The Journal of Systems and Software 229 (2025) 112475

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Fault localization of AI-enabled cyber–physical systems by exploiting temporal neuron activation☆

Deyun Lyu [a],*, Yi Li [a], Zhenya Zhang [a], Paolo Arcaini [b], Xiao-Yi Zhang [c], Fuyuki Ishikawa [b], Jianjun Zhao [a]

[a] Kyushu University, Fukuoka, Japan
[b] National Institute of Informatics, Tokyo, Japan
[c] University of Science and Technology Beijing, Beijing, China

ARTICLE INFO

Keywords:
Fault localization
Neural network controllers
Cyber–physical systems
Neuron activation

ABSTRACT

Modern *cyber–physical systems (CPS)* are evolving to integrate *deep neural networks (DNNs)* as controllers, leading to the emergence of *AI-enabled CPSs*. An inadequately trained DNN controller may produce incorrect control actions, exposing the system to safety risks. Therefore, it is crucial to localize the faulty neurons of the DNN controller responsible for the wrong decisions. However, since an unsafe system behavior typically arises from a sequence of control actions, establishing a connection between unsafe behaviors and faulty neurons is challenging. To address this problem, we propose TACTICAL that localizes faults in an AI-enabled CPS by exploiting *temporal neuron activation criteria* that capture temporal aspects of the DNN controller inferences. Specifically, based on testing results, for each neuron, TACTICAL constructs a *spectrum*, which considers the specification satisfaction and the evolution of the activation status of the neuron during the system execution. Then, starting from the spectra of all the neurons, TACTICAL applies suspiciousness metrics to compute a suspiciousness score for each neuron, from which the most suspicious ones are selected. We assess TACTICAL configured with eight *temporal neuron activation criteria*, on 3504 faulty AI-enabled CPS benchmarks spanning over different domains. The results show the effectiveness of TACTICAL w.r.t. a baseline approach.

1. Introduction

Cyber–physical systems (CPSs) refer to systems that apply computer technologies to control the behaviors of physical plants. CPSs emerge in different safety-critical application domains, such as transportation, healthcare, robotics, and smart grids. Recently, modern CPSs (e.g., autonomous driving (Bojarski et al., 2016) and robotics (Gu et al., 2017)) are increasingly adopting *deep neural networks (DNN)* as controllers for decision making, leading to the emergence of *AI-enabled CPSs* (Manzanas Lopez et al., 2023; Johnson et al., 2020; Tran et al., 2019; Huang et al., 2019; Tran et al., 2020; Zhang et al., 2023). DNN controllers have shown to provide different advantages w.r.t. classical controllers (Song et al., 2022), such as the ability to interact with complex environments and make complex decisions. With the continuous evolution of AI technologies, AI-enabled CPSs are expected to be increasingly adopted in the future.

However, despite all their performance advantages, the safety of AI-enabled CPSs constitutes a serious concern. Indeed, the DNN controller could produce some unexpected erroneous control actions, which could expose the whole system to huge safety risks. In safety-critical application domains, such as autonomous driving systems, these malfunctions can lead to catastrophic social and economic losses.

To prevent unsafe behaviors from happening, it is important to localize the faulty components in DNN controllers; specifically, in this paper we target neurons, and we consider a neuron *faulty* if it is not well-tuned, to the point of leading to failures of the whole system. Such neurons should be later fixed by approaches like DNN repair (Sohn et al., 2023; Li Calsi et al., 2023b,a; Ren et al., 2022; Henriksen et al., 2022; Tokui et al., 2022; Sun et al., 2022) to remove the wrong DNN behaviors. Different works (Usman et al., 2021; Eniser et al., 2019; Sohn et al., 2023; Ghanbari et al., 2023) have addressed the problem of *DNN fault localization*, specifically in the context of classification. In those approaches, various *neuron activation criteria* have been proposed to estimate the contribution of a neuron to the final DNN output. Some of these approaches use statistical analysis inspired by *spectrum-based*

☆ Editor: Prof Raffaela Mirandola.

* Corresponding author.
E-mail addresses: lyu.deyun.107@s.kyushu-u.ac.jp (D. Lyu), li.yi.870@s.kyushu-u.ac.jp (Y. Li), zhang@ait.kyushu-u.ac.jp (Z. Zhang), arcaini@nii.ac.jp (P. Arcaini), xiaoyi@ustb.edu.cn (X.-Y. Zhang), f-ishikawa@nii.ac.jp (F. Ishikawa), zhao@ait.kyushu-u.ac.jp (J. Zhao).

https://doi.org/10.1016/j.jss.2025.112475
Received 12 June 2024; Received in revised form 13 March 2025; Accepted 23 April 2025
Available online 12 May 2025

fault localization (SBFL) (Wong et al., 2016) to establish a correlation between incorrect DNN inferences and faulty neurons.

However, existing fault localization approaches are not applicable to DNNs used in AI-enabled CPSs. Indeed, existing DNN fault localization approaches have features that are not available in AI-enabled CPSs: the DNN is analyzed in isolation, tests are constituted by single inferences of the DNN, and an oracle is available to assess the expected classification. So, we identify three main challenges to do fault localization of DNN controllers of AI-enabled CPSs:

- *Non-modularity*: The DNN controller cannot be analyzed in isolation, but it can only be observed when it operates as a component of the whole system and interacts with the plant.
- *Temporal decision logic*: The decision logic of the DNN controller is not due to single inferences, but to sequences of inferences during the system execution.
- *Lack of ground truth*: There is no ground truth for the DNN controller, i.e., it is difficult to directly decide whether a sequence of control actions is appropriate for the control task.

Contributions. To tackle the three above challenges, we propose the framework Tactical (T*emporal* ACTI*vation-based* A*i-Cps fault* L*ocalization*), to localize the faulty neurons in DNN controllers that are responsible for the unsafe system behaviors of an AI-enabled CPS. Two key elements of Tactical that allow to tackle the above challenges are:

- As the control task is due to sequences of DNN inferences (challenge "*temporal decision logic*" described above), we need to capture the different ways in which a DNN controller can operate over time; specifically, we need to identify when the neurons contribute to the control decisions. To do this, Tactical employs eight *temporal neuron activation criteria*, that have been adapted from coverage criteria proposed for AI-enabled CPSs (Zhang et al., 2023). Such criteria define patterns of activation of the neurons over time; for example, *time persistent activation criteria* define patterns in which one or more neurons are constantly activated over a period of time, possibly identifying a constant control action; *time differential activation criteria*, instead, define patterns in which a neuron increases/decreases its activation rapidly in a short period of time, possibly identifying a control action that must be applied only for a short period of time. The intuition of Tactical is that sequences of control actions happen when the neurons manifest one of the criteria; thus, to detect faulty neurons, we analyze neurons when they are activated according to the criteria.
- Since there is no ground truth for the DNN inferences (challenge "*lack of ground truth*" described above) and so the DNN cannot be assessed in isolation (challenge "*non-modularity*" described above), we assess their correctness by checking whether they trigger correct or wrong behaviors of the plant. We do this by checking the satisfaction of a system-level specification written in Signal Temporal Logic (STL) (Donzé and Maler, 2010), that predicates about the correctness of the whole AI-enabled CPS.

Specifically, Tactical works as follows. The approach takes in input a test suite TS, a specification φ, and a temporal neuron activation criterion Cr (which, as explained above, is used to identify whether a neuron is *activated*). Given the execution of each test case $\mathbf{u}$ of the test suite TS, Tactical first checks whether $\mathbf{u}$ satisfies or violates the system specification φ; in case of violation, it identifies the moment τ in which the violation episode starts. Then, for each neuron $\mathbf{n}$, it checks whether it is activated according to criterion Cr during the execution of $\mathbf{u}$. In case of a failing test case, the criterion is checked only before the beginning of the violation episode; this is done to detect a possible causal relationship between the DNN controller's behavior captured by the coverage criterion and the specification violation. Then, for each neuron $\mathbf{n}$, Tactical constructs a *spectrum* that considers how many times $\mathbf{n}$ was activated or not (according to Cr) in passing and failing tests. Using the spectra, Tactical computes a *suspiciousness score* for each neuron, which tells the likelihood that a neuron is responsible for the unsafe system behaviors. The top s neurons having the highest suspiciousness scores are returned.

The neurons returned by Tactical can be later used to improve the DNN controller using, for example, a repair approach like the search-based repair technique proposed in Lyu et al. (2024). We will show this application in our experiments in Sections 6 and 7.

To summarize, our main novel contributions are as follows:

- We establish eight temporal neuron activation criteria that assess whether a neuron contributed to the system behavior; while the criteria were originally proposed for testing, this is the first work that uses them for fault localization. We investigate which criterion is more effective in characterizing behaviors that are correlated with system failures, and, therefore, is more suitable for fault localization;
- We propose the fault localization framework Tactical that exploits the temporal neuron activation criteria to establish a connection between neurons and unsafe system behaviors; while existing DNN fault localization approaches assess single inferences of a DNN, Tactical assesses sequences of DNN inferences. Moreover, since there is no ground truth for control decisions of the DNN controller, their correctness is assessed by checking a specification that predicates over the whole system;
- We experimentally evaluate the effectiveness of Tactical over 3504 faulty benchmarks, and the results show that Tactical can successfully localize the artificial faults injected to the DNN controllers. We further show that the neurons identified by Tactical can be effectively used to repair the DNN controller.

Paper structure. Section 2 provides the necessary background. Section 3 provides an overview of the approach, that is described in details in Sections 4 and 5. Section 6 introduces the design of the experiments, and Section 7 discusses experimental results. Then, Section 8 discusses threats that may affect the validity of the approach. Finally, Section 9 reviews related work, and Section 10 concludes the paper.

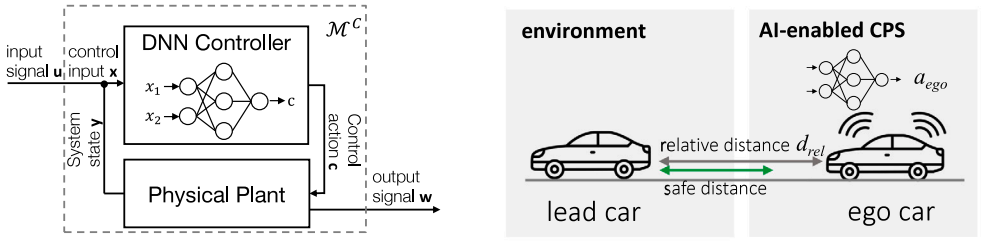
2. Preliminaries

2.1. AI-enabled cyber–physical systems

Definition 1 (*AI-enabled CPS*)**.** An AI-enabled CPS $\mathcal{M}^{\mathcal{C}}$ consists of a DNN controller $\mathcal{C}$ and a physical plant $\mathcal{M}$, as shown in Fig. 1. At a time instant t, the controller $\mathcal{C}$ receives as input $\mathbf{x}(t)$, which consists of an external signal $\mathbf{u}(t)$ and the plant state $\mathbf{y}(t)$, and outputs a *control action* $\mathbf{c}(t)$ to trigger the state evolution of the plant $\mathcal{M}$. On the side of plant $\mathcal{M}$, the state evolution $\dot{\mathbf{y}}(t)$ at time t is decided by both the plant state $\mathbf{y}(t)$ and the control action $\mathbf{c}(t)$. Overall, the whole system $\mathcal{M}^{\mathcal{C}}$ maps an input signal $\mathbf{u}$ to an output signal $\mathbf{w}$, where $\mathbf{w}$ involves the plant state variables in $\mathbf{y}$ that are observable from external environment. ◁

Note that we consider DNN controllers specifically for decision making (e.g., accelerate/decelerate, steering), and not for other activities involved in control such as perception; indeed, these latter DNNs can be tested and debugged in isolation by several approaches (Ma et al., 2018a; Sohn et al., 2023; Zhang et al., 2022).

Example 1. We use the *Adaptive Cruise Control* (ACC) system (later introduced in Section 6.2) to illustrate the operation of an AI-enabled CPS. ACC is an advanced control system that automatically adjusts the speed of a vehicle (*ego car*) to maintain a safe distance from the preceding vehicle (*lead car*), as shown in Fig. 1(b). The *ego car* is considered as the plant, i.e., the physical actuator in the system. The lead car is considered as the surrounding environment, and its speed



(a) AI-enabled CPS (b) Adaptive Cruise Control (ACC)

Fig. 1. AI-enabled CPS model and an example ACC.

and position perceived by the ego car are treated as external input signal $\mathbf{u}(t)$ of ACC. The signal $\mathbf{y}(t)$ fed back from the plant is the speed and position of the ego car. The two signals are combined into $\mathbf{x}(t)$ as the input sent to the controller C for decision making. The controller C outputs a control action $\mathbf{c}(t)$ back to the plant to actuate its state evolution. The output signal $\mathbf{w}(t)$ of the system involves the position of the ego car, by which the relative distance between the two cars can be computed. ◁

In an AI-enabled CPS, the DNN controller serves as the central component determining the system's evolution. In this paper, by following related literature (Tran et al., 2020; Katz et al., 2017; Yang et al., 2022; Zhang et al., 2023), our focus is on fully connected DNNs, defined as follows.

Definition 2 (*DNN Controller*)**.** A DNN controller C is a fully-connected DNN that consists of an *input* layer, an *output* layer, and L *hidden* layers. At each timestamp, C takes as input a vector $\vec{x}$ from the input layer, and computes an output c as the control decision at the output layer. Each hidden layer consists of J_i ($i \in \{1, \ldots, L\}$) *neurons*. Given $\vec{x}$ as the input for C, a neuron $\mathbf{n}_{ij}$ ($i \in \{1, \ldots, L\}, j \in \{1, \ldots, J_i\}$) at the ith layer outputs $\phi^{\mathbf{n}_{ij}}(\vec{x}) \in \mathbb{R}$, by taking as input the neuron output $\phi^{\mathbf{n}_{(i-1)j'}}(\vec{x})$ from the neurons at the $(i-1)$th layer, as follows:

$$\phi^{\mathbf{n}_{ij}}(\vec{x}) \;=\; \sigma\left(b + \sum_{j'=1}^{J_{i-1}} \omega_{j'} \cdot \phi^{\mathbf{n}_{(i-1)j'}}(\vec{x})\right) \tag{1}$$

where $\omega_{j'} \in \mathbb{R}$ is a weight (which is the j'th component of the weight vector $\omega \in \mathbb{R}^{J_{i-1}}$ for the neuron $\mathbf{n}_{ij}$) and $b \in \mathbb{R}$ is the bias for the neuron $\mathbf{n}_{ij}$, and σ is a non-linear function known as *activation function*. Common choices of activation function σ include ReLU, sigmoid, tanh, etc. The output c of the DNN controller C is computed by taking a weighted sum of the neuron outputs $\phi^{\mathbf{n}_{Lj}}(\vec{x})$ ($j \in \{1, \ldots, J_L\}$) at the last hidden layer. ◁

2.2. System specification

A specification serves as a formal description of a system's expected behavior. It plays a crucial role in defining the conditions that a system must meet, encompassing functionalities, constraints, performance expectations, safety requirements, etc. Specifications are typically expressed in formal languages, ensuring accuracy, clarity, and verifiability. In this context, the systems presented in this paper adopt the widely recognized *Signal Temporal Logic (STL)* (Donzé and Maler, 2010), renowned for its expressivity for continuous dynamic system properties. Below, we overview STL syntax and semantics.

Definition 3 (*STL Syntax*)**.** Let $\vec{w} \in \mathbb{R}^d$ be a vector. In STL, an *atomic proposition* is represented as $\alpha :\equiv (f(\vec{w}) > 0)$, in which $f : \mathbb{R}^d \to \mathbb{R}$ is a function that maps $\vec{w}$ to a real number. The syntax of an STL formula φ is defined as follows:

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

Here, I is a time interval $[a, b]$, where $a, b \in \mathbb{R}$ and $a < b$. $\Box_I$, $\Diamond_I$, $\mathcal{U}_I$ are temporal operators *always*, *eventually*, and *until*, which allow to express complex temporal properties. The syntax of STL defines a set of rules by which one can decide whether an STL formula is valid. ◁

The (Boolean) semantics (Fainekos and Pappas, 2009; Donzé and Maler, 2010) of STL characterizes the satisfaction condition for a signal $\mathbf{w}$ to an STL specification φ. We introduce the formal definition of STL semantics.

Definition 4 (*STL Semantics*)**.** Given a system output signal $\mathbf{w}$ and an STL specification φ, STL semantics defines the conditions to determine whether $\mathbf{w}$ satisfies φ (denoted as $\mathbf{w} \vDash \varphi$). Formally, the satisfaction of φ by $\mathbf{w}$ can be determined by checking the following conditions recursively:

$$\mathbf{w} \vDash \alpha \iff f(\mathbf{w}(0)) > 0 \qquad \mathbf{w} \vDash \neg\varphi \iff \mathbf{w} \nvDash \varphi$$

$$\mathbf{w} \vDash \varphi_1 \wedge \varphi_2 \iff \mathbf{w} \vDash \varphi_1 \wedge \mathbf{w} \vDash \varphi_2$$

$$\mathbf{w} \vDash \Box_I \varphi \iff \forall t \in I.\ (\mathbf{w}^t \vDash \varphi)$$

$$\mathbf{w} \vDash \varphi_1 \mathcal{U}_I \varphi_2 \iff \exists t \in I.\ \left(\mathbf{w}^t \vDash \varphi_2 \wedge \forall t' \in [0, t).\ (\mathbf{w}^{t'} \vDash \varphi_1)\right)$$

where $\mathbf{w}^t$ denotes the t-shift of $\mathbf{w}$, namely, for an arbitrary $t' \in [0, T-t]$, it holds that $\mathbf{w}^t(t') = \mathbf{w}(t+t')$. The semantics of the omitted operators can be derived from those of the existing operators, based on their syntactic equivalence relations. Intuitively, the semantics of STL defines the meaning of different operators. For example, $\mathbf{w} \vDash \Box_I \varphi$ requires that $\mathbf{w}^t$ satisfies φ at each $t \in I$, and, therefore, this operator can be interpreted as "always". ◁

Example 2. Consider ACC in Example 1. The specification $\varphi \equiv \Box_{[0,T]}(d_{rel} \geq 10)$ is an STL formula. In terms of the syntax, the formula is defined using a $\Box$ operator, nested with an atomic proposition. In terms of the semantics, the formula requires that d_{rel} (i.e., the relative distance between the ego car and the leading car) is always greater than the safe distance 10 during the time interval $[0, T]$, in order to keep the system safe. ◁

3. Overview of the proposed approach

In this section, we introduce the problem that we want to target (Section 3.1), and overview the proposed solution (Section 3.2).

3.1. Motivation and problem statement

In this work, we target failures of AI-enabled CPSs. We use Example 3 to show the main source of system failures.

Example 3. Consider ACC in Example 1 and its specification given in Example 2. Fig. 2 shows the trace of a_{ego}, i.e., the acceleration of the ego car over time given by the controller; the figure also shows the trace of d_{rel} over time, which is the output of the system. We notice that the system specification is violated as, at time τ, d_{rel} starts to be smaller than the safe distance 10. This can be due to the fact that, in the interval $[0, \tau]$ preceding the violation, the DNN controller does not properly decide to decelerate so to guarantee a safe distance. ◁

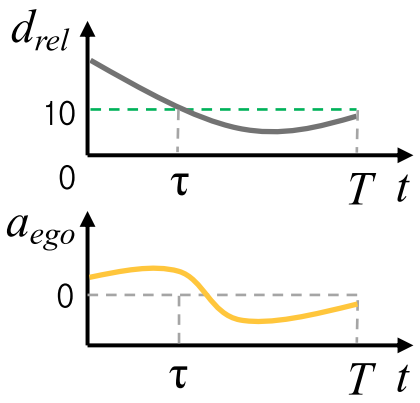


Fig. 2. An execution that violates $\varphi \equiv \square_{[0,T]}(d_{rel} \geq 10)$.

From the previous example, we understand that wrong decisions of the DNN controller are responsible for system failures. However, due to the opaque inference logic of DNNs, it is hard to know why such wrong decisions occurred. As explained in Definition 2, the process of decision making for control, i.e., the inference of the DNN controller, is primarily decided by the DNN parameters, namely, the *weights* and *biases* in the neurons of DNN controllers. If these parameters are inadequately tuned, some neurons will produce erroneous outputs, so influencing the overall output of the DNN controller that leads to wrong control decisions. Then, these erroneous control decisions propagate throughout the system execution, ultimately resulting in system failures.

The objective of this paper is to localize those "faulty neurons" whose erroneous outputs trigger wrong decisions of the DNN controller that lead to system failures. By identifying these faulty neurons, we aim to provide engineers with an insight into the root causes of system failures, aiding them in the improvement of the system by, e.g., automated repair.

Problem statement. Given an AI-enabled CPS $\mathcal{M}^C$ equipped with a DNN controller C, a specification φ, and a test suite TS, our aim is to identify the subset of the neurons in C whose erroneous outputs lead to the violation of φ by $\mathcal{M}^C$.

3.2. Overview

In this section, we overview our proposed fault localization approach for DNN controllers. We first review an existing fault localization approach for DNNs from which we take inspiration. Then, we explain the limits of this approach and why it cannot be used for fault localization of DNN controllers.

3.2.1. DNN fault localization

For classical code, *spectrum-based fault localization (SBFL)* (Wong et al., 2016) is an efficient approach to localize "suspicious" statements that are likely to be faulty. Given a test suite and its execution results, SBFL builds a *spectrum* for each statement, telling how often the statement is executed or not in passing and failing tests. Starting from the spectra, a *suspiciousness metric* is used to identify the statements that are more likely to be faulty.

By borrowing the same concept of SBFL, Eniser et al. (2019) proposed *DeepFault* that tries to identify faulty neurons in a DNN classifier. To do this, it identifies when a neuron is "relevant" in a DNN inference (the counterpart of statement execution in SBFL for classical code): if the *neuron activation* (Pei et al., 2019), measured in terms of neuron outputs, surpasses a given threshold, the neuron is deemed to be activated and, therefore, relevant. *DeepFault* then builds spectra for the different neurons considering activation and whether the DNN correctly classifies an input or not. Finally, it uses classical suspiciousness metrics to assign a suspiciousness score to each neuron and ranks them accordingly.

3.2.2. Limitations of classic DNN fault localization

Existing fault localization approaches as *DeepFault* are not applicable to AI-enabled CPSs for three main reasons:

- *Non-modularity.* The DNN controller can only be analyzed as component of the whole system, and not in isolation.
- *Lack of ground truth.* Differently from standalone DNNs (e.g., image classifiers) considered by *DeepFault*, for a DNN controller C embedded in an AI-enabled CPS $\mathcal{M}^C$, we do not have the ground truth. Specifically, given a sequence of control actions $\mathbf{c}(t)$, we do not know whether they are "correct". Nevertheless, the controller drives the evolution of system $\mathcal{M}^C$, for which we have requirements specified in terms of a formal specification φ. Our intuition is that the satisfaction/violation of φ can be used to "indirectly" assess the correctness of the control actions.
- *Temporal decision logic.* In *DeepFault*, fault localization is performed by considering the activation of each neuron in each single inference of the DNN, as the correctness of each inference can be assessed individually. Such type of fault localization is not applicable to AI-enabled CPSs $\mathcal{M}^C$, where the plant $\mathcal{M}$ is controlled by a *sequence* of control actions of C, and the correctness of a system execution (i.e., its satisfaction to φ) is jointly decided by all those control actions. In this case, a system failure is often not attributed to a specific control action at a given instant, but rather to a sub-sequence of control actions over a time interval. Thus, we need new notions of neuron activation that consider the consecutive inferences of the DNN.

3.2.3. Proposed fault localization framework

In this paper, we propose a fault localization approach for DNN controllers used in AI-enabled CPS, that tackles the issues reported in Section 3.2.2.

First of all, to capture the temporal decision logic, we introduce different *temporal neuron activation* criteria, inspired by temporal coverage criteria for DNN controllers (Zhang et al., 2023). These criteria characterize the temporal behavior of DNN controller inferences in different ways, including *time instant* activation, *time persistent* activation, and *time differential* activation. We will introduce the detailed definitions of these criteria in Section 4.

Based on these criteria, we propose the fault localization approach Tactical (T*emporal* ACTI*vation-based* A*i-Cps fault* L*ocalization*) shown in Fig. 3 and described in Section 5. Tactical takes as input a test suite TS of input signals for an AI-enabled CPS $\mathcal{M}^C$ and a system specification φ. Tactical executes the tests, and builds spectra for the different neurons by considering their activation using a temporal neuron activation criterion Cr and the satisfaction/violation of φ over the tests. Neurons are then sorted in terms of *suspiciousness* of being faulty, by using classic SBFL suspiciousness metrics.

4. Temporal neuron activation criteria

The goal of our fault localization approach is to identify which neurons are responsible for behaviors of the DNN controller that lead to unsafe states of the AI-enabled CPS. So, the first step is to know when a neuron is "involved" in a control decision.

In classical fault localization for code (Wong et al., 2016), the coverage of a statement is used to identify the involvement of the statement in a computation. However, for DNNs, there is not such a crisp notion. Therefore, fault localization techniques for DNNs (Sohn et al., 2023; Eniser et al., 2019; Li Calsi et al., 2023b) use *neuron activation* as a heuristic measure to define involvement of a neuron in a DNN inference. To characterize the neuron activation in a DNN classifier, it suffices to measure the neuron output, that identifies the numerical contribution to the DNN output.

However, in an AI-enabled CPS, a control decision (that could lead to a wrong behavior of the system) is given by a sequence of control

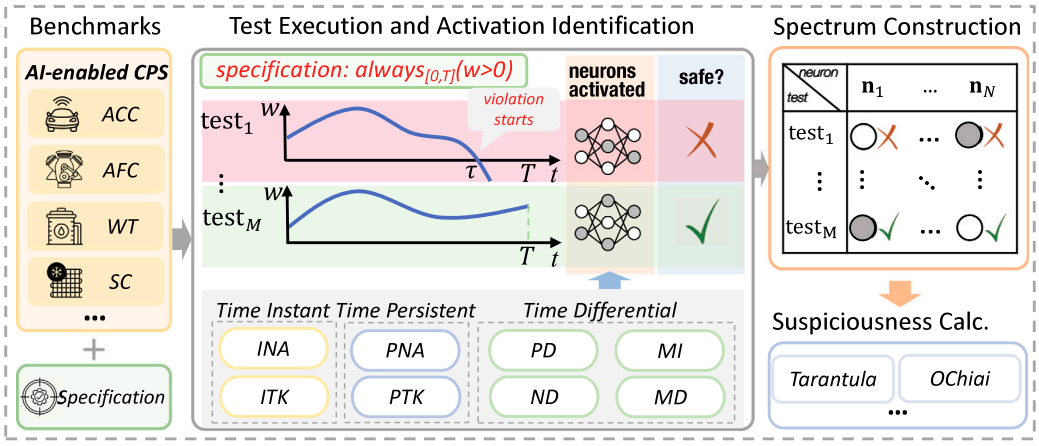


Fig. 3. TACTICAL – Workflow of the approach.

actions over time. So, in order to capture when the neurons contribute to the control actions, we need to define neuron activation criteria that consider sequences of DNN controller inferences.

To this end, we present a set of *temporal neuron activation criteria*, adapted from Zhang et al. (2023); in the following, for brevity, by "criterion" we mean "temporal neuron activation criterion". In contrast to the simple definition of neuron activation in DNN classifiers (Eniser et al., 2019), these criteria integrate temporal aspects of neuron behaviors into the neuron activation definition from different perspectives: *time instant* activation, *time interval* activation, and *time differential* activation, to enable an effective discovery of hidden temporal patterns in neuron outputs.

In the work by Zhang et al. (2023), these activation criteria are used to measure the coverage of the neurons that contribute considerably to control action sequences; these criteria identify different control patterns, and covering them allows to guarantee a good exploration of the behavior of the whole AI-enabled CPSs. In this paper, we adopt a similar conceptual usage of these criteria: we use them to characterize different control patterns, and link specific neuron activations to the system failures.

In the following, we give the formal definitions of the three classes of criteria. Let $\mathbf{n}$ be a neuron at the ith layer of a DNN controller C, and $\mathbf{x} = \langle \mathbf{x}(0), \ldots \mathbf{x}(\tau) \rangle$ be a sequence of DNN inputs over a time interval $[0, \tau]$. Given an activation criterion Cr, we define a predicate Θ_{Cr} that returns a Boolean verdict in $\{\top, \bot\}$ (i.e., *true* or *false*) telling whether $\mathbf{n}$ is activated under $\mathbf{x}$ over $[0, \tau]$, by the criterion Cr. In the following, predicates Θ_{Cr} are implicitly parameterized with $\mathbf{n}$ and $\mathbf{x}$.

Time instant activation. For the criteria of this class, a neuron $\mathbf{n}$ is deemed as *activated* if, at a particular time instant, its output meets a specified condition that identifies a substantial relevance of the neuron output to the overall DNN output.

Instant Neuron Activation (INA): In this criterion, the condition (inspired by *neuron coverage* (Pei et al., 2019)) assesses whether the neuron output is greater than a given threshold h. Θ_{INA} is defined as:

$$\Theta^{(h)}_{\mathsf{INA}} := \Big(\exists t \in [0, \tau], \phi^{\mathbf{n}}(\mathbf{x}(t)) > h\Big)$$

The neuron $\mathbf{n}$ is deemed as activated if its output surpasses h at some instant t during $[0, \tau]$.

Instant Top-k Neuron Activation (ITK): In this criterion, the condition assesses whether the neuron output is greater than the outputs of most of the other neurons at the same layer. Θ_{ITK} is defined as:

$$\Theta^{(k)}_{\mathsf{ITK}} := \Big(\exists t \in [0, \tau], \mathbf{n} \in \mathsf{top}^k(\mathbf{x}(t), i)\Big)$$

where $\mathsf{top}^k(\mathbf{x}(t), i)$ denotes the set of neurons $\mathbf{n}'$ such that their neuron outputs $\phi^{\mathbf{n}'}(\mathbf{x}(t))$ rank in the top^k greatest outputs among all the neurons at the ith layer.

Time persistent activation. Similarly to time instant activation criteria, also the criteria in this class rely on a condition that identifies a substantial relevance of the neuron output to the overall DNN output. However, unlike time instant activation criteria, in this class, a neuron $\mathbf{n}$ is deemed as *activated* if not only its output can satisfy the condition, but also the condition satisfaction persists for a time interval.

Persistent Neuron Activation (PNA): We first define a predicate $\mathsf{P}_{\mathsf{PNA}}$ parameterized by three parameters (a time instant t, a threshold h, and a time interval Δ) as follows:

$$\mathsf{P}_{\mathsf{PNA}}(t, h, \Delta) := \Big(\forall t' \in [t, t + \Delta], \phi^{\mathbf{n}}(\mathbf{x}(t')) > h\Big)$$

$\mathsf{P}_{\mathsf{PNA}}$ holds if, during the time interval $[t, t + \Delta]$, the neuron output $\phi^{\mathbf{n}}(\mathbf{x}(t'))$ is persistently greater than h. Subsequently, we define Θ_{PNA} as follows:

$$\Theta^{(h,\Delta)}_{\mathsf{PNA}} := \Big(\exists t \in [0, \tau - \Delta], \mathsf{P}_{\mathsf{PNA}}(t, h, \Delta)\Big)$$

The neuron $\mathbf{n}$ is activated if there exists a time instant t that ensures the satisfaction of $\mathsf{P}_{\mathsf{PNA}}$.

Persistent Top-k Neuron Activation (PTK): We first define predicate $\mathsf{P}_{\mathsf{PTK}}$ parameterized by three parameters (a time instant t, an integer k, and a time interval Δ) as follows:

$$\mathsf{P}_{\mathsf{PTK}}(t, k, \Delta) := \Big(\forall t' \in [t, t + \Delta], \mathbf{n} \in \mathsf{top}^k(\mathbf{x}(t'), i)\Big)$$

$\mathsf{P}_{\mathsf{PTK}}$ holds if, during the time interval $[t, t + \Delta]$, the neuron output persistently ranks in the top k greatest outputs among all the neurons at the ith layer. Subsequently, we define Θ_{PTK} as follows:

$$\Theta^{(k,\Delta)}_{\mathsf{PTK}} := \Big(\exists t \in [0, \tau - \Delta], \mathsf{P}_{\mathsf{PTK}}(t, k, \Delta)\Big)$$

The neuron $\mathbf{n}$ is activated if there exists a time instant t that ensures the satisfaction of $\mathsf{P}_{\mathsf{PTK}}$.

Time differential activation. For the criteria of this class, a neuron $\mathbf{n}$ is deemed as *activated* if the change of the neuron output over a time interval follows a specified pattern. The pattern identifies a specific way in which the neuron contributes to the overall DNN output. We consider two types of patterns, described below. For convenience, we introduce the following notion to measure the differential behavior between two time instants: $\phi^{\mathbf{n}}_{\langle t_1, t_2 \rangle}(\mathbf{x}) = \phi^{\mathbf{n}}(\mathbf{x}(t_2)) - \phi^{\mathbf{n}}(\mathbf{x}(t_1))$.

Positive/Negative Differential Neuron Activation (PD/ND): These criteria require that the neuron output increases or decreases more than a threshold over a time interval. We first define predicates P_{PD} and P_{ND}:

$$\mathsf{P}_{\mathsf{PD}}(t, h, \Delta) := \Big(\exists t_1, t_2 \in [t, t + \Delta], t_1 < t_2 \wedge \phi^{\mathbf{n}}_{\langle t_1, t_2 \rangle}(\mathbf{x}) > h\Big)$$

$$\mathsf{P}_{\mathsf{ND}}(t, h, \Delta) := \Big(\exists t_1, t_2 \in [t, t + \Delta], t_1 < t_2 \wedge \phi^{\mathbf{n}}_{\langle t_2, t_1 \rangle}(\mathbf{x}) > h\Big)$$

$\mathsf{P_{PD}}$ (or $\mathsf{P_{ND}}$) holds if, during the time interval $[t, t+\Delta]$, there exist two instants t_1 and t_2 such that the increase (or decrease) of neuron output between t_1 and t_2 is greater than h. Subsequently, we define Θ_{PD} and Θ_{ND} as follows:

$$\Theta_{\mathsf{PD}}^{(h,\Delta)} := \Big(\exists t \in [0, \tau-\Delta], \mathsf{P_{PD}}(t, \Delta, h)\Big)$$
$$\Theta_{\mathsf{ND}}^{(h,\Delta)} := \Big(\exists t \in [0, \tau-\Delta], \mathsf{P_{ND}}(t, \Delta, h)\Big)$$

The neuron $\mathbf{n}$ is activated in terms of PD (or ND), if there exists a time instant t that ensures the satisfaction of $\mathsf{P_{PD}}$ (or $\mathsf{P_{ND}}$).

Monotonic Increase/Decrease Neuron Activation (MI/MD): These criteria require that, during a time interval, the neuron output should monotonically increase or decrease. We first define predicates $\mathsf{P_{MI}}$ and $\mathsf{P_{MD}}$ as:

$$\mathsf{P_{MI}}(t, \Delta) := \Big(\forall t_1, t_2 \in [t, t+\Delta], (t_2 - t_1) \cdot \phi^{\mathbf{n}}_{\langle t_1, t_2\rangle}(\mathbf{x}) > 0\Big)$$
$$\mathsf{P_{MD}}(t, \Delta) := \Big(\forall t_1, t_2 \in [t, t+\Delta], (t_2 - t_1) \cdot \phi^{\mathbf{n}}_{\langle t_1, t_2\rangle}(\mathbf{x}) < 0\Big)$$

$\mathsf{P_{MI}}$ (or $\mathsf{P_{MD}}$) holds if, during the time interval $[t, t+\Delta]$, the neuron output keeps increasing (decreasing) monotonically. Subsequently, we define Θ_{MI} and Θ_{MD} as follows:

$$\Theta_{\mathsf{MI}}^{(\Delta)} := \Big(\exists t \in [0, \tau-\Delta], \mathsf{P_{MI}}(t, \Delta)\Big)$$
$$\Theta_{\mathsf{MD}}^{(\Delta)} := \Big(\exists t \in [0, \tau-\Delta], \mathsf{P_{MD}}(t, \Delta)\Big)$$

The neuron $\mathbf{n}$ is activated in terms of MI (or MD), if there exists a time instant t that ensures the satisfaction of $\mathsf{P_{MI}}$ (or $\mathsf{P_{MD}}$).

All the criteria presented above are proxy measures to assess the involvement of a neuron in the DNN temporal behavior. Some of them will be more or less effective in doing so. Since we will use them in our proposed fault localization approach (see Section 5), they will affect its effectiveness. Therefore, in the experiments (see Sections 6 and 7), we will assess which criterion is more effective.

5. TACTICAL – Proposed approach

We assume to have an AI-enabled CPS $\mathcal{M}^C$ embedded with a DNN controller C, and a system specification φ that could be violated by $\mathcal{M}^C$. We also assume to have a test suite TS that consists of M test cases (i.e., external input signals for $\mathcal{M}^C$), each of which can be executed by the system for a simulation time T (T guarantees that φ's satisfaction can be assessed).

The proposed approach TACTICAL is shown in Alg. 1. It takes as input the system $\mathcal{M}^C$, the specification φ, the test suite TS, a temporal neuron activation criterion Cr (see Section 4), and the number s of neurons to be returned as the suspicious ones.

Overview. Overall, TACTICAL assesses the "suspiciousness" of each neuron to be the cause of the specification violation, by constructing its *execution spectrum*, which records the information regarding *neuron activation* and *specification satisfaction/violation*, obtained by executing the test cases in the test suite. The execution spectrum of a neuron $\mathbf{n}_i$ is composed of four components $\langle ap, af, np, nf\rangle$, whose meaning is:

- ap: number of *p*assing tests in which $\mathbf{n}_i$ is *a*ctivated;
- af: number of *f*ailing tests in which $\mathbf{n}_i$ is *a*ctivated;
- np: number of *p*assing tests in which $\mathbf{n}_i$ is *n*ot activated;
- nf: number of *f*ailing tests in which $\mathbf{n}_i$ is *n*ot activated.

At the beginning, the spectrum $\mathsf{spectrum}_i$ of each neuron is initialized to $\langle 0,0,0,0\rangle$ (Lines 2–3). The spectra are later updated after the execution of each test case $\mathbf{u}$, considering the neuron activation and specification assessment for $\mathbf{u}$ (Line 9).

Specification assessment. Each test $\mathbf{u}$ in the test suite TS is executed, by feeding it to the system $\mathcal{M}^C$, and obtaining the system output signal $\mathbf{w}$ and a matrix Φ, where $\Phi(\mathbf{n}_i, t)$ stores the output of neuron $\mathbf{n}_i$ at time instant $t \in [0, T]$ (Line 5).

Algorithm 1: TACTICAL – the proposed approach

Input: A system $\mathcal{M}^C$ with a DNN controller C consisting of N neurons; an STL specification φ; a test suite TS; a neuron activation criterion Cr; a suspiciousness metric SMet; number s of suspicious neurons
Output: A set SusN of s suspicious neurons

1 **function FaultLocalization**($\mathcal{M}^C, \varphi, \mathsf{Cr}, \mathsf{SMet}, s$)
2 **for** $i \in \{1, \ldots, N\}$ **do**
3 $\mathsf{spectrum}_i \leftarrow \langle 0,0,0,0\rangle$ // init. spectrum $\langle ap,af,np,\rangle$
4 **for** $\mathbf{u} \in TS$ **do**
5 $\mathbf{w}, \Phi \leftarrow \mathcal{M}^C(\mathbf{u})$ // system execution
6 identify τ as { the start of violation episode, if $\mathbf{w} \not\models \varphi$; the execution time T, otherwise
7 **for** $i \in \{1, \ldots, N\}$ **do**
8 $\Theta \leftarrow$ **ActivationComputation**($\Phi_i, \tau, \mathsf{Cr}$)
9 { $\mathsf{spectrum}_i(ap) \leftarrow \mathsf{spectrum}_i(ap) + 1$ if $\mathbf{w} \models \varphi \wedge \Theta = \top$; $\mathsf{spectrum}_i(af) \leftarrow \mathsf{spectrum}_i(af) + 1$ if $\mathbf{w} \not\models \varphi \wedge \Theta = \top$; $\mathsf{spectrum}_i(np) \leftarrow \mathsf{spectrum}_i(np) + 1$ if $\mathbf{w} \models \varphi \wedge \Theta = \bot$; $\mathsf{spectrum}_i() \leftarrow \mathsf{spectrum}_i() + 1$ if $\mathbf{w} \not\models \varphi \wedge \Theta = \bot$
10 $\mathsf{SusScores} \leftarrow \emptyset$ // init. neurons' susp. scores
11 **for** $i \in \{1, \ldots, N\}$ **do**
12 $\mathsf{nSusScore}_i \leftarrow$ **SuspScoreComputation**($\mathsf{spectrum}_i, \mathsf{SMet}$)
13 $\mathsf{SusScores} \leftarrow \mathsf{SusScores} \cup \{\mathsf{nSusScore}_i\}$
14 $\mathsf{SusN} \leftarrow$ **SelectSuspNeurons**($\mathsf{SusScores}, s$)
15 **return** SusN

Given the output signal $\mathbf{w}$, the approach checks whether it satisfies the specification φ or not. In case of violation, it analyzes the output signal $\mathbf{w}$ and obtains the time instant τ that identifies the *start* of the specification violation (Line 6). For instance, in Example 3 (Fig. 2), the violation starts at τ when d_{rel} starts to be less than the safe distance 10, so τ can be deemed as the starting instant of the specification violation. In general, the identification of such a τ can be achieved by *trace diagnostics* as introduced in Bartocci et al. (2018). The incorrect control actions made by the controller C must have happened before τ (i.e., in $[0, \tau)$); their effect propagates throughout the execution and results in the violation at τ. If $\mathbf{w}$ satisfies the specification, the approach sets τ as the execution time T (Line 6).

Activation identification. For each neuron $\mathbf{n}_i$, the approach checks whether $\mathbf{n}_i$ is activated under the neuron activation criterion Cr given as input, using function **ActivationComputation** (Line 8); the function takes as input Φ_i, the neuron outputs of $\mathbf{n}_i$ during the execution in $[0, T]$, the start of specification violation τ, and the criterion Cr. Then, according to the definition of the criterion in Section 4, it computes $\Theta \in \{\top, \bot\}$ telling if $\mathbf{n}_i$ is activated during interval $[0, \tau]$.

Spectrum update. Based on the information about specification satisfaction and neuron activation of $\mathbf{n}_i$, the spectrum of $\mathbf{n}_i$ is updated accordingly, i.e., one of the components $\langle ap, af, np, nf\rangle$ is incremented by 1 (Line 9).

Suspiciousness computation. Finally, using the spectra of all the neurons, TACTICAL estimates the correlation between the behaviors of the neurons and the specification satisfaction/violation. Specifically, it computes the *suspiciousness score* of each neuron by adopting *suspiciousness metrics* borrowed from SBFL (Wong et al., 2016), in order to identify the neurons that are likely responsible for the system specification violation. Table 1 presents six widely used suspiciousness metrics in SBFL; indeed, according to Pearson et al. (2017), Tarantula, Ochiai, D*, and Op2 are among the most effective SBFL metrics[1]; we also

[1] Note that, in this paper, we only consider SBFL metrics. So, we do not consider mutation-based metrics such as Metallix and MUSE, or model-based ones such as Barinel.

Table 1
Adopted suspiciousness metrics SMet.

Tarantula (Jones et al., 2002)	Ochiai (Abreu et al., 2006)	D* (Wong et al., 2014)	Jaccard (Abreu et al., 2009)	Kulczynski2 (Naish et al., 2011)	Op2 (Naish et al., 2011)
$\frac{\frac{af}{af+}}{\frac{af}{af+}+\frac{ap}{ap+np}}$	$\frac{af}{\sqrt{(af+)(af+ap)}}$	$\frac{af^*}{ap+}$	$\frac{af}{af++ap}$	$\frac{1}{2}\left(\frac{af}{af+}+\frac{af}{af+ap}\right)$	$af-\frac{ap}{ap+np+1}$

In D*, we set * as 2 and 3, by following Wong et al. (2012) and Wong et al. (2014).

consider Jaccard and Kulczynski2 because they are other two widely used metrics in SBFL (Wong et al., 2016). The main idea underpinning these metrics is that, the more frequently a neuron is activated in failing tests, and the less frequently it is activated in passing tests, the more suspicious the neuron is. This process is illustrated in Lines 10–14 of Alg. 1. For each neuron $\mathbf{n}_i$, Tactical calculates the *suspiciousness score* of $\mathbf{n}_i$, by applying the selected suspiciousness metric SMet to the spectrum spectrum_i of the neuron (Line 12). Then, all the scores are collected in the set SusScores (Line 13), from which the top s greatest neurons are selected as the final set SusN of suspicious neurons (Line 14).

In our experiments, we will experiment will all the six metrics in Table 1; for D*, we will experiment with $*$ set to 2 and to 3 (i.e., D^2 and D^3). So, in total, we will experiment with seven suspiciousness metrics.

5.1. Hyperparameter selection of the criteria

Tactical relies on the use of the criteria (see Section 4) that can be tuned with some hyperparameters. The selection of hyperparameters is critical, as it may affect the performance of Tactical. In this section, we propose a heuristic strategy to select reasonable ranges for the hyperparameters.

Activation threshold h. The activation threshold h is used to determine whether the output of a neuron at a time instant t exceeds a specific value. If it does, the neuron is considered activated; otherwise, it is considered not activated. Hence, to select h, the first step is to narrow down the hyperparameter space to the range of neuron outputs. Specifically, given an AI-enabled CPS $\mathcal{M}^C$, a specification φ and a test suite TS, we execute the system and record the neuron outputs at each instant of each single execution. Then, we obtain the minimum neuron output ϕ_{min} and maximum neuron output ϕ_{max}. The threshold h should be selected in the range $[\phi_{min}, \phi_{max}]$, by considering how strict the criterion should be (higher values make the criterion more strict). In the experiments (RQ1), we will select three representative hyperparameters values for each system, and conduct a study to investigate how the selection of hyperparameters influences the effectiveness of Tactical.

Time interval Δ. Hyperparameter Δ indicates for how long a condition must hold in order for the neuron to be considered activated (e.g., in PNA, the neuron output must be higher than h for more than Δ). The selection of Δ should consider the system dynamics and how fast the system changes its behavior under the DNN controller decisions. For example, in a system like ACC from Example 1, the system reacts and changes behavior in the order of few seconds. In the experiments (RQ1), we will set Δ by applying this type of domain knowledge.

top k. In ITK and PTK, k determines how many of the neurons in a layer with the highest output should be considered activated. Setting k must consider that having a too high k would lead to having too many neurons activated, which produces a criterion not able to discriminate among neurons. In the experiments (RQ1), we will try different values, by considering the distribution of output values: more spread distributions will have higher values of k.

6. Experiment design

In the section, we report the design of the experiments we conducted to assess Tactical. The source code, benchmarks, and results are publicly available at Lyu et al. (2025).

6.1. Research questions (RQs)

We consider the following six research questions:

RQ1: *How does the selection of hyperparameters of the eight temporal neuron activation criteria affect the effectiveness of* Tactical*?* This RQ identifies the best setting for the hyperparameters of the criteria.

RQ2: *Is* Tactical *better than a random localization approach which selects the neurons randomly?* This RQ assesses whether Tactical provides any effective guidance for fault localization.

RQ3: *How does the used temporal neuron activation criterion* Cr *affect the effectiveness of* Tactical*?* This RQ identifies the criterion that provides the best guidance in fault localization.

RQ4: *How does the selection of the suspiciousness metric* SMet *affect the effectiveness of* Tactical*?* This RQ identifies the suspiciousness metric that provides the best guidance in fault localization.

RQ5: *How does the size of test suite TS affect the effectiveness of* Tactical*?* This RQ aims to investigate how the fault localization change by changing the size of the test suite.

RQ6: *Are the neurons identified by* Tactical *useful for improving the DNN controller performance?* This RQ investigates whether the performance of the DNN controller can be improved by repairing the neurons identified by Tactical.

6.2. Benchmarks

We use four widely-recognized CPSs to be used as plant $\mathcal{M}$, spanning over different domains such as automotive and chemistry, and selected from existing literature. All of these CPSs are developed in Simulink (Mathworks, 2024), the de facto standard formalism for modeling control systems in industry.

Each subject CPS can be embedded with a different DNN controller C, so creating a different AI-enabled CPS $\mathcal{M}^C$. For each subject CPS, we use an existing approach for training DNN controllers (Tran et al., 2020; Zhang et al., 2023) to obtain two DNN controllers C with different complex architectures. Thus, we have eight AI-enabled CPS $\mathcal{M}^C$, as shown in Table 2. The table reports the system complexity in terms of number of Simulink blocks *#blocks* of $\mathcal{M}$ (a Simulink block is the most fundamental element that represents a specific operation, function, or system component in a Simulink model), and the *structure* (i.e., number of neurons in each layer) and the number of weights *#weights* of C.

In the following, we introduce the CPSs and their respective specifications. By the term *correct benchmark*, we mean the combination of an AI-enabled CPS instance $\mathcal{M}^C$ (as shown in Table 2) and a specification φ, denoted as $\mathcal{M}^C_\varphi$. In total, we consider 12 correct benchmarks.

Adaptive Cruise Control (ACC). ACC has been introduced in Example 1 and used as benchmark in Manzanas Lopez et al. (2023), Tran et al. (2020, 2019), Zhang et al. (2023) and Song et al. (2022). Its specifications are:

- $\varphi_1 \equiv \square_{[0,50]}(d_{rel} \geq d_{safe} + 1.4 \cdot v_{ego} \wedge v_{ego} \leq 30)$: φ_1 requires that the relative distance d_{rel} between two cars should always be greater than the safety distance, and the speed v_{ego} of the ego car should be lower than 30;
- $\varphi_2 \equiv \square_{[0,45]}(d_{rel} < 12 + 1.4 \cdot v_{ego} \rightarrow \Diamond_{[0,5]}(d_{rel} \geq 12 + 1.4 \cdot v_{ego}))$: φ_2 requires that, during $[0, 45]$, the ego car should act within 5 s when the system is not in a steady state (i.e., $d_{rel} < 12 + 1.4 \cdot v_{ego}$).

Table 2
AI-enabled CPSs $\mathcal{M}^C$.

$\mathcal{M}^C$	ACC#1	ACC#2	AFC#1	AFC#2	WT#1	WT#2	SC#1	SC#2
#blocks of $\mathcal{M}$	49	49	153	153	11	11	55	55
Structure of C	[10 10 10 10]	[15 15 15]	[15 15 15]	[15 15 15 15]	[5 5 5]	[15 15 15]	[10 10 10 10]	[15 15 15 15]
Training algorithm	LMBP[a]	SCG[b]	LMBP	LMBP	BFG[c]	BFG	LMBP	LMBP
#weights of C	300	450	450	675	50	450	300	675

[a] LMBP: *Levenberg–Marquardt Backpropagation (LMBP) Algorithm* (Lv et al., 2018).
[b] SCG: *Scaled Conjugate Gradient Backpropagation (SCG) Algorithm* (Møller, 1993).
[c] BFG: *BFGS quasi-Newton Backpropagation (BFG) Algorithm* (Gill et al., 2019).

Abstract Fuel Control (AFC). AFC, developed by Toyota (Jin et al., 2014), has been widely used as a benchmark in literature (Menghi et al., 2023). It is a powertrain control system that outputs $\mu = \frac{|AF - AF_{ref}|}{AF_{ref}}$, i.e., the deviation of *air-to-fuel (AF)* ratio from a reference value AF_{ref}. Its specifications are:

- $\varphi_3 \equiv \square_{[0,30]}(\mu \leq \mu_{set})$: it requires that μ should always be less than μ_{set} (set as 0.2), during $[0, 30]$;
- $\varphi_4 \equiv \square_{[10,28.5]}(\mu > 0.1 \rightarrow \Diamond_{[0,1.5]}(\mu \leq 0.1))$: it checks whether the system can return to a stable state within 1.5 s when receiving a safety warning, during $[10, 28.5]$.

Water Tank (WT). WT (Song et al., 2022) is a water container that controls the water level. The system takes a reference value h_{ref} as the input signal and outputs the actual water level h_{out} at runtime. Its specification is:

- $\varphi_5 \equiv \square_{[4,5]\cup[9,10]\cup[14,15]}(|error| \leq err_{set})$: it requires that the absolute deviation between h_{ref} and h_{out} should consistently remain below a predefined threshold err_{set} during $[4, 5]$, $[9, 10]$, and $[14, 15]$. Here, err_{set} is set to 0.86.

Steam Condenser (SC). SC (Yaghoubi and Fainekos, 2019; Menghi et al., 2023) is a component of an electric power system for stabilizing the pressure in a condenser. The system input is the steam mass flow rate, while the output is the internal pressure of the condenser.

- $\varphi_6 \equiv \square_{[30,35]}(pressure \in [87, 87.5])$: it requires that the pressure should be maintained in $[87, 87.5]$ during $[30, 35]$.

6.2.1. Test suite

All the 12 correct benchmarks have been well trained and tested; they have shown to satisfy their specifications for large scale and diverse test suites. For each correct benchmark $\mathcal{M}^C_\varphi$, we select a test suite $TS_{\mathcal{M}^C_\varphi}$ comprising 100 test cases by uniformly sampling the input space of $\mathbf{u}$; all the tests in $TS_{\mathcal{M}^C_\varphi}$ satisfy the specification φ.

6.2.2. Faulty benchmarks

To assess the ability of Tactical to detect faults, we inject artificial faults into the DNN controllers C of the correct benchmarks, and we check whether an FL approach (see Section 6.3) can successfully detect the faulty neurons. Namely, we alter neuron weights' values, to mimic a non-suitable training of some weights; this is similar to what done by Sohn et al. (2023) when assessing fault localization for DNN classifiers. This type of modification is similar to the model-level mutation testing operators proposed by Ma et al. (2018c) in *DeepMutation*, a mutation analysis framework for DNN; as in their case, our modifications try to emulate real faults.

To assess the degree of faultiness of a controller, we introduce the *correctness measure* that reports the percentage of tests of the test suite that satisfy the specification, i.e.,

$$CM(\mathcal{M}^C, \varphi, TS) = \frac{|\{\mathbf{u} \in TS | \mathcal{M}^C(\mathbf{u}) \models \varphi\}|}{|TS|}$$

Each of the 12 correct benchmarks has a correctness measure of 100%.

For each correct benchmark, we first generate a set of faulty benchmarks $Fbenchs^1_{\mathcal{M}^C_\varphi}$, each containing a single fault; namely, for each weight $\mathbf{w}$ of C:

- we randomly sample a value v' in $[\mathbf{w}_{min}, \mathbf{w}_{max}]$, i.e., the range of weights values of the original controller C. We set $\mathbf{w}$ to v', so obtaining the modified DNN $\widetilde{\mathcal{M}^C}$. By sampling in the weights' values of the original controller, we aim at achieving a realistic controller that could have been obtained by a normal training;
- we execute the test suite $TS_{\mathcal{M}^C_\varphi}$ over $\widetilde{\mathcal{M}^C}$ and assess the φ's satisfaction for each test; if the correctness measure $CM(\widetilde{\mathcal{M}^C}, \varphi, TS_{\mathcal{M}^C_\varphi})$ is between 10% and 90%, $\widetilde{\mathcal{M}^C}$ is kept as faulty model (i.e., added to $Fbenchs^1_{\mathcal{M}^C_\varphi}$), otherwise it is discarded and we try another value. At most, we make 20 attempts per weight.

Then, we generate faulty benchmarks containing two faults (set $Fbenchs^2_{\mathcal{M}^C_\varphi}$), by randomly merging two faulty benchmarks from $Fbenchs^1_{\mathcal{M}^C_\varphi}$. Also in this case, we keep the faulty benchmarks $\widetilde{\mathcal{M}^C}$ whose correctness measure $CM(\widetilde{\mathcal{M}^C}, \varphi, TS_{\mathcal{M}^C_\varphi})$ is between 10% and 90%; moreover, we require that the CM of the new faulty benchmark is lower than the CM of its constituent faulty benchmarks. At most, we generate 200 faulty benchmarks in $Fbenchs^2_{\mathcal{M}^C_\varphi}$. Finally, in a similar way, we produce faulty benchmarks containing three faults (set $Fbenchs^3_{\mathcal{M}^C_\varphi}$), by randomly merging one faulty benchmark from $Fbenchs^1_{\mathcal{M}^C_\varphi}$ and one from $Fbenchs^2_{\mathcal{M}^C_\varphi}$.

We identify with $Fbenchs_{\mathcal{M}^C_\varphi} = \bigcup_{i=1}^{3} Fbenchs^i_{\mathcal{M}^C_\varphi}$ the set of all the faulty benchmarks of $\mathcal{M}^C_\varphi$.

From all the 12 correct benchmarks, we produce faulty benchmarks with one fault, with two faults, and with three faults, that are partitioned in two set of benchmarks containing 20% and 80% of the total benchmarks:

- `Bench1`: it contains 111 faulty benchmarks with one fault, 355 with two faults, 250 with three faults, for a total set of 716 faulty benchmarks.
- `Bench2`: it contains 421 faulty benchmarks with one fault, 1395 with two faults, 972 with three faults, for a total set of 2788 faulty benchmarks.

The two benchmark sets will be used to answer different RQs. As hyperparameters play an important role in the proposed neuron activation criteria, in RQ1 we first study the influence of the hyperparameters using `Bench1` and find the optimal settings to fix our approach Tactical; then, we study the performance of Tactical (with the optimal hyperparameters found in RQ1) in the remaining RQs, using `Bench2`.

6.3. Compared FL approaches

We will assess Tactical embedded with each of the eight criteria; we identify with $\text{Tactical}_{\text{Cr}}$ the approach embedded with criterion Cr. Moreover, as explained in Section 4, the different criteria are parameterized with some hyperparameters. In Section 5.1, we explain how to guide the selection of the hyperparameters. By following that heuristic

approach, we select three values (*small* (S), *medium* (M), and *large* (L)) for the three types of hyperparameters:

- *Activation threshold* h: for each benchmark, we set S, M, and L to the value at 0%, 5%, and 10% of the range $[\phi_{min}, \phi_{max}]$ of the neuron outputs; the rationale for choosing these values is that higher values would make the criterion too strict (as higher output values are produced by few neurons, and so very few neurons would be selected with a too high h). We identify the three thresholds as h_S, h_M, and h_L;
- *Time interval* Δ: for all the benchmarks, we set S, M, and L to 0.5 s, 1 s, and 1.5 s, as all the CPSs have a behavior that changes in the order of a second; therefore, the wrong behavior of the controller that leads to an unsafe state should be in this order of time. We identify the three intervals as Δ_S, Δ_M, and Δ_L;
- *Top* k: it is set by considering the size of the network; specifically, networks with more neurons in a layer should use higher values of k, to have more guarantees to detect the real faulty neuron. Therefore, we set [S, M, L] to [1, 2, 3] for benchmarks of WT#1 that is a small network; to [2, 3, 4] for benchmarks of ACC#1 and SC#1 that are networks of medium size; to [3, 4, 5] for all the other benchmarks of bigger networks. We identify the three *Top* k values as k_S, k_M, and k_L.

So, we experiment with 48 versions of TACTICAL: three versions of $\text{TACTICAL}_{\text{INA}}$, $\text{TACTICAL}_{\text{ITK}}$, $\text{TACTICAL}_{\text{MI}}$, and $\text{TACTICAL}_{\text{MD}}$, and nine versions of $\text{TACTICAL}_{\text{PNA}}$, $\text{TACTICAL}_{\text{PTK}}$, $\text{TACTICAL}_{\text{PD}}$, and $\text{TACTICAL}_{\text{ND}}$.

To assess the effectiveness of TACTICAL, we also compare it with a baseline approach RANDOM based on random sampling. For a given s (i.e., the number of selected suspicious neurons. See Line 14 in Alg. 1), RANDOM randomly samples s neurons from all the neurons (except the first hidden layer) in the DNN C and returns them as the suspicious neurons. For every faulty benchmark, we repeat RANDOM 200 times and report the average values of its evaluation metrics.

6.4. Evaluation metrics

In order to assess the effectiveness of an approach, we introduce the following metrics. Given a faulty benchmark $\widetilde{\mathcal{M}^C}$ and an approach APP, the *recall* is measured as the percentage of real faulty neurons returned in the final results SusN (see Alg. 1):

$$recall(\text{APP}, \widetilde{\mathcal{M}^C}) = \frac{\text{\# of mutated neurons of } \widetilde{\mathcal{M}^C} \text{ returned in SusN by APP}}{\text{\# of mutated neurons of } \widetilde{\mathcal{M}^C}}$$

where "mutated neuron" identifies a neuron in which a weight has been mutated. Note that recall is in $[0, 1]$.

Given a set of faulty benchmarks $Fbenchs_{\mathcal{M}^C_\varphi}$, the *detection rate* of an approach APP is defined as:

$$DR(\text{APP}, Fbenchs_{\mathcal{M}^C_\varphi}) = \frac{\sum_{\widetilde{\mathcal{M}^C} \in Fbenchs_{\mathcal{M}^C_\varphi}} recall(\text{APP}, \widetilde{\mathcal{M}^C})}{|Fbenchs_{\mathcal{M}^C_\varphi}|}$$

We compute the DR for increasing values of s (i.e., number of returned suspicious neurons), from 1 to 20% of the total number of neurons of the DNN. Then, we compute the area under the curve (AUC) of DR. The approaches will be compared in terms of their AUC. As an example, Fig. 4 shows the plot of the DR values for TACTICAL (a setting for each criterion) and RANDOM, for a given benchmark set; AUC values are the areas below each of these curves.

To answer RQ1, RQ2, and RQ3, we compare the approaches by performing statistical analysis of their AUC results. Namely, given two approaches APP1 and APP2, we collect the values of AUC obtained on each of the faulty benchmarks using each of the seven suspiciousness metrics; then, we do pairwise comparison of the AUC values using the non-parametric test Wilcoxon signed-rank test, using $\alpha = 0.05$ as significance level. If the p-value is less than α, we reject the null hypothesis that there is no significant difference, and we use Cohen's d effect size (Cohen, 1969) to assess the strength of the significance. If $d > 0$, then APP1 is better; otherwise APP2 is better. We use the categories from Kitchenham et al. (2017) to interpret the effect size: *small* if $d \in (0, 0.2)$, *medium* if $d \in [0.2, 0.8)$, and *large* if $d \geq 0.8$. Similar categories hold for $d < 0$.

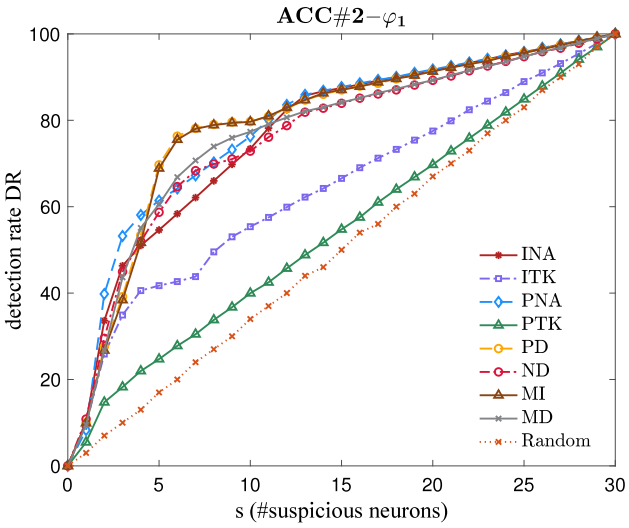


Fig. 4. Example of AUC.

In RQ1, we compare, for each criterion Cr, all versions of $\text{TACTICAL}_{\text{Cr}}$ (i.e., using all the hyperparameters' values reported in Section 6.3) among each other, using benchmark set Bench1. In this way, we select the best hyperparameter setting for each criterion Cr.

In RQ2, we compare the eight versions of TACTICAL (each initialized with the best hyperparameter setting found in RQ1) with RANDOM, using benchmark set Bench2.

In RQ3, we compare the eight versions of TACTICAL among each other, using benchmark set Bench2.

To answer RQ4, we perform a similar statistical analysis, but comparing suspiciousness metrics, still using benchmark set Bench2. Namely, given a suspiciousness metrics SMet, we collect the AUC values of all the approaches initialized with SMet and all the criteria, for all the faulty benchmarks. Then, given two suspiciousness metrics SMet_1 and SMet_2, we compare their AUC values using Wilcoxon signed-rank test and Cohen's d as described above.

To answer RQ5, we execute each version of TACTICAL (each criterion Cr is initialized with the best hyperparameters' settings found in RQ1) using test suites of different sizes (namely, 20, 40, 60, 80, and 100) over the benchmarks in Bench2. Then, we compare the results of AUC obtained using the different test suite sizes, using statistical tests as described before.

To answer RQ6, we show that the neurons identified by TACTICAL can be used to repair the DNN controller of an AI-enabled CPS using the search-based repair approach CONTREP (Lyu et al., 2024). We choose CONTREP rather than other DNN repair approaches (e.g., Sohn et al. (2023)), since it is the only one that can repair DNN controllers of AI-enabled CPSs by considering system-level specifications. CONTREP works as follows. Given an AI-enabled CPS $\mathcal{M}^C$, a set of suspicious weights SW, a system-level specification φ, and a test suite TS, CONTREP searches for alternative values of weights SW that allow to repair the controller C, i.e., the specification is satisfied by all the tests in TS. CONTREP adopts Differential Evolution (DE) (Storn and Price, 1997) as the underlying search algorithm; search variables $\overline{x}$ define possible alternative values for SW; a search individual $\overline{v}$ identifies a new DNN controller whose weights SW are set to the values $\overline{v}$ and the other weights keep the same value as in C. The fitness function, that needs to be maximized, checks the correctness measure CM of a

Table 3
AI-enabled CPSs with real faulty controllers.

$\mathcal{M}^C$	$ACC_{RF}-\varphi_1$	$ACC_{RF}-\varphi_2$	$AFC_{RF}-\varphi_3$	$AFC_{RF}-\varphi_4$	$WT_{RF}-\varphi_5$	$SC_{RF}-\varphi_6$
#blocks of $\mathcal{M}$	49	49	153	153	11	55
Structure of C	[15 15 15]	[25 25 25]	[15 15 15]	[30 30 30]	[15 15 15]	[15 15 15 15]
Training algorithm	LMBP[a]	SCG[b]	LMBP	LMBP	LMBP	CGF[c]
#weights of C	450	1250	450	1800	450	675

[a] LMBP: *Levenberg–Marquardt Backpropagation (LMBP) Algorithm* (Lv et al., 2018).
[b] SCG: *Scaled Conjugate Gradient Backpropagation (SCG) Algorithm* (Møller, 1993).
[c] CGF: *Conjugate Gradient Backpropagation with Fletcher–Reeves Updates (CGF) Algorithm* (Scales, 1985).

modified version of $\mathcal{M}^C$ (i.e., with a modified DNN controller) over the test suite TS. ContrRep returns the individual with the maximum fitness value, which can be used as repaired model. In our experiments, we use ContrRep as follows. Given the neurons SusN returned by Tactical (using the best criterion PD found in RQ3, and the best suspiciousness metric Tarantula found in RQ4) for an AI-enabled CPS $\mathcal{M}^C$, we use all the weights of SusN as search variables of ContrRep. We use a population size of 40 individuals, and run ContrRep for 50 generations. To check whether repair is effective, we check the correctness measure CM on the original AI-enabled CPS $\mathcal{M}^C$ and on the repaired one $\mathcal{M}^C_{\text{rep}}$. We perform two types of experiment in which we repair two different types of benchmarks.

In the first experiment, we select some faulty benchmarks from Bench2.[2] We select four correct benchmarks, namely the ones with the most complex DNN controller (see Table 2) using one of their specifications; namely, we select ACC#2–φ_1, AFC#2–φ_3, WT#2 –φ_5 , and SC#2–φ_6. For each correct benchmark $\mathcal{M}^C_\varphi$, we select one faulty benchmark from the set $Fbenchs^1_{\mathcal{M}^C_\varphi}$, one from $Fbenchs^2_{\mathcal{M}^C_\varphi}$, and one from $Fbenchs^3_{\mathcal{M}^C_\varphi}$. So, in this experiment, we repair 12 faulty benchmarks from Bench2.

In the second experiment, we select real faulty DNN controllers (i.e., not artificially mutated). We obtain these faulty controllers as follows. For each of the six STL specifications of the four CPSs (see Section 6.2), we train a DNN controller such that the final trained model has a correctness measure CM lower than 100%:

- we consider different DNN architectures, including the most complex one in Table 2;
- for a given architecture, we try to train it using different training algorithms (namely, LMBP (Lv et al., 2018), SCG (Møller, 1993) and CGF (Scales, 1985));
- if a training algorithm obtains a model having a CM value lower than 100%, we take the trained model as faulty controller; otherwise, we try another training algorithm for the same architecture;
- if for a given architecture we cannot train a model with $CM <$ 100% (with any training algorithm), we try another architecture.

In this way, we obtain six AI-enabled CPSs with real faults in their DNN controllers; we name these AI-enabled CPSs as: $ACC_{RF}-\varphi_1$, $ACC_{RF}-\varphi_2$, $AFC_{RF}-\varphi_3$, $AFC_{RF}-\varphi_4$, $WT_{RF}-\varphi_5$, and $SC_{RF}-\varphi_6$. The architectures and the training algorithms of these models are reported in Table 3. As before, we repair these six benchmarks.

7. Experimental results

In this section, we review the experimental results. Table 4 reports, for each version of Tactical_{Cr} (initialized with the best hyperparameters setting found in RQ1), the AUC values computed over the faulty benchmarks Bench2 of each of the 12 correct benchmarks. Results are the averages across the seven suspiciousness metrics. We observe that Tactical is always better than Random with almost all criteria. Moreover, we notice that there is no best criterion, and the best results are obtained by different criteria for different benchmarks; still, we observe that PD, INA, and MI are overall better than the other criteria.

In the following sections, we analyze the experimental results by answering the research questions in Section 6.1.

RQ1. In this RQ, we check how the setting of the hyperparameters of the criteria affects the results. Table 5 reports, for each criterion Cr, the statistical comparison among the different hyperparameters settings of Cr (see Section 6.4). Regarding the threshold h, we observe that in INA, PNA, PD, and ND, the best hyperparameters setting has always h_S; this means that it is better to have a less strict criterion that considers neurons activated more often (as the threshold for being considered activated is low).

For the time interval Δ, for MI, MD, PNA, and ND, the best value is Δ_S, while only for PTK is Δ_M and for PD is Δ_L; still, the second best of PTK and PD is Δ_S. This could mean that, in general, large intervals lead to too strict criteria that are seldom satisfied, and so do not consider too many neurons as activated.

For k, we observe that the best in ITK is k_S and in PTK is k_M.

Answer to RQ1: Regarding the threshold h and the time interval Δ, it is usually better to use a small value. Instead, for parameter k, it is better to use a small or medium value.

RQ2. In this RQ, we assess whether Tactical can effectively localize faulty neurons. To do this, we compare the eight versions of Tactical_{Cr} (using, for each criterion Cr, the best setting of the hyperparameters as found in RQ1) with Random, using the statistical tests as explained in Section 6.4. Results show that all the versions of Tactical are better than Random with *large* strength for the effect size. This can be also seen in Table 4 where Random has the lowest AUC value in almost all the cases, except for two in which it is has the second and third lowest value. This shows that Tactical is indeed able to effectively localize faulty neurons.

Answer to RQ2: Any version of Tactical is significantly better, with large effect size, than a random approach which selects neurons randomly.

RQ3. Temporal neuron activation criteria are used in Tactical as proxy measures to estimate the involvement of a neuron in the temporal DNN behavior. Being heuristic measures, it is necessary to assess their effectiveness. Therefore, in this RQ, we want to assess which criterion Cr provides the best results. As explained in Section 6.4, for each criterion Cr, we select the best setting of its hyperparameters as discovered in RQ1. Results are reported in Table 6. First of all, we observe that there are differences among the different criteria, which means that they are not equivalent and that they capture different behaviors of a DNN controller.

Then, we notice that PD is the best criterion, better than all the others with strength *medium*; this means that an increment of the neuron output for a given period of time is a good proxy for the relevance of a neuron; when this pattern occurs, it means that the neuron is involved in the control decision constantly for a period of time. INA is the second best criterion. Indeed INA is similar to PD, as

[2] Note that we had to select some benchmarks, as repairing all of them would have made the experiments too expensive.

Table 4
Comparison of different approaches with their best hyperparameters in terms of average AUC with $s \leq 20\%$ (average across all suspiciousness metrics SMet); computed over Bench2. The higher the value, the better (the best approach is highlighted in gray; the second best approach is highlighted in light gray).

	INA	ITK	PNA	PTK	PD	ND	MI	MD	RANDOM
	h_S	k_S	$\langle h_S, \Delta_S \rangle$	$\langle \Delta_M, k_M \rangle$	$\langle h_S, \Delta_L \rangle$	$\langle h_S, \Delta_S \rangle$	Δ_S	Δ_S	
ACC#1−φ_1	0.1857	0.1685	0.1920	0.1672	0.1985	0.1650	0.1976	0.1664	0.1001
ACC#1−φ_2	0.0850	0.1046	0.0842	0.1088	0.1114	0.1037	0.1113	0.0914	0.1003
ACC#2−φ_1	0.2078	0.1818	0.1851	0.1796	0.2434	0.1864	0.2367	0.1924	0.0998
ACC#2−φ_2	0.2570	0.1903	0.1888	0.1823	0.2140	0.2287	0.2126	0.2156	0.1002
AFC#1−φ_3	0.2682	0.1939	0.2465	0.1464	0.2952	0.1800	0.1983	0.1786	0.0994
AFC#1−φ_4	0.2235	0.1689	0.2314	0.1630	0.2346	0.2119	0.1807	0.1709	0.1010
AFC#2−φ_3	0.2306	0.2350	0.2123	0.1293	0.2626	0.1185	0.2029	0.1143	0.0997
AFC#2−φ_4	0.2106	0.1836	0.2270	0.1344	0.1939	0.2053	0.1956	0.1287	0.1016
WT#1−φ_5	0.1202	0.1169	0.1227	0.1280	0.1727	0.1179	0.1520	0.1295	0.1004
WT#2 −φ_5	0.2320	0.1811	0.2240	0.1819	0.2309	0.2167	0.2146	0.2150	0.0999
SC#1−φ_6	0.2787	0.2706	0.2676	0.3348	0.2800	0.2787	0.2820	0.3108	0.1013
SC#2−φ_6	0.0997	0.1724	0.1011	0.1694	0.0997	0.0997	0.1318	0.0995	0.1009

it requires that a neuron is activated at a given timestamp; so, to some extent, PD subsumes INA.

MI is the third best criterion, only worse than INA and PD. MI is similarly to PD, as it requires that the neuron output increases; differently from PD, it requires that the increment is monotonic. So, its good performance confirms that the increment of the neuron output is indeed an indication of the relevance of the neuron; however, requiring the increment to be monotonic could be too strict.

The good results of PD, INA, and MI are confirmed by the results in Table 4, where these three criteria are those with more benchmarks with the highest and second highest AUC values.

MD and ND, instead, do not provide a good performance. This means that the decrease of the neuron activation is a not a good indicator of the periods in which a neuron is relevant; as a matter of fact, the timestamps in which MD and ND are covered are those in which the neuron becomes to be less influential.

Finally, we observe that also PTK is not a good criterion. This is because, as it considers as activated the top k neurons in a layer, it cannot properly discriminate among them.

Answer to RQ3: PD is the best criterion, followed by INA, and MI. MD, ND, and PTK are the worst criteria.

RQ4. In this RQ, we want to assess which is the best suspiciousness metric SMet. Table 7 reports the statistical comparison among all the seven considered suspiciousness metrics, as explained in Section 6.4. We observe that Tarantula is the best metric, followed by Jaccard. Op2, instead, is the worst metric. The definition of Op2 gives a lot of importance to af, i.e., the number of failing tests in which the neuron is activated. Tarantula and Jaccard, instead, in addition to af, also give some importance to ap, i.e., the number of passing tests in which the neuron is activated. So, it seems that, in DNNs, it is important to consider the influence of a neuron on passing tests.

Answer to RQ4: Tarantula and Jaccard are the best suspiciousness metrics, while Op2 is the worst.

RQ5. In this RQ, we want to assess the influence of the test suite size on the effectiveness of TACTICAL. To do this, we check the effectiveness of TACTICAL using test suites containing 20, 40, 60, 80, and 100 tests. We initialize TACTICAL with the best criterion identified in RQ3 (i.e., PD) and the best suspiciousness metric identified in RQ4 (i.e., Tarantula). Table 8 reports the statistical comparison between the execution with the different test suite sizes. We observe that larger test suites allow to obtain significantly better results that smaller test suites. This is expected, as more tests allow to have a better assessment of the contribution of different neurons to the failure.

Answer to RQ5: Larger test suites allow to obtain better fault localization results.

RQ6. In this RQ, we investigate whether the neurons detected by TACTICAL can be used to improve the AI-enabled CPS. As explained in Section 6.4, we use the search-based repair tool CONTREP (Lyu et al., 2024) to repair all the weights of the suspicious neurons identified by TACTICAL. We repair two sets of benchmarks: a subset of the artificial faulty benchmarks in Bench2, and a set of real faulty benchmarks obtained by a non-optimal training. The results are reported in Table 9. The table reports, for each model under repair, the correctness measure CM before and after repair. We observe that, in all cases, we manage to raise CM of an AI-enabled CPS with a non-optimal controller (CM before repair ranges from 12% to 85% in Table 9a, and from 53% to 89% in Table 9b) to 100%. The results are consistent for the artificial faulty benchmarks and the real faulty benchmarks. The results demonstrate that the neurons identified by TACTICAL are indeed helpful to improve the performance of the AI-enabled CPSs.

Answer to RQ6: The neurons identified by TACTICAL can be effectively used as target of a search-based repair technique of AI-enabled CPSs.

8. Threats to validity

We here discuss threats that may affect the validity of the approach (Wohlin et al., 2012).

Construct validity. The metrics used in the assessment of the approach could be not suitable. Since we want to assess the effectiveness of TACTICAL, we mutate some weights of the DNN controller C and check whether the corresponding neurons can be localized. Moreover, TACTICAL depends on parameterized criteria; different hyperparameters' settings can affect the effectiveness of the approach. To mitigate this threat, in RQ2, we assessed the performance under different settings of the hyperparameters.

Conclusion validity. The random nature of RANDOM can affect the final result. Following Arcuri and Briand (2011), we executed RANDOM 200 times for each benchmark. We have also used suitable statistical tests to compare the different approaches, by considering both significant difference and effect size.

Internal validity. The obtained results could be obtained by chance, due to a faulty implementation. To mitigate this threat, we have carefully tested the implementation of TACTICAL.

External validity. The generalizability of TACTICAL is an external validity threat. To mitigate it, we consider 12 correct benchmarks that are widely used in the community of AI-enabled CPSs, and we experiment TACTICAL with 3504 faulty benchmarks obtained from these correct benchmarks.

Table 5
RQ1 — Statistical comparison between the hyperparameters settings of the eight temporal neuron activation criteria, using benchmark set `Bench1`.

(a) INA

	h_S	h_M	h_L
h_S	–	✓✓	✓✓
h_M	✗✗	–	✓
h_L	✗✗	✗	–

(b) ITK

	k_S	k_M	k_L
k_S	–	✓✓	✓
k_M	✗✗	–	✗
k_L	✗	✓	–

(c) MI

	Δ_S	Δ_M	Δ_L
Δ_S	–	✓✓	✓✓
Δ_M	✗✗	–	✓
Δ_L	✗✗	✗	–

(d) MD

	Δ_S	Δ_M	Δ_L
Δ_S	–	✓✓	✓✓
Δ_M	✗✗	–	✗
Δ_L	✗✗	✓	–

(e) PNA

	$\langle h_S, \Delta_S\rangle$	$\langle h_S, \Delta_M\rangle$	$\langle h_S, \Delta_L\rangle$	$\langle h_M, \Delta_S\rangle$	$\langle h_M, \Delta_M\rangle$	$\langle h_M, \Delta_L\rangle$	$\langle h_L, \Delta_S\rangle$	$\langle h_L, \Delta_M\rangle$	$\langle h_L, \Delta_L\rangle$
$\langle h_S, \Delta_S\rangle$	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_S, \Delta_M\rangle$	✗✗	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_S, \Delta_L\rangle$	✗✗	✗✗	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_M, \Delta_S\rangle$	✗✗	✗✗	✗✗	–	✓✓	✓✓	✓	✓✓	✓✓
$\langle h_M, \Delta_M\rangle$	✗✗	✗✗	✗✗	✗✗	–	✗✗	✗✗	✗✗	✓
$\langle h_M, \Delta_L\rangle$	✗✗	✗✗	✗✗	✗✗	✓✓	–	✗	✗✗	✓
$\langle h_L, \Delta_S\rangle$	✗✗	✗✗	✗✗	✗	✓✓	✓	–	✗	✓✓
$\langle h_L, \Delta_M\rangle$	✗✗	✗✗	✗✗	✗	✓✓	✓✓	✓	–	✓✓
$\langle h_L, \Delta_L\rangle$	✗✗	✗✗	✗✗	✗✗	✗	✗	✗✗	✗✗	–

(f) PTK

	$\langle \Delta_S, k_S\rangle$	$\langle \Delta_S, k_M\rangle$	$\langle \Delta_S, k_L\rangle$	$\langle \Delta_M, k_S\rangle$	$\langle \Delta_M, k_M\rangle$	$\langle \Delta_M, k_L\rangle$	$\langle \Delta_L, k_S\rangle$	$\langle \Delta_L, k_M\rangle$	$\langle \Delta_L, k_L\rangle$
$\langle \Delta_S, k_S\rangle$	–	✗✗	✗	✓✓	✗✗	✗	✓✓	✗✗	✗
$\langle \Delta_S, k_M\rangle$	✓✓	–	✓✓	✓✓	✗	✓✓	✓✓	✓	✓✓
$\langle \Delta_S, k_L\rangle$	✓	✗✗	–	✓	✗✗	✓	✓✓	✗✗	✓✓
$\langle \Delta_M, k_S\rangle$	✗✗	✗✗	✗	–	✗✗	✗	✓✓	✗✗	✗
$\langle \Delta_M, k_M\rangle$	✓✓	✓	✓✓	✓✓	–	✓✓	✓✓	✓✓	✓✓
$\langle \Delta_M, k_L\rangle$	✓	✗✗	✗	✓	✗✗	–	✓✓	✗✗	✓
$\langle \Delta_L, k_S\rangle$	✗✗	✗✗	✗✗	✗✗	✗✗	✗✗	–	✗✗	✗✗
$\langle \Delta_L, k_M\rangle$	✓✓	✗	✓✓	✓✓	✗✗	✓✓	✓✓	–	✓✓
$\langle \Delta_L, k_L\rangle$	✓	✗✗	✗✗	✓	✗✗	✗	✓✓	✗✗	–

(g) PD

	$\langle h_S, \Delta_S\rangle$	$\langle h_S, \Delta_M\rangle$	$\langle h_S, \Delta_L\rangle$	$\langle h_M, \Delta_S\rangle$	$\langle h_M, \Delta_M\rangle$	$\langle h_M, \Delta_L\rangle$	$\langle h_L, \Delta_S\rangle$	$\langle h_L, \Delta_M\rangle$	$\langle h_L, \Delta_L\rangle$
$\langle h_S, \Delta_S\rangle$	–	✓	✗	✓✓✓	✓✓	✓✓	✓✓✓	✓✓	✓✓
$\langle h_S, \Delta_M\rangle$	✗	–	✗✗	✓✓✓	✓✓	✓✓	✓✓✓	✓✓	✓✓
$\langle h_S, \Delta_L\rangle$	✓	✓✓	–	✓✓✓	✓✓	✓✓	✓✓✓	✓✓	✓✓
$\langle h_M, \Delta_S\rangle$	✗✗✗	✗✗✗	✗✗✗	–	✗✗	✗✗	✓✓	✓	✗
$\langle h_M, \Delta_M\rangle$	✗✗	✗✗	✗✗	✓✓	–	✗✗	✓✓	✓✓	✓✓
$\langle h_M, \Delta_L\rangle$	✗✗	✗✗	✗✗	✓✓	✓✓	–	✓✓	✓✓	✓✓
$\langle h_L, \Delta_S\rangle$	✗✗✗	✗✗✗	✗✗✗	✗✗	✗✗	✗✗	–	✗✗	✗✗
$\langle h_L, \Delta_M\rangle$	✗✗	✗✗	✗✗	✗	✗✗	✗✗	✓✓	–	✗✗
$\langle h_L, \Delta_L\rangle$	✗✗	✗✗	✗✗	✓	✗✗	✗✗	✓✓	✓✓	–

(h) ND

	$\langle h_S, \Delta_S\rangle$	$\langle h_S, \Delta_M\rangle$	$\langle h_S, \Delta_L\rangle$	$\langle h_M, \Delta_S\rangle$	$\langle h_M, \Delta_M\rangle$	$\langle h_M, \Delta_L\rangle$	$\langle h_L, \Delta_S\rangle$	$\langle h_L, \Delta_M\rangle$	$\langle h_L, \Delta_L\rangle$
$\langle h_S, \Delta_S\rangle$	–	✓✓	✓✓	✓✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_S, \Delta_M\rangle$	✗✗	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_S, \Delta_L\rangle$	✗✗	✗✗	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
$\langle h_M, \Delta_S\rangle$	✗✗✗	✗✗	✗✗	–	✓	✗✗	✗	✗	✗
$\langle h_M, \Delta_M\rangle$	✗✗	✗✗	✗✗	✗	–	✗✗	✗	✗	✗✗
$\langle h_M, \Delta_L\rangle$	✗✗	✗✗	✗✗	✓✓	✓✓	–	✓	✓	✓
$\langle h_L, \Delta_S\rangle$	✗✗	✗✗	✗✗	✓	✓	✗	–	✓	✗
$\langle h_L, \Delta_M\rangle$	✗✗	✗✗	✗✗	✓	✓	✗	✗	–	✗
$\langle h_L, \Delta_L\rangle$	✗✗	✗✗	✗✗	✓	✓✓	✗	✓	✓	–

(Legend. ≡: no difference between the two approaches. ✓, ✓✓, ✓✓✓: the approach on the row is better than the approach on the column with strength *small*, *medium*, *large*. ✗, ✗✗, ✗✗✗: the approach on the row is worse than the approach on the column with strength *small*, *medium*, *large*. Best and second best settings are in green and light green resp.)

Table 6
RQ3 — Statistical comparison between the eight temporal neuron activation criteria with their best hyperparameters, using benchmark set Bench2.

	INA	ITK	PNA	PTK	PD	ND	MI	MD
INA	–	✓✓	✓✓	✓✓	✗✗	✓✓	✓	✓✓
ITK	✗✗	–	✗✗	✓✓	✗✗	✓	✗✗	✓✓
PNA	✗✗	✓✓	–	✓✓	✗✗	✓✓	✗	✓✓
PTK	✗✗	✗✗	✗✗	–	✗✗	✗	✗✗	✓
PD	✓✓	✓✓	✓✓	✓✓	–	✓✓	✓✓	✓✓
ND	✗✗	✗	✗✗	✓	✗✗	–	✗✗	✓✓
MI	✗	✓✓	✓	✓✓	✗✗	✓✓	–	✓✓
MD	✗✗	✗✗	✗✗	✗	✗✗	✗✗	✗✗	–

(Legend as in Table 5).

Table 7
RQ4 — Statistical comparison between each pair of suspiciousness metrics, using benchmark set Bench2.

	Tarantula	Ochiai	D^2	D^3	Jaccard	Kulczynski2	Op2
Tarantula	–	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
Ochiai	✗✗	–	✓✓	✓✓	✗✗	✓✓	✓✓
D^2	✗✗	✗✗	–	✓✓	✗✗	✓✓	✓✓
D^3	✗✗	✗✗	✗✗	–	✗✗	✗	✓✓
Jaccard	✗✗	✓✓	✓✓	✓✓	–	✓✓	✓✓
Kulczynski2	✗✗	✗✗	✗✗	✓	✗✗	–	✓✓
Op2	✗✗	✗✗	✗✗	✗✗	✗✗	✗✗	–

(Legend as in Table 5).

Table 8
RQ5 — Statistical comparison between different versions of TACTICAL (the best setting for each criterion) that use test suites TS of different sizes ($|TS| \in \{20, 40, 60, 80, 100\}$). TACTICAL uses the best suspiciousness metric Tarantula identified in RQ4 (see Table 7). Experiments are conducted over Bench2.

	20	40	60	80	100
20	–	✗	✗	✗	✗
40	✓	–	✗	✗	✗
60	✓	✓	–	✗	✗
80	✓	✓	✓	–	✗
100	✓	✓	✓	✓	–

(Legend as in Table 5).

9. Related work

DNN fault localization. Fault localization for standalone DNNs has been extensively investigated. Eniser et al. (2019) introduce *DeepFault* that employs SBFL to identify faulty neurons based on their activation. Duran et al. (2021) illustrate that simultaneously performing fault localization for various misclassifications may lead to suboptimal results, attributed to the masking effect across different misclassifications. Ma et al. (2018b) propose *MODE*, which adopts state-differential analysis between misclassification and correct classification to identify faulty neurons. Sohn et al. (2023) present *Arachne* for DNN repair, where they address the issue of fault localization targeting neuron weights. Wardat et al. (2021) present *DeepLocalize* for pinpointing faults in DL programs, e.g., identifying issues with activation function configurations. Wardat et al. (2022) also propose *DeepDiagnosis* that extends *DeepLocalize* to detect faults during training and suggest possible repair actions. Cao et al. (2022) introduce *DeepFD* that leverages training information for fault localization. Ghanbari et al. (2023) propose *deepmufl*, which uses mutation-based fault localization to identify faults in DNN models due to wrong DL program configurations. Schoop et al. (2021), instead, propose *UMLAUT* that aims to localize the problems in deep learning programs, which significantly diverges from our approach.

All previous approaches are not applicable to fault localization for AI-enabled CPSs that presents specific challenges discussed in Sections 1 and 3.1, i.e., the DNN cannot be analyzed in isolation, there is no ground truth for DNN controllers, and the control is due to a sequence of DNN controller inferences.

DNN repair. Fault localization approaches as those described above allow to find critical components of the DNN that must be fixed. DNN repair is an emerging approach that can be used to fix the DNN. Usman et al. (2021) propose a repair technique that encodes the objective of keeping positive examples and fixing adversarial examples as logical constraints and solve them by off-the-shelf solvers. Sohn et al. (2023) propose *Arachne* that repairs DNNs by searching for corrections of model parameters based on fault localization. Li Calsi et al. (2023a) propose an adaptive repair framework that can handle the change of suspicious model parameters during the repair process of DNNs. Li Calsi et al. (2023b) target the problem in which there are multiple misclassifications and propose a technique that prioritizes the critical ones. Tokui et al. (2022) propose *NeuRecover*, a DNN repair method that leverages training history to identify the critical weights, as these weights play a key role in regression; then, they optimize them using a search-based repair approach similar to that of *Arachne*. Zhang and Chan (2019) propose *Apricot*, which synthesizes a new DNN model by retraining multiple reduced models on smaller training datasets and adapting their weights iteratively. Sotoudeh and Thakur (2021) aim at obtaining a repaired DNN that satisfies a given specification; to do this, they formulate the repair problem as a linear programming problem by the introduction of a *Decoupled* DNN architecture. Sun et al. (2022) utilize causality analysis to pinpoint neurons responsible for erroneous behavior and adjust their weights through optimization to correct the misbehavior while preserving model accuracy. Similarly, Wu et al. (2021) propose *GenMuNN*, a mutation-based repair approach for DNNs that ranks weights based on their influence on the final prediction output and applies genetic algorithms to iteratively select mutants of DNN models with improved accuracy. Henriksen et al. (2022) modify localized weights guided by gradient analysis to repair NN misclassifications without access to the training dataset.

A different type of approaches aim at identifying and fixing problems in the DNN architecture and training setting. Kim et al. (2023) survey and evaluate existing repair techniques for DNN model architecture faults. Zhang et al. (2021) propose *AutoTrainer*, an automatic tool to identify and repair deep learning training problems; it can monitor the training process, identify problems from the collected training details, and apply the built-in state-of-the-art repair solutions to improve the DNN models.

The previous approaches can be used to repair DNN for which there is an explicit ground truth (like DNN classifiers), but cannot be applied to repair the DNN controllers considered in this work for which there is no explicit ground truth. Lyu et al. (2024) propose an approach that is able to repair DNN controllers relying on system level specifications to provide the assessment of the decisions of the DNN controller. We use this repair approach in our experiments to show that the neurons identified by TACTICAL can be effectively used to repair the DNN controller.

Fault localization of classic CPS. For classic CPS fault localization, multiple lines of research have been developed. First, Liu et al. (2016a) introduce an iterative approach to fault localization by using decision trees. Liu et al. (2016b) rely on statistical debugging and dynamic model slicing for fault localization. To further enhance accuracy, Liu et al. (2017) introduce a search-based approach that can generate compact and diverse test suites. Second, Bartocci et al. (2018) utilize trace diagnostics for fault localization of Simulink models. Later, they develop *CPSDebug* (Bartocci et al., 2019) that can debug Simulink failures. Moreover, they introduce a search-based approach (Bartocci et al., 2022) that can generate passing test cases closely mirroring failing tests, such that the precise fault location can be extracted from the system.

These studies target classic CPS models that involve classic CPS blocks only, and these approaches are not designed to handle the specific architectures of DNN controllers in AI-enabled CPS. In contrast, our approach is specifically devised for detecting faulty neurons in DNN controllers, considering the temporal features of DNN controller inferences.

Table 9
RQ6 — Results of repair of AI-enabled CPSs using the neurons identified by Tactical.

(a) Artificial faulty benchmarks (selected from Bench2)

	ACC#2−φ_1			AFC#2−φ_3			WT#2 −φ_5			SC#2−φ_6		
$Fbenchs^{X\in\{1,2,3\}}_{\mathcal{M}^c_\varphi}$	1	2	3	1	2	3	1	2	3	1	2	3
CM before repair	85	29	82	34	83	41	79	61	30	82	12	17
CM after repair	100	100	100	100	100	100	100	100	100	100	100	100

(b) Real faulty benchmarks

	$ACC_{RF}-\varphi_1$	$ACC_{RF}-\varphi_2$	$AFC_{RF}-\varphi_3$	$AFC_{RF}-\varphi_4$	$WT_{RF}-\varphi_5$	$SC_{RF}-\varphi_6$
CM before repair	89	55	60	53	66	85
CM after repair	100	100	100	100	100	100

10. Conclusion

We proposed Tactical that can localize faulty neurons in DNN controllers of AI-enabled CPSs, by exploiting a series of neuron activation criteria that consider temporal aspects of DNN controller inferences. Based on extensive executions of test cases, we construct a spectrum for each neuron of a DNN controller, by which we can compute a suspiciousness score for each neuron and thereby select those suspicious neurons.

CRediT authorship contribution statement

Deyun Lyu: Writing – original draft, Software, Methodology. **Yi Li:** Software. **Zhenya Zhang:** Writing – review & editing, Writing – original draft, Methodology. **Paolo Arcaini:** Writing – review & editing, Writing – original draft, Methodology. **Xiao-Yi Zhang:** Writing – review & editing, Writing – original draft, Methodology. **Fuyuki Ishikawa:** Writing – review & editing, Supervision. **Jianjun Zhao:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

P. Arcaini and F. Ishikawa are supported by Engineerable AI Techniques for Practical Applications of High-Quality Machine Learning-based Systems Project (Grant Number JPMJMI20B8), JST-Mirai. X. Zhang is supported by Youth Fund of the National Natural Science Foundation of China (No. 62302035). Zhenya Zhang and Jianjun Zhao are supported by JSPS KAKENHI Grant No. JP23H03372. Zhenya Zhang is also supported by JSPS KAKENHI No. JP23K16865, JP25K21179; and JST BOOST Grant No. JPMJBY24D7.

Data availability

Data will be made available on request.

References

Abreu, R., Zoeteweij, P., Golsteijn, R., van Gemund, A.J., 2009. A practical evaluation of spectrum-based fault localization. J. Syst. Softw. 82 (11), 1780–1792. http://dx.doi.org/10.1016/j.jss.2009.06.035.

Abreu, R., Zoeteweij, P., Van Gemund, A.J., 2006. An evaluation of similarity coefficients for software fault localization. In: 2006 12th Pacific Rim International Symposium on Dependable Computing. PRDC'06, pp. 39–46. http://dx.doi.org/10.1109/PRDC.2006.18.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering. ICSE '11, ACM, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/1985793.1985795.

Bartocci, E., Ferrère, T., Manjunath, N., 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). HSCC '18, Association for Computing Machinery, New York, NY, USA, pp. 197–206. http://dx.doi.org/10.1145/3178126.3178131.

Bartocci, E., Manjunath, N., Mariani, L., Mateis, C., Ničković, D., 2019. Automatic failure explanation in CPS models. In: Software Engineering and Formal Methods: 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings. Springer-Verlag, Berlin, Heidelberg, pp. 69–86. http://dx.doi.org/10.1007/978-3-030-30446-1_4.

Bartocci, E., Mariani, L., Ničković, D., Yadav, D., 2022. Search-based testing for accurate fault localization in CPS. In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering. ISSRE, pp. 145–156. http://dx.doi.org/10.1109/ISSRE55969.2022.00024.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K., 2016. End to end learning for self-driving cars. CoRR abs/1604.07316, arXiv:1604.07316.

Cao, J., Li, M., Chen, X., Wen, M., Tian, Y., Wu, B., Cheung, S.-C., 2022. DeepFD: Automated fault diagnosis and localization for deep learning programs. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 573–585. http://dx.doi.org/10.1145/3510003.3510099.

Cohen, J., 1969. Statistical Power Analysis for the Behavioral Sciences. Academic Press, New York.

Donzé, A., Maler, O., 2010. Robust satisfaction of temporal logic over real-valued signals. In: Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems. FORMATS '10, Springer-Verlag, Berlin, Heidelberg, pp. 92–106.

Duran, M., Zhang, X., Arcaini, P., Ishikawa, F., 2021. What to blame? On the granularity of fault localization for deep neural networks. In: 2021 IEEE 32nd International Symposium on Software Reliability Engineering. ISSRE, pp. 264–275. http://dx.doi.org/10.1109/ISSRE52982.2021.00037.

Eniser, H.F., Gerasimou, S., Sen, A., 2019. DeepFault: Fault localization for deep neural networks. In: Hähnle, R., van der Aalst, W. (Eds.), Fundamental Approaches To Software Engineering. Springer International Publishing, Cham, pp. 171–191.

Fainekos, G.E., Pappas, G.J., 2009. Robustness of temporal logic specifications for continuous-time signals. Theoret. Comput. Sci. 410 (42), 4262–4291. http://dx.doi.org/10.1016/j.tcs.2009.06.021.

Ghanbari, A., Thomas, D.-G., Arshad, M.A., Rajan, H., 2023. Mutation-based fault localization of deep neural networks. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 1301–1313. http://dx.doi.org/10.1109/ASE56229.2023.00171.

Gill, P.E., Murray, W., Wright, M.H., 2019. Practical Optimization. SIAM.

Gu, S., Holly, E., Lillicrap, T., Levine, S., 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation. ICRA, IEEE Press, pp. 3389–3396. http://dx.doi.org/10.1109/ICRA.2017.7989385.

Henriksen, P., Leofante, F., Lomuscio, A., 2022. Repairing misclassifications in neural networks using limited data. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. SAC '22, Association for Computing Machinery, New York, NY, USA, pp. 1031–1038. http://dx.doi.org/10.1145/3477314.3507059.

Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q., 2019. ReachNN: Reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. 18 (5s), http://dx.doi.org/10.1145/3358228.

Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K., 2014. Powertrain control verification benchmark. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control. HSCC '14, Association for Computing Machinery, New York, NY, USA, pp. 253–262. http://dx.doi.org/10.1145/2562059.2562140.

Johnson, T.T., Manzanas Lopez, D., Musau, P., Tran, H.-D., Botoeva, E., Leofante, F., Maleki, A., Sidrane, C., Fan, J., Huang, C., 2020. ARCH-COMP20 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (Eds.), ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20). 74, EasyChair, pp. 107–139. http://dx.doi.org/10.29007/9xgv.

Jones, J.A., Harrold, M.J., Stasko, J., 2002. Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering. ICSE '02, Association for Computing Machinery, New York, NY, USA, pp. 467–477. http://dx.doi.org/10.1145/581339.581397.

Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J., 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunˇ cak, V. (Eds.), Computer Aided Verification. Springer International Publishing, Cham, pp. 97–117.

Kim, J., Humbatova, N., Jahangirova, G., Tonella, P., Yoo, S., 2023. Repairing DNN architecture: Are we there yet? In: 2023 IEEE Conference on Software Testing, Verification and Validation. ICST, pp. 234–245. http://dx.doi.org/10.1109/ICST57152.2023.00030.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A., 2017. Robust statistical methods for empirical software engineering. Empir. Softw. Eng. 22 (2), 579–630. http://dx.doi.org/10.1007/s10664-016-9437-5.

Li Calsi, D., Duran, M., Laurent, T., Zhang, X., Arcaini, P., Ishikawa, F., 2023a. Adaptive search-based repair of deep neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '23, Association for Computing Machinery, New York, NY, USA, pp. 1527–1536. http://dx.doi.org/10.1145/3583131.3590477.

Li Calsi, D., Duran, M., Zhang, X., Arcaini, P., Ishikawa, F., 2023b. Distributed repair of deep neural networks. In: 2023 IEEE Conference on Software Testing, Verification and Validation. ICST, pp. 83–94. http://dx.doi.org/10.1109/ICST57152.2023.00017.

Liu, B., Lucia, Nejati, S., Briand, L.C., 2017. Improving fault localization for Simulink models using search-based testing and prediction models. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. SANER, pp. 359–370. http://dx.doi.org/10.1109/SANER.2017.7884636.

Liu, B., Lucia, Nejati, S., Briand, L., Bruckmann, T., 2016a. Localizing multiple faults in Simulink models. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. SANER, vol. 1, pp. 146–156. http://dx.doi.org/10.1109/SANER.2016.38.

Liu, B., Lucia, Nejati, S., Briand, L.C., Bruckmann, T., 2016b. Simulink fault localization: an iterative statistical debugging approach. Softw. Test. Verif. Reliab. 26 (6), 431–459. http://dx.doi.org/10.1002/stvr.1605.

Lv, C., Xing, Y., Zhang, J., Na, X., Li, Y., Liu, T., Cao, D., Wang, F.-Y., 2018. Levenberg–marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system. IEEE Trans. Ind. Inform. 14 (8), 3436–3446. http://dx.doi.org/10.1109/TII.2017.2777460.

Lyu, D., Li, Y., Zhang, Z., Arcaini, P., Zhang, X., Ishikawa, F., Zhao, J., 2025. Code of the paper "Fault Localization of AI-Enabled Cyber-Physical Systems by Exploiting Temporal Neuron Activation". URL: https://github.com/jst-qaml/TACTICAL.

Lyu, D., Zhang, Z., Arcaini, P., Ishikawa, F., Laurent, T., Zhao, J., 2024. Search-based repair of DNN controllers of AI-Enabled Cyber-Physical Systems guided by system-level specifications. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '24, Association for Computing Machinery, New York, NY, USA, pp. 1435–1444. http://dx.doi.org/10.1145/3638529.3654078.

Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y., 2018a. DeepGauge: Multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. In: ASE 2018, Association for Computing Machinery, New York, NY, USA, pp. 120–131. http://dx.doi.org/10.1145/3238147.3238202.

Ma, S., Liu, Y., Lee, W.-C., Zhang, X., Grama, A., 2018b. 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. In: ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA, pp. 175–186. http://dx.doi.org/10.1145/3236024.3236082.

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., Wang, Y., 2018c. DeepMutation: Mutation testing of deep learning systems. In: 2018 IEEE 29th International Symposium on Software Reliability Engineering. ISSRE, pp. 100–111. http://dx.doi.org/10.1109/ISSRE.2018.00021.

Manzanas Lopez, D., Althoff, M., Forets, M., Johnson, T.T., Ladner, T., Schilling, C., 2023. ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (Eds.), Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). vol. 96, EasyChair, pp. 89–125. http://dx.doi.org/10.29007/x38n.

Mathworks, 2024. Simulink. https://www.mathworks.com/products/simulink.html.

Menghi, C., Arcaini, P., Baptista, W., Ernst, G., Fainekos, G., Formica, F., Gon, S., Khandait, T., Kundu, A., Pedrielli, G., Peltomäki, J., Porres, I., Ray, R., Waga, M., Zhang, Z., 2023. ARCH-COMP23 category report: Falsification. In: Frehse, G., Althoff, M. (Eds.), Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23). vol. 96, EasyChair, pp. 151–169. http://dx.doi.org/10.29007/6nqs.

Møller, M.F., 1993. A scaled conjugate gradient algorithm for fast supervised learning. Neural Netw. 6 (4), 525–533. http://dx.doi.org/10.1016/S0893-6080(05)80056-5.

Naish, L., Lee, H.J., Ramamohanarao, K., 2011. A model for spectra-based software diagnosis. ACM Trans. Softw. Eng. Methodol. 20 (3), http://dx.doi.org/10.1145/2000791.2000795.

Pearson, S., Campos, J., Just, R., Fraser, G., Abreu, R., Ernst, M.D., Pang, D., Keller, B., 2017. Evaluating and improving fault localization. In: Proceedings of the 39th International Conference on Software Engineering. ICSE '17, IEEE Press, pp. 609–620. http://dx.doi.org/10.1109/ICSE.2017.62.

Pei, K., Cao, Y., Yang, J., Jana, S., 2019. DeepXplore: Automated whitebox testing of deep learning systems. Commun. ACM 62 (11), 137–145. http://dx.doi.org/10.1145/3361566.

Ren, X., Chen, J., Juefei-Xu, F., Xue, W., Guo, Q., Ma, L., Zhao, J., Chen, S., 2022. DARTSRepair: Core-failure-set guided DARTS for network robustness to common corruptions. Pattern Recognit. 131 (C), http://dx.doi.org/10.1016/j.patcog.2022.108864.

Scales, L.E., 1985. Introduction to Non-Linear Optimization. Springer-Verlag, Berlin, Heidelberg.

Schoop, E., Huang, F., Hartmann, B., 2021. UMLAUT: Debugging deep learning programs using program structure and model behavior. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. CHI '21, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3411764.3445538.

Sohn, J., Kang, S., Yoo, S., 2023. Arachne: Search-based repair of deep neural networks. ACM Trans. Softw. Eng. Methodol. 32 (4), http://dx.doi.org/10.1145/3563210.

Song, J., Lyu, D., Zhang, Z., Wang, Z., Zhang, T., 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. In: ICSE-SEIP '22, Association for Computing Machinery, New York, NY, USA, pp. 343–352. http://dx.doi.org/10.1145/3510457.3513049.

Sotoudeh, M., Thakur, A.V., 2021. Provable repair of deep neural networks. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. In: PLDI 2021, Association for Computing Machinery, New York, NY, USA, pp. 588–603. http://dx.doi.org/10.1145/3453483.3454064.

Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. 11 (4), 341–359. http://dx.doi.org/10.1023/A:1008202821328.

Sun, B., Sun, J., Pham, L.H., Shi, J., 2022. Causality-based neural network repair. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 338–349. http://dx.doi.org/10.1145/3510003.3510080.

Tokui, S., Tokumoto, S., Yoshii, A., Ishikawa, F., Nakagawa, T., Munakata, K., Kikuchi, S., 2022. NeuRecover: Regression-controlled repair of deep neural networks with training history. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, pp. 1111–1121. http://dx.doi.org/10.1109/SANER53432.2022.00128.

Tran, H.-D., Cai, F., Diego, M.L., Musau, P., Johnson, T.T., Koutsoukos, X., 2019. Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. 18 (5s), http://dx.doi.org/10.1145/3358230.

Tran, H.-D., Yang, X., Manzanas Lopez, D., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T., 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I. Springer-Verlag, Berlin, Heidelberg, pp. 3–17. http://dx.doi.org/10.1007/978-3-030-53288-8_1.

Usman, M., Gopinath, D., Sun, Y., Noller, Y., Păsăreanu, C.S., 2021. NNrepair: Constraint-based repair of neural network classifiers. In: Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I. Springer-Verlag, Berlin, Heidelberg, pp. 3–25. http://dx.doi.org/10.1007/978-3-030-81685-8_1.

Wardat, M., Cruz, B.D., Le, W., Rajan, H., 2022. DeepDiagnosis: Automatically diagnosing faults and recommending actionable fixes in deep learning programs. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, pp. 561–572. http://dx.doi.org/10.1145/3510003.3510071.

Wardat, M., Le, W., Rajan, H., 2021. DeepLocalize: Fault localization for deep neural networks. In: Proceedings of the 43rd International Conference on Software Engineering. ICSE '21, IEEE Press, pp. 251–262. http://dx.doi.org/10.1109/ICSE43902.2021.00034.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

Wong, W.E., Debroy, V., Gao, R., Li, Y., 2014. The DStar method for effective software fault localization. IEEE Trans. Reliab. 63 (1), 290–308. http://dx.doi.org/10.1109/TR.2013.2285319.

Wong, W.E., Debroy, V., Li, Y., Gao, R., 2012. Software fault localization using DStar (D*). In: Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability. SERE '12, IEEE Computer Society, USA, pp. 21–30. http://dx.doi.org/10.1109/SERE.2012.12.

Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F., 2016. A survey on software fault localization. IEEE Trans. Softw. Eng. 42 (8), 707–740. http://dx.doi.org/10.1109/TSE.2016.2521368.

Wu, H., Li, Z., Cui, Z., Zhang, J., 2021. A mutation-based approach to repair deep neural network models. In: 2021 8th International Conference on Dependable Systems and their Applications. DSA, pp. 730–731. http://dx.doi.org/10.1109/DSA52907.2021.00106.

Yaghoubi, S., Fainekos, G., 2019. Gray-box adversarial testing for control systems with machine learning components. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. HSCC '19, Association for Computing Machinery, New York, NY, USA, pp. 179–184. http://dx.doi.org/10.1145/3302504.3311814.

Yang, X., Yamaguchi, T., Tran, H.-D., Hoxha, B., Johnson, T.T., 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. Springer-Verlag, Berlin, Heidelberg, pp. 221–236. http://dx.doi.org/10.1007/978-3-031-15839-1_13.

Zhang, H., Chan, W.K., 2019. Apricot: A weight-adaptation approach to fixing deep learning models. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. ASE '19, IEEE Press, pp. 376–387. http://dx.doi.org/10.1109/ASE.2019.00043.

Zhang, J.M., Harman, M., Ma, L., Liu, Y., 2022. Machine learning testing: Survey, landscapes and horizons. IEEE Trans. Softw. Eng. 48 (1), 1–36. http://dx.doi.org/10.1109/TSE.2019.2962027.

Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J., 2023. FalsifAI: Falsification of AI-enabled hybrid control systems guided by time-aware coverage criteria. IEEE Trans. Softw. Eng. 49 (4), 1842–1859. http://dx.doi.org/10.1109/TSE.2022.3194640.

Zhang, X., Zhai, J., Ma, S., Shen, C., 2021. AutoTrainer: An automatic DNN training problem detection and repair system. In: Proceedings of the 43rd International Conference on Software Engineering. ICSE '21, IEEE Press, pp. 359–371. http://dx.doi.org/10.1109/ICSE43902.2021.00043.

Deyun Lyu is a Ph.D. student in the Department of Information Science and Electrical Engineering, Kyushu University, Japan. He obtained his bachelor and master degree from Dalian University of Technology respectively in 2018 and 2020. His research topics include software testing, formal methods, and their applications to quality assurance of AI-enabled cyber–physical systems.

Yi Li is a master's student in the Department of Information Science and Electrical Engineering, Kyushu University, Japan. He obtained his bachelor degree from the University of Electronic Science and Technology of China. His research interests include software engineering for AI-enabled cyber–physical systems.

Zhenya Zhang is an assistant professor at Kyushu University, Japan. He obtained his Ph.D. from National Institute of Informatics, the Graduate University for Advanced Studies (SOKENDAI) in 2020. Before joining Kyushu University, he was a research fellow at Nanyang Technological University, Singapore. He is a recipient of "ACM SIGSOFT Distinguished Paper Award" at ASE 2024. His research topics include verification and testing of safety-critical systems, such as cyber–physical systems and AI-based systems.

Paolo Arcaini is an associate professor at the National Institute of Informatics, Japan. He received a Ph.D. in Computer Science from the University of Milan in 2013. Before joining NII, he held an assistant professor position at Charles University, Czech Republic. His current main research interests are related to testing of autonomous driving systems, testing of quantum programs, automatic repair of neural networks, and falsification of hybrid systems. More information at https://group-mmm.org/~arcaini/

Xiao-Yi Zhang received the B.Sc. degree and Ph.D. degree from Beihang University (BUAA), in 2010 and 2018, respectively. He is currently an associate professor with the School of Computer and Communication Engineering at University of Science and Technology Beijing (USTB), China. His research interests include software testing and safety analysis for smart software, such as cyber–physical systems, autonomous driving systems, and deep learning models. https://researchmap.jp/xiaoyi-zhang

Fuyuki Ishikawa is an associate professor at National Institute of Informatics and at Sokendai, Japan. His research interests include software engineering techniques for trustworthy smart systems, including testing, verification, and repair techniques for automated driving systems and learning-based systems. Ph.D. (Information science and technology, The University of Tokyo, 2007).

Jianjun Zhao is a full professor at the Faculty of Information Science and Electrical Engineering, Kyushu University. He received his B.S. degree in Computer Science from Tsinghua University, China, in 1987, and his Ph.D. in Computer Science from Kyushu University, Japan, in 1997. He served as an assistant/associate professor at the Fukuoka Institute of Technology, Japan (1997–2005), and later as a full professor at Shanghai Jiao Tong University, China (2005–2016). From 2002 to 2003, he was a visiting scientist at the MIT Laboratory for Computer Science (MIT LCS). His research interests span software engineering and programming languages for both classical and non-classical computing.