman, C. Păsăreanu, and N. Zhan, Eds., vol. 13047. Springer, 2021, pp. 330–348.
- [13] Z. Zhang, D. Lyu, P. Arcaini, L. Ma, I. Hasuo, and J. Zhao, "Falsifiai: Falsification of ai-enabled hybrid control systems guided by time-aware coverage criteria," *IEEE Transactions on Software Engineering*, 2022.
- [14] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed* and Fault-Tolerant Systems. Springer, 2004, pp. 152–166.
- [15] G. Ernst, P. Arcaini, I. Bennani, A. Donzé, G. Fainekos, G. Frehse, L. Mathesen, C. Menghi, G. Pedrielli, M. Pouzet, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "ARCH-COMP 2020 category report: Falsification," in Proceedings of the 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 74. EasyChair, 2020, pp. 140–152.
- [16] G. Ernst, P. Arcaini, I. Bennani, A. Chandratre, A. Donzé, G. Fainekos, G. Frehse, K. Gaaloul, J. Inoue, T. Khandait, L. Mathesen, C. Menghi, G. Pedrielli, M. Pouzet, M. Waga, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "ARCH-COMP 2021 category report: Falsification with validation of results," in *Proceedings of the 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 80. EasyChair, 2021, pp. 133–152.
- [17] A. Pnueli, "The temporal logic of programs," in 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). ieee, 1977, pp. 46–57.
- [18] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for stl," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 264–279.
- [19] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.
- [20] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, "Quantitative monitoring of stl with edit distance," Formal methods in system design, vol. 53, no. 1, pp. 83–112, 2018.
- [21] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.
- [22] Z. Zhang, P. Arcaini, and X. Xie, "Online reset for signal temporal logic monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4421–4432, 2022.
- [23] E. Bartocci, T. Ferrère, N. Manjunath, and D. Ničković, "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), 2018, pp. 197–206.

- [24] G. Ernst, P. Arcaini, G. Fainekos, F. Formica, J. Inoue, T. Khandait, M. M. Mahboob, C. Menghi, G. Pedrielli, M. Waga, Y. Yamagata, and Z. Zhang, "Arch-comp 2022 category report: Falsification with ubounded resources," in *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, ser. EPiC Series in Computing, G. Frehse, M. Althoff, E. Schoitsch, and J. Guiochet, Eds., vol. 90. EasyChair, 2022, pp. 204–221.
- [25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolu*tionary computation, vol. 6, no. 2, pp. 182–197, 2002.
- [26] Mathworks, "Adaptive cruise control system using model predictive control," https://www.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html.
- [27] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *Proceedings of the 17th international* conference on Hybrid systems: computation and control, 2014, pp. 253– 262.
- [28] Mathworks, "watertank simulink model," https://www.mathworks.com/help/slcontrol/gs/watertank-simulinkmodel.html.
- [29] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, and F.-Y. Wang, "Levenberg-marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3436–3446, 2018.
- [30] P. E. Gill, W. Murray, and M. H. Wright, Practical optimization. SIAM, 2019.
- [31] J. V. Deshmukh, J. P. Kapinski, T. Yamaguchi, and D. Prokhorov, "Learning deep neural network controllers for dynamical systems with safety guarantees," in 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2019, pp. 1–7.
- [32] S. A. Nivison and P. P. Khargonekar, "Development of a robust, sparsely-activated, and deep recurrent neural network controller for flight applications," in 2018 IEEE Conference on Decision and Control (CDC). IEEE, 2018, pp. 384–390.
- [33] A. Gilra and W. Gerstner, "Non-linear motor control by local learning in spiking neural networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1773–1782.
- [34] J. Ferlez, X. Sun, and Y. Shoukry, "Two-level lattice neural network architectures for control of nonlinear systems," in 2020 59th IEEE Conference on Decision and Control (CDC). IEEE, 2020, pp. 2198– 2203.
- [35] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control* Systems Letters, vol. 2, no. 3, pp. 543–548, 2018.
- [36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [37] J. Song, D. Lyu, Z. Zhang, Z. Wang, T. Zhang, and L. Ma, "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*, 2022, pp. 343– 352.
- [38] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.
- [39] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference* on *Machine Learning*. PMLR, 2018, pp. 5286–5295.
- [40] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!" in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 3358–3369.
- [41] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "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, 2018, pp. 175–186.
- [42] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, pp. 1–16, 2021.
- [43] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1147–1158.

- [44] J. Sohn, S. Kang, and S. Yoo, "Search based repair of deep neural networks," 2019.
- [45] H. Wang, B. Ustun, and F. Calmon, "Repairing without retraining: Avoiding disparate impact with counterfactual distributions," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6618–6627.
- [46] H. Zhang and W. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 376– 387.
- [47] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov, "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. Springer, 2022, pp. 221–236.
- [48] W. Zhou, R. Gao, B. Kim, E. Kang, and W. Li, "Runtime-safety-guided policy repair," in *International Conference on Runtime Verification*. Springer, 2020, pp. 131–150.
- [49] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings* of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.
- [50] G. Anderson, A. Verma, I. Dillig, and S. Chaudhuri, "Neurosymbolic reinforcement learning with formally verified exploration," *Advances in neural information processing systems*, vol. 33, pp. 6172–6183, 2020.
- [51] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan, "An inductive synthesis framework for verifiable reinforcement learning," in *Proceedings of the* 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019, pp. 686–701.
- [52] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," Advances in neural information processing systems, vol. 30, 2017.