

Quality Assurance of AI-Enabled Cyber-Physical Systems

呂, 徳運

<https://hdl.handle.net/2324/7363810>

出版情報 : Kyushu University, 2024, 博士 (情報科学) , 課程博士
バージョン :
権利関係 :



Quality Assurance of AI-Enabled Cyber-Physical Systems



Deyun Lyu

Department of Advanced Information Technology
Graduate School of Information Science and Electrical Engineering
Kyushu University

A dissertation submitted for the degree of
Doctor of Philosophy in Engineering

December 2024

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

As I conclude this dissertation with my acknowledgments, I realize that I am reaching the end of my journey as a student. This marks a significant milestone and, simultaneously, a new beginning. I would like to express my sincere gratitude to everyone who helped me throughout my doctoral journey.

First and foremost, I am deeply indebted to my supervisor, Prof. Jianjun Zhao. When I first met him, I caused him quite a bit of trouble, and, of course, I continued to bother him many times throughout my doctoral journey. As I prepare to leave, I would like to express my most sincere gratitude for his invaluable guidance, unwavering support, and endless patience. His insightful advice, encouragement, and belief in my abilities have been instrumental in shaping this work.

I am deeply grateful to my co-advisers, Assistant Professor Zhenya Zhang and Associate Professor Paolo Arcaini, for their profound expertise, consistent guidance and unwavering support both in research and in life. They have taught me so much, and I truly appreciate their patience in helping me become a better researcher. They are the guiding lights on my academic path, it has been my great honour to work with them.

I am profoundly thankful to Prof. Ishikawa for providing me with the opportunity to intern at National Institute of Informatics. During my time there, I had the privilege of meeting like-minded individuals and making new friends. It was a memorable experience I will treasure forever. I also wish to extend my gratitude to Prof. Ichiro Hasuo, Associate Professor Xiao-Yi Zhang, Associate Research Professor Jie An and Dr. Thomas Laurent for their expertise and guidance.

I would like to express my sincere thanks to the members of the doctoral defence committee: Prof. Shaoying Liu, Prof. Jianjun Zhao, and Prof. Yasutaka Kamei, for their constructive feedback, insightful questions, and rigorous review of this dissertation.

Finally, I would like to extend my heartfelt thanks to my parents for their unconditional love, rock-solid support and countless sacrifices. Their unwavering belief in me has been a constant source of motivation.

Deyun Lyu

Kyushu University, Japan

December 2024

ABSTRACT

Cyber-Physical Systems (CPSs) have been broadly deployed in safety-critical domains, such as autonomous driving, medical devices, *etc.* In recent years, Artificial Intelligence (AI), mainly deep neural networks (DNN), has been increasingly adopted to control CPSs, giving birth to AI-enabled CPSs. Despite the great potential in terms of control, the safety of AI-enabled CPSs becomes a major concern. Due to the inherent uncertainty and opacity nature, a DNN controller may produce unforeseen and inexplicable behaviours that lead the plant to malfunctions and catastrophic hazards. Hence, it is essential to detect the potential unsafe behaviours of DNN controllers and identify the root causes that produce them. To enhance the reliability of AI-enabled CPSs, fixing the unsafe system behaviours is also necessary after identifying DNN parameters responsible for the violation, which has not received much attention. Besides, CPSs are often kept as private intellectual properties by industrial practitioners, very few benchmarks of AI-enabled CPSs are publicly available for quality assurance research.

To address these challenges, this dissertation explores the entire lifecycle of quality assurance of AI-enabled CPSs (i.e., benchmarking, falsification, fault localisation and repair), and makes four key contributions:

- *Benchmarking AI-enabled CPSs.* We present a public benchmark of industry-level CPSs in seven domains and build AI controllers for them via state-of-the-art deep reinforcement learning (DRL) methods. Based on that, we further perform a systematic evaluation of these AI-enabled systems with their traditional counterparts to identify current challenges

and future opportunities.

- *Falsification of AI-enabled CPSs.* We proposed a coverage-guided falsification framework FalsifAI, which exhaustively employs eight time-aware coverage criteria as guidance to explore the temporal behaviours of AI-enabled CPSs. These coverage criteria aim to capture different time-series features of DNN behaviours and provide better guidance to find violation cases to the system specification. The large-scale evaluation demonstrates the effectiveness of FalsifAI and eight time-aware coverage criteria.
- *Fault localisation of AI-enabled CPSs.* We propose SPECTACLE, that can localise the faulty parameters in DNN controllers. SPECTACLE considers the DNN inferences preceding the specification violation and uses forward impact to determine the DNN parameters that are more relevant to the DNN outputs. Then, it identifies which of these parameters are responsible for the specification violation, by adapting classic suspiciousness metrics. Moreover, we propose two versions of SPECTACLE, that consider differently the timestamps that precede the specification violation.
- *Search-based repair of DNN controllers of AI-enabled CPSs.* We propose a novel DNN repair approach guided by system-level specifications. The approach takes as input a system-level specification, some tests violating the specification, and some faulty DNN weights. The approach searches for alternative weight values with the goal of fixing the behaviour on the failing tests without breaking the passing tests. We also propose a heuristic that allows us to accelerate the search by avoiding the execution of some tests.

To summarize, this dissertation spans the entire lifecycle of building reliable AI-enabled CPSs, advances the understanding and deployment of AI in CPSs, paving the way for more trustworthy AI-enabled CPSs.

CONTENTS

Dedication	ii
Acknowledgments	iii
Abstract	v
List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Motivation	1
1.2 Challenges in Building Reliable AI-Enabled CPSs	2
1.3 Contributions	3
1.4 Organization of the Dissertation	5
2 Background	7
2.1 Cyber-Physical Systems and AI Controllers	7
2.2 System Specifications	9
2.3 CPS Testing Methodology	10
2.4 Spectrum-Based Fault Localisation	11

3	Related Work	13
3.1	CPS Benchmarks	13
3.2	AI Controllers for CPS	13
3.3	CPS Testing and Verification	14
3.3.1	Coverage-Guided Testing	14
3.3.2	Verification & Falsification for AI-Enabled Control Systems	15
3.4	Fault Localisation	17
3.4.1	DNN Fault Localisation	17
3.4.2	Fault Localisation of Classic CPSs	19
3.5	Repair Techniques for DNNs and CPSs	20
3.5.1	DNN Repair	20
3.5.2	Repair of DNN Controllers	20
3.5.3	CPS Repair	21
4	When Cyber-Physical Systems Meet AI: A Benchmark, an Evaluation, and a Way Forward	24
4.1	Introduction	24
4.2	Benchmark Collection	27
4.2.1	Benchmark Collection	27
4.2.2	Three Representative CPS Examples	29
4.3	Experiment Design	31
4.3.1	System Specifications	31
4.3.2	Research Questions	32
4.4	Experimental Results	36
4.4.1	RQ1. Performance of AI-enabled CPSs vs. Traditional CPSs	36
4.4.2	RQ2. Effectiveness of Falsification	39

4.4.3	RQ3. Performance of Hybrid Controllers	40
4.5	Discussions, Future Directions and Threats to Validity	42
4.6	Conclusion	43
5	FalsifAI: Falsification of AI-Enabled Hybrid Control Systems Guided by Time-Aware Coverage Criteria	45
5.1	Introduction	45
5.2	Preliminaries	49
5.2.1	System Model	50
5.2.2	Formal Specification	52
5.2.3	The Classic Black-Box Falsification Method	53
5.3	Time-Aware Coverage Criteria	54
5.3.1	Time Instant Coverage Criteria	56
5.3.2	Time Interval Coverage Criteria	57
5.3.3	Time Differential Coverage Criteria	58
5.4	Coverage-Guided Falsification	60
5.4.1	Overview of the Coverage-Guided Falsification	61
5.4.2	Hill Climbing-Based Mutation	64
5.5	Evaluation Design and Setting	64
5.5.1	Research Questions	65
5.5.2	Evaluation Metrics	65
5.5.3	Benchmarks	67
5.5.4	Experiment Setup	71
5.6	Evaluation Results	73
5.7	Threats to Validity	81
5.8	Conclusion and Future Work	83

6	SPECTACLE: Fault Localisation of AI-Enabled CPS by Exploiting Sequences of DNN Controller Inferences	84
6.1	Introduction	84
6.2	Problem Definition	87
6.2.1	AI-Enabled CPS	87
6.3	Motivation	89
6.4	SPECTACLE	92
6.4.1	Test Suite Execution	93
6.4.2	Forward Impact Calculation	95
6.4.3	Execution Spectrum Construction	96
6.4.3.1	Two approaches for spectrum construction	97
6.4.3.2	Scaling factor	99
6.4.4	Suspiciousness Score Calculation	100
6.5	Experiment Setup	102
6.5.1	Research Questions	102
6.5.2	Benchmarks	103
6.5.3	Compared Approaches	111
6.5.4	Evaluation Metrics	112
6.5.5	Software and Hardware Specifications	115
6.6	Experimental Results	115
6.6.1	RQ1: Effectiveness of SPECTACLE	119
6.6.2	RQ2: Hyperparameter Impact on the Effectiveness of SPECTACLE	121
6.6.3	RQ3: SPECTACLE _{UW} vs. SPECTACLE _W	122
6.6.4	RQ4: Suspiciousness Metric Impact on the Effectiveness of SPECTACLE	123
6.6.5	RQ5: Usefulness of Identified Weights by SPECTACLE	124
6.6.6	RQ6: Efficiency of SPECTACLE	125

6.7	Threats to Validity	126
6.8	Conclusion and Future Work	128
7	Search-Based Repair of DNN Controllers of AI-Enabled Cyber-Physical Systems	
	Guided by System-Level Specifications	130
7.1	Introduction	130
7.2	Problem Definition	133
7.2.1	AI-Enabled Cyber-Physical Systems	133
7.3	Proposed Repair Approach	134
7.3.1	Identification of the Weights to Repair	135
7.3.2	CONTRREP – Search-Based Repair of the DNN Controller	136
7.3.3	CONTRREP _{FAST} – Speeding Up the Fitness Computation	137
7.4	Experiment Design	139
7.4.1	Research Questions	140
7.4.2	Benchmarks	140
7.4.3	Compared Approaches	142
7.4.4	Evaluation Metrics and Assessment	143
7.5	Experiment Results	144
7.5.1	RQ1: Effectiveness of CONTRREP	144
7.5.2	RQ2: Usefulness of CONTRREP _{FAST}	146
7.5.3	RQ3: Effectiveness of CONTRREP in Fixing Failing Tests without Breaking Passing Tests	148
7.6	Threats to Validity	149
7.7	Conclusion	150
8	Conclusion	152
8.1	Contribution	152

8.2 Future Work	153
Bibliography	155
Published Papers	179

LIST OF FIGURES

2.1	Abstract workflow of CPS, with a traditional controller and an AI controller, respectively	8
4.1	Workflow summary of AI-enabled CPSs dataset and benchmark construction, and high-level empirical study design.	27
4.2	Performance comparison on systems with traditional and DRL controllers (RQ1) .	36
4.3	Performance comparison on systems with hybrid controllers (RQ3)	40
5.1	FalsifAI workflow: A coverage-guided falsification framework for DNN-enabled control systems	48
5.2	AI-enabled hybrid control system	51
5.3	RQ3 – Coverage Achieved by Successful and Unsuccessful Falsification Trials . .	77
5.4	RQ4 – Parameter Impact on Coverage Effectiveness	79
6.1	The architecture of a typical AI-enabled CPS	88
6.2	Example – Adaptive Cruise Control system that violates the specification $\varphi \equiv \square_{[0,T]}(d_{rel} \geq 10)$	91
6.3	SPECTACLE – Workflow of the approach	92
6.4	Forward impact of $W_{(i-1,j-1)(i,j)}$ to the final DNN output O_c	95
6.5	Window functions	98

6.6	Effectiveness of the compared approaches (in terms of detection rate DR) on the 13 groups of faulty benchmarks (using Kulczynski2 as suspiciousness metric SusMetr)	116
6.7	Effectiveness of the compared approaches (in terms of false discovery rate FDR) on the 13 groups of faulty benchmarks (using Kulczynski2 as suspiciousness metric SusMetr)	117
7.1	AI-enabled CPS model and an example ACC	133

LIST OF TABLES

4.1	The collected subject CPSs	28
4.2	The STL specifications of the benchmark CPSs. S1 follows the pattern: $\Box_I(\varphi_1)$; S2 follows the pattern: $\Box_I(\varphi_2)$; S3 follows the pattern: $\Diamond_I\Box_{I'}(\varphi_3)$; S4 follows the pattern: $\Box_I(\varphi_{41} \rightarrow \Diamond_{I'}\varphi_{42})$; S5 follows the pattern: $\Diamond_I(\varphi_5)$;	31
4.3	Falsification performance comparison between four existing falsification algo- rithms (RQ2)	38
4.4	Performance comparison on 4 hybrid benchmark CPSs with falsification tools (RQ3)	40
5.1	The structures of the neural network controllers used in our experiments	70
5.2	Parameter selection for coverage criteria. τ : secs	71
5.3	RQ1 and RQ2 – Experimental results for ACC, AFC, and DPC with their spec- ifications. (<i>SR</i> : number of successful runs finding a falsifying input (out of 30). <i>time</i> : average time of successful runs. <i>#sim</i> : average number of simulations of successful runs). Best results are highlighted in gray	74
5.4	RQ2 – Comparison of guidance of coverage criteria (number of benchmarks in which the approach on the row is statistically significantly better than the one on the column, using the Fisher’s exact test). The performance of a criterion is indicated by the degree of gray; a highlight in darker gray indicates a better result.	75

5.5	RQ5 – Comparison between time taken by simulation and coverage computation.	
	Time in seconds	80
6.1	Suspiciousness metrics SusMetr used in this paper	100
6.2	An illustrative example of SPECTACLE _{UW} (gray cells indicate top- k)	101
6.3	Benchmark AI-enabled CPSs – CPSs and their DNN controllers ($ \mathcal{W} $ is the number of weights of C ; $ \mathcal{M} $ is the number of blocks of $\mathcal{M}$)	103
6.4	Training details of the DNN controllers C	107
6.5	Number of faulty benchmarks (for each $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^i$ ($i \in \{1, \dots, 5\}$) and for $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$) generated from the 13 correct benchmarks	110
6.6	Distribution of safety rate SR across the faulty benchmarks	111
6.7	Effectiveness of the compared approaches in terms of AUC_{DR} with $s \leq 20\%$, using Kulczynski2 as suspiciousness metric SusMetr. The higher the value, the better (the best approach and second best approach are highlighted in gray and light gray, respectively)	118
6.8	Effectiveness of the compared approaches in terms of AUC_{FDR} with $s \leq 20\%$, using Kulczynski2 as suspiciousness metric SusMetr. The lower the value, the better (the best approach and second best approach are highlighted in gray and light gray, respectively)	119
6.9	RQ1, RQ2, RQ3 – Statistical comparison between each pair of approaches in terms of AUC_{DR} . (Legend. $\equiv$: there is no difference between the two approaches. $\checkmark$, $\checkmark\checkmark$, $\checkmark\checkmark\checkmark$: the approach on the rows is significantly better than the approach on the column with strength <i>small</i> , <i>medium</i> , <i>large</i> . $\times$, $\times\times$, $\times\times\times$: the approach on the row is significantly worse than the approach on the column with strength <i>small</i> , <i>medium</i> , <i>large</i>)	120

6.10	RQ1, RQ2, RQ3 – Statistical comparison between each pair of approaches in terms of AUC_{FDR} . (Legend. $\equiv$: there is no difference between the two approaches. ✓, ✓✓, ✓✓✓: the approach on the rows is significantly better than the approach on the column with strength <i>small</i> , <i>medium</i> , <i>large</i> . ✗, ✗✗, ✗✗✗: the approach on the row is significantly worse than the approach on the column with strength <i>small</i> , <i>medium</i> , <i>large</i>)	120
6.11	RQ3 – Comparison between $SPECTACLE_{UW}$ and $SPECTACLE_W$ (better approach in gray)	122
6.12	RQ4 – Statistical comparison between each pair of suspiciousness metrics in terms of AUC_{DR} . (Legend. $\equiv$: there is no difference between the two metrics. ✓, ✓✓, ✓✓✓: the metric on the rows is significantly better than the metric on the column with strength <i>small</i> , <i>medium</i> , <i>large</i> . ✗, ✗✗, ✗✗✗: the metric on the row is significantly worse than the metric on the column with strength <i>small</i> , <i>medium</i> , <i>large</i>)	123
6.13	RQ4 – Statistical comparison between each pair of suspiciousness metrics in terms of AUC_{FDR} . (Legend. $\equiv$: there is no difference between the two metrics. ✓, ✓✓, ✓✓✓: the metric on the rows is significantly better than the metric on the column with strength <i>small</i> , <i>medium</i> , <i>large</i> . ✗, ✗✗, ✗✗✗: the metric on the row is significantly worse than the metric on the column with strength <i>small</i> , <i>medium</i> , <i>large</i>)	124
6.14	RQ5 – Results of repair of AI-enabled CPSs using the weights identified by $SPECTACLE$	125
6.15	RQ6 – Computational cost of $SPECTACLE$ (secs.)	126
7.1	AI-enabled CPSs $\mathcal{M}^C$ and correctness measure of its faulty versions $\mathcal{M}_f^C$ ($CM(\mathcal{M}_f^C, \varphi, TS)$ (%))	141

7.2	RQ1 and RQ2 – Average $CM (\mathcal{M}^{C_{best}}, \varphi, TS)$ (%) of CONTRREP and CONTRREP _{FAST} across the 10 runs. Values of CM for CONTRREP _{FAST} report the exact value and not the approximate value computed by the approach	145
7.3	RQ2 – Comparison of CONTRREP _{FAST} and CONTRREP in terms of number of executed tests <i>ExecTests</i> . Legend: ($\equiv$: no sign. diff. between the two approaches. $\checkmark$: the approach on the left is <i>better</i> than the one on top, $\times$ means that it is worse; the num. of symbols is the strength: <i>negligible</i> ($\checkmark$, $\times$), <i>small</i> ($\checkmark\checkmark$, $\times\times$), <i>medium</i> ($\checkmark\checkmark\checkmark$, $\times\times\times$), <i>large</i> ($\checkmark\checkmark\checkmark\checkmark$, $\times\times\times\times$)	146
7.4	RQ2 – Comparison of CONTRREP and CONTRREP _{FAST} in terms of $CM (\mathcal{M}^{C_{best}}, \varphi, TS)$ (legend as in Table 7.3)	147
7.5	RQ3 – Number of repaired and broken tests (average across 10 runs)	149

1 | INTRODUCTION

This chapter first motivates this dissertation, then discusses the challenges in building reliable AI-enabled CPSs. Finally, it summarizes the contributions and introduces the organisation of this dissertation.

1.1 MOTIVATION

A Cyber-physical system (CPS) is the combination of a physical system (plant) and a computational unit (controller), in which the controller monitors and controls the behaviours of physical system in real time, according to the system states and external environments. In recent years, *deep neural networks* (DNNs) have been increasingly deployed as controllers to perform extremely complex control tasks that require advanced intelligent autonomy, such as autonomous driving and medical devices, *etc.* Such CPSs empowered by AI-based controllers are called as AI-enabled CPSs. Compared with classic controllers, DNN controllers can approximate any non-linear function in theory and have powerful parallel processing capability, so AI-enabled CPSs have recently become the subject of intense research and demonstrated great promises.

Despite their intelligence when facing various complex situations, DNN controllers also have inherent drawbacks, such as their complexity and unaccountable decision logic. Indeed, a DNN controller may produce unforeseen and inexplicable behaviours that lead the plant to malfunctions and catastrophic hazards, which can further bring irreversible social and economic losses.

In practice, AI-enabled CPSs are always deployed under safety-critical scenarios to perform complex control tasks, which raise the risk and probability of catastrophic accidents. Hence, quality assurance of AI-enabled CPSs is of critical importance.

1.2 CHALLENGES IN BUILDING RELIABLE AI-ENABLED CPSs

Quality assurance of AI-enabled systems is a critical challenge in software engineering due to the non-deterministic and opaque nature of AI components. Unlike traditional software, AI-enabled systems rely heavily on data-driven models, making it difficult to ensure their safety and reliability. Testing and validation of such systems require new techniques, such as coverage-guided testing and automated test generation, to detect unpredictable behaviours. Fault localization and diagnosis are also complex, as system failures may stem from data, models, or algorithms. Furthermore, system-level behaviour assurance is critical, given the interactions between AI components and other modules, requiring simulation and formal methods. To maintain long-term reliability, mechanisms for repairing and updating AI models are essential, particularly in dynamic environments. Additionally, explainability is crucial for building trustworthy AI-enabled systems, especially in safety-critical applications. These aspects highlight the urgent need for enhancing the safety, trustworthiness, and reliability of AI-enabled systems.

In particular, when AI components are used as controllers in CPSs, these challenges become even more pronounced. The deployment of DNN controllers introduces a series of problems and brings unique challenges to the quality assurance of AI-enabled CPSs.

Lack of publicly available benchmarks. Unlike the large and growing open-source community in conventional software, CPSs are often kept as private intellectual properties by industrial practitioners, in which strong domain knowledge and know-how are encoded. Therefore, even up to the present, very few benchmarks of AI-enabled CPSs are publicly available for analysis.

Complex temporal logic and feedback loops (the combination of AI and CPSs). As a decision

component, a DNN controller in a CPS maintains continuous state evolution of the physical system. This ongoing feedback loop, combined with complex temporal logic, makes it challenging to ensure the correctness of system behaviours – especially when the DNN controller’s control decision depends on prior inputs and states. Besides, pinpointing the exact parameters within the DNN controller responsible for specification violations is challenging because it requires analysing sequences of control decisions over time, rather than single, isolated control decisions.

Lack of explainability (general challenge of AI systems). Classic controllers, such as those based on proportional integral derivative (PID) and model predictive control (MPC), often adopt human-crafted heuristics or linear models to make control decisions. Unlike classic controllers, the decision-making processes of DNN controllers in AI-enabled CPSs are often opaque, making it difficult to trace or predict behaviours. This complexity poses a challenge for debugging and ensuring the DNN controllers’ reliability.

Lack of ground truth label for single control decision (the unique challenge of DNN controllers compared to DNN-based image classifiers). Unlike classification tasks, control decisions in CPSs lack clear correctness criteria for each timestamp. Assessing DNN outputs requires evaluating overall system behaviour, often against system-level temporal specifications, rather than checking individual control action.

Infinite state space of input signal and DNN controllers (search efficiency in testing and repair) The vast search space for possible inputs and control strategies in CPSs complicates the process of finding diverse system behaviours and efficiently repairing faults, as traditional testing and repair methods may not scale well to such high complexity.

1.3 CONTRIBUTIONS

This dissertation makes the following contributions:

- **A public benchmark of industry-level AI-enabled CPSs across various domains.**

We present a public benchmark of industry-level CPSs in seven domains and build AI controllers for them via state-of-the-art deep reinforcement learning (DRL) methods. Based on that, we further perform a systematic evaluation of these AI-enabled systems with their traditional counterparts to identify current challenges and future opportunities. Our key findings include (1) AI controllers do not always outperform traditional controllers, (2) existing CPS testing techniques (falsification, specifically) fall short of analysing AI-enabled CPSs, and (3) building a hybrid system that strategically combines and switches between AI controllers and traditional controllers can achieve better performance across different domains. Our results highlight the need for new testing techniques for AI-enabled CPSs and the need for more investigations into hybrid CPSs to achieve optimal performance and reliability.

- **Falsification of AI-enabled CPSs guided by eight time-aware coverage criteria.** We make an early attempt at exploring the temporal behaviour determined by the repeated executions of the DNN controllers in CPSs and first propose eight time-aware coverage criteria specifically designed for neural network controllers in the context of CPSs, which consider different features by design: the simple temporal activation of a neuron, the continuous activation of a neuron for a given duration, and the differential neuron activation behaviour over time. Secondly, we introduce a falsification framework, named *FalsifAI*, that exploits the coverage information for better falsification guidance. Namely, inputs of the controller that increase the coverage (so improving the exploration of the DNN behaviours), are prioritized in the exploitation phase of robustness minimization. Our large-scale evaluation over a total of 3 typical CPS tasks, 6 system specifications, 18 DNN models and more than 12,000 experiment runs, demonstrates 1) the advantage of our proposed technique in outperforming two state-of-the-art falsification approaches, and 2) the usefulness of our proposed time-aware coverage criteria for effective falsification guidance.
- **Fault localisation approach of AI-enabled CPSs by exploiting sequences of DNN**

controller inferences. We propose SPECTACLE, that can localise the faulty parameters in DNN controllers. SPECTACLE considers the DNN inferences preceding the specification violation and uses forward impact to determine the DNN parameters that are more relevant to the DNN outputs. Then, it identifies which of these parameters are responsible for the specification violation, by adapting classic suspiciousness metrics. Moreover, we propose two versions of SPECTACLE, that consider differently the timestamps that precede the specification violation. We experimentally evaluate the effectiveness of SPECTACLE on 6,067 faulty benchmarks, spanning over different application domains. The results show that SPECTACLE can detect most of the faults.

- **Search-based repair for DNN controllers of AI-enabled CPSs.** We propose a novel DNN repair approach that is guided by system-level specifications. The approach takes in input a system-level specification, some tests violating the specification, and some faulty DNN weights. The approach searches for alternative weight values with the goal of fixing the behaviour on the failing tests without breaking the passing tests. We also propose a heuristic that allows us to accelerate the search by avoiding the execution of some tests. Experiments on real-world AI-enabled CPSs show that the approach effectively repairs their controllers.

1.4 ORGANIZATION OF THE DISSERTATION

Chapter 2 introduces necessary preliminaries to better understand this dissertation and Chapter 3 reviews some related work. Then, Chapter 4 presents a public benchmark of industry-level CPSs along with a comprehensive evaluation. Chapter 5 introduces FalsifAI, a time-aware coverage criteria guided falsification framework. Chapter 6 introduces SPECTACLE, a fault localisation approach for identifying the faulty parameters in DNN controllers. Taking the detected faulty parameters as input, Chapter 7 presents a search-based repair approach for DNN con-

trollers guided by system-level specifications. Finally, Chapter 8 concludes the contribution of this dissertation and discusses some promising directions for future work.

2 | BACKGROUND

This chapter introduces the necessary preliminaries to better understand this dissertation. This section gives an introduction on CPSs and AI controllers. Specifically, we describe deep reinforcement learning (DRL) based AI controllers in this paper. We also briefly describe *Signal Temporal Logic (STL)*, a specification language of CPSs, and *falsification*, a typical and important testing method for CPSs based on STL.

2.1 CYBER-PHYSICAL SYSTEMS AND AI CONTROLLERS

CPSs have been widely adopted in safety-critical industrial domains, such as automotive systems, avionics, medical devices, *etc.* CPSs make use of computer programs to monitor and control the behaviours of mechanical systems.

Fig. 2.1(a) shows a brief overview of a CPS, which consists of a plant M and a controller C . The plant is a physical environment, whose next state y is decided by the current state and the control command u . The control command u is issued by a software controller. Commonly-used traditional controllers include *proportional integral derivative (PID) control*, *model predictive control (MPC)*, *etc.* (detailed in §4.2). These controllers make the control decision u based on the state y and an external input i .

In CPSs, *signals* are passed between components in a system and between the system and the external environment. Formally, a signal $s: [0, T] \rightarrow \mathbb{R}^{\text{dim}}$ is defined as a *time-variant* function,

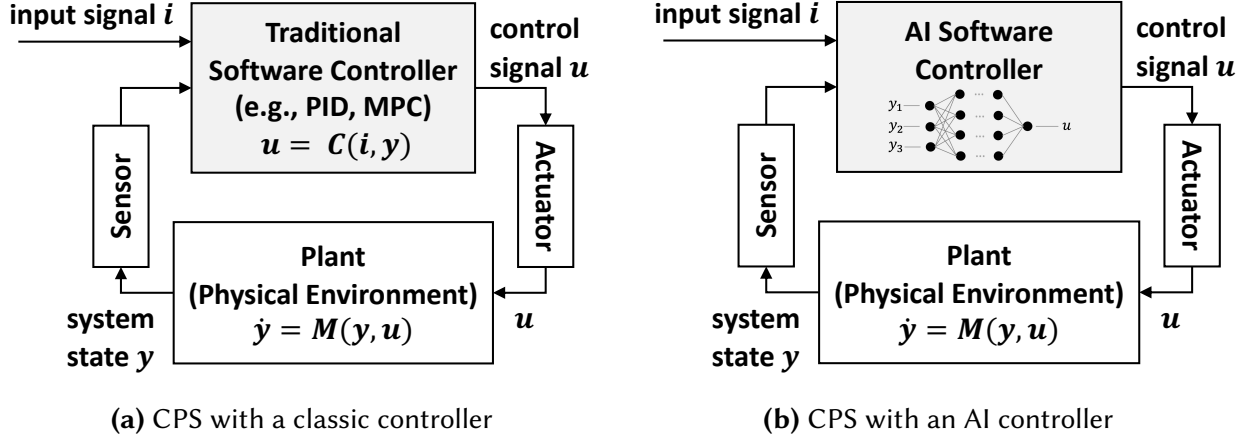


Figure 2.1: Abstract workflow of CPS, with a traditional controller and an AI controller, respectively

where $T \in \mathbb{R}_+$ denotes the time horizon, and $\dim \in \mathbb{N}_+$ is called the dimension of the signal. In Fig. 2.1(a), the system state y , the control decision u , and the external input i are all signals. Sensors and actuators are used to collect, process, and pass these signals between the plant M and the controller C in a CPS model.

AI Controller. With the boom of AI in the past few years, practitioners have started considering replacing traditional controllers with AI controllers. Fig. 2.1(b) gives an overview of *AI-enabled CPS*.

An AI controller can be implemented and trained in different ways, such as *supervised learning* and *reinforcement learning*. Among different methods, deep reinforcement learning (DRL) is considered as the state-of-the-art, and has succeeded in various application domains [15]. In this study, we mainly focus on DRL-based controllers. We briefly introduce DRL below.

Deep Reinforcement Learning. Unlike supervised learning, DRL does not require collecting labelled datasets beforehand. Instead, it follows the “*trial-and-error*” paradigm—learning the best strategy via repeated interactions with the external environment. Technically, DRL requires to train an *agent* in an environment so that the agent learns the best *policy* that maximizes the cumulative *reward*. At each step, the agent has a *state* given by the environment and the agent

itself, and it is required to take an *action* to transit from the current state to a new one. A policy is a function that maps a state-action pair to a real-valued reward [16]. For a fixed state, the reward of each action depends on the feedback from the environment. In order to learn the best policy, the agent repeatedly tries an action to obtain a reward, and updates the policy accordingly. There is a variety of policy updating methods, such as *Deep deterministic policy gradient (DDPG)* [92], *Twin-delayed deep deterministic policy gradient (TD3)* [54], *Actor-critic (A2C)* [36], *Proximal policy optimization (PPO)* [131], *Soft actor-critic (SAC)* [63], etc. We refer readers to [74] for more details. DRL can be naturally applied to learn control policies for CPS by interacting with its *plant*, which is the physical environment of the system.

2.2 SYSTEM SPECIFICATIONS

A system specification describes a property that the system must guarantee. In the domain of CPS, a widely-adopted formalism is *Signal Temporal Logic (STL)* [35], that has a *formal syntax* to allow engineers to express their desired properties, and a *formal semantics* that allows to check whether the system output satisfies the specification.

Definition 1 (STL syntax). Let $\vec{v}$ be an N -dimensional vector. *Atomic predicates* are represented as $\alpha ::= (f(\vec{v}) > 0)$, where f is a function that maps $\vec{v}$ to a real number. The syntax of an STL formula φ is defined as follows:

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

Here, I is a closed time interval $[a, b]$, where $a, b \in \mathbb{R}$ and $a < b$. Note that we only consider specifications with bounded time, such that the specification satisfaction can always be assessed. $\Box_I$, $\Diamond_I$, and $\mathcal{U}_I$ are temporal operators *always*, *eventually*, and *until*, which allow to express complex temporal properties. ◁

An advantage of STL is its quantitative robust semantics [35, 51], that tells not only *whether* a system output $\mathbf{v}$ satisfies a specification φ , but also *how robustly* it does so. Next, we give the definition of quantitative robust semantics of STL, that maps a system output signal $\mathbf{v}$ and an STL formula to a real number. Here, the signal $\mathbf{v}: [0, T] \rightarrow \mathbb{R}^N$ is a time-variant function that represents an execution trace of the system, where T is the time horizon, and N is the dimension of $\mathbf{v}$.

Definition 2 (Quantitative robust semantics). Given a system output $\mathbf{v}$ and an STL formula φ , the *robust semantics* provides a quantity $\llbracket \mathbf{v}, \varphi \rrbracket \in \mathbb{R} \cup \{\infty, -\infty\}$ that tells *how robustly* $\mathbf{v}$ satisfies or violates φ . The semantics is defined as follows:

$$\begin{aligned} \llbracket \mathbf{v}, \alpha \rrbracket &:= f(\mathbf{v}(0)) & \llbracket \mathbf{v}, \perp \rrbracket &:= -\infty & \llbracket \mathbf{v}, \neg\varphi \rrbracket &:= -\llbracket \mathbf{v}, \varphi \rrbracket \\ \llbracket \mathbf{v}, \varphi_1 \wedge \varphi_2 \rrbracket &:= \min(\llbracket \mathbf{v}, \varphi_1 \rrbracket, \llbracket \mathbf{v}, \varphi_2 \rrbracket) & \llbracket \mathbf{v}, \varphi_1 \vee \varphi_2 \rrbracket &:= \max(\llbracket \mathbf{v}, \varphi_1 \rrbracket, \llbracket \mathbf{v}, \varphi_2 \rrbracket) \\ \llbracket \mathbf{v}, \Box_I \varphi \rrbracket &:= \inf_{t \in I} (\llbracket \mathbf{v}^t, \varphi \rrbracket) & \llbracket \mathbf{v}, \Diamond_I \varphi \rrbracket &:= \sup_{t \in I} (\llbracket \mathbf{v}^t, \varphi \rrbracket) \\ \llbracket \mathbf{v}, \varphi_1 \mathcal{U}_I \varphi_2 \rrbracket &:= \sup_{t \in I} (\min(\llbracket \mathbf{v}^t, \varphi_2 \rrbracket, \inf_{t' \in [0, t)} \llbracket \mathbf{v}^{t'}, \varphi_1 \rrbracket)) \end{aligned}$$

where $\mathbf{v}^t$ denotes the t -shift of $\mathbf{v}$, namely, $\mathbf{v}^t$ maps a timestamp $t' \in [0, T - t]$ to $\mathbf{v}^t(t') = \mathbf{v}(t + t')$.

By the robust semantics, we can infer the satisfaction of $\mathbf{v}$ to φ : if $\llbracket \mathbf{v}, \varphi \rrbracket$ is positive, it means that $\mathbf{v}$ satisfies φ (i.e., $\mathbf{v} \models \varphi$); if $\llbracket \mathbf{v}, \varphi \rrbracket$ is negative, it means that $\mathbf{v}$ violates φ (i.e., $\mathbf{v} \not\models \varphi$). $\triangleleft$

2.3 CPS TESTING METHODOLOGY

Falsification is an established methodology in CPS testing [11, 34, 37, 178]. Given a model M^C and an STL specification φ , falsification searches for a counterexample input signal i such that the corresponding output signal $M^C(i)$ violates φ . In this way, it proves the unsatisfiability of the system model M^C to the specification φ .

Alg. 1 describes a basic falsification algorithm. Essentially, this algorithm formulates the testing problem as an optimization one, by taking the robustness as the objective function. In this

Algorithm 1 The classic falsification algorithm

Require: CPS model M^C , an STL specification φ , a budget B_G .

```

1: function HILL-CLIMB-FALSIFY( $M^C, \varphi, B_G$ )
2:   initialize a placeholder  $i$  ▷ best input so far
3:    $rb \leftarrow \infty$  ▷ minimum robustness so far
4:   for  $k \in \{0, \dots, B_G\}$  do
5:      $i_k \leftarrow \text{HILL-CLIMB}(\langle i_l, rb_l \rangle_{l=0, \dots, k-1})$ 
6:      $rb_k \leftarrow \llbracket M^C(i_k), \varphi \rrbracket$  ▷ compute robustness
7:     if  $rb_k < rb$  then
8:        $rb \leftarrow rb_k, i \leftarrow i_k$ 
9:   return  $\begin{cases} i & \text{if } rb < 0 \\ \text{Failure} & \text{otherwise, no violation is found} \end{cases}$ 
    
```

algorithm, the CPS model M^C is treated as a black box—only its input signal i and the corresponding output signal $M^C(i)$ (system states) can be observed. In the main loop (Lines 4–8 in Alg. 1), falsification tries different input signals i_k to minimize the robustness value rb_k computed based on $M^C(i_k)$ and φ , so that the system is closer to violation of the specification φ . Once a negative robustness is observed, falsification will terminate and return that input as a counterexamples. Otherwise, it will keep searching until the time budget is run out (Line 9).

To solve the optimization problem, falsification employs *hill-climbing optimization* algorithms, as shown in Line 5 of Alg. 1. *Hill-climbing optimization* algorithms are a family of stochastic meta-heuristics-based optimization algorithms. In general, these algorithms first do random sampling in the input space and obtain the robustness values of these inputs. Then based on the observations, they propose new samplings with the aim of decreasing robustness. Typical such algorithms include CMAES [65], Global Nelder-Mead [98], Simulated Annealing [147], *etc.*

2.4 SPECTRUM-BASED FAULT LOCALISATION

Spectrum-based Fault localisation (SBFL) is an established technique to identify the root cause of software failures [cite]. Given a test suite TS and a program P , it operates by analyzing the execution traces of P triggered by TS and records frequencies of entities being executed in P , such as statements. Then it constructs an execution spectrum for each program entity, $\{e_p, e_f, u_p, u_f\}$,

where e and u denote an entity is executed or not separately; while p and f represent a test case passes or fails separately. Finally, it computes suspiciousness scores for each entity. Higher scores means current entity is more related to software failures. With dynamic program execution data, SBFL can assist the developers to localise software defects effectively.

3 | RELATED WORK

This chapter discusses related work of benchmarking CPSs, AI controllers, CPS Testing and Verification, fault localisation and DNN repair.

3.1 CPS BENCHMARKS

As mentioned in 1.2, collecting benchmarks of CPS is challenging. An annual workshop, namely ARCH, aims to mitigate this problem by bringing together CPS benchmarks and holding competitions¹ for different research topics. The most relevant competitions to this paper are *Artificial Intelligence and Neural Network Control Systems* [74] and *Falsification* [47]. However, the benchmark in [47] only includes traditional CPS rather than AI-enabled CPS. While the benchmark in [74] includes AI-enabled CPS, their benchmark includes less and simpler CPS such as *Cart-Pole*, which are not from industrial application domains.

3.2 AI CONTROLLERS FOR CPS

With the rapid development of AI, practitioners consider replacing traditional controllers with DNN controllers [31, 52, 59, 97, 122, 160]. A DNN controller can be trained in different ways, including *supervised learning* [67] and *deep reinforcement learning (DRL)* [15]. Supervised learning requires to collect system executions of an original CPS to train new DNN controllers. For

¹<https://cps-vo.org/group/ARCH/FriendlyCompetition>

example, Hertneck et al. [67] use DNNs to approximate a robust MPC controller. In contrast, DRL follows the “*trial-and-error*” paradigm, learning the best strategy via interactions with environment. Duan *et al.* [42] proposed a benchmark on continuous control tasks. However, this benchmark involves game scenarios only such as *Cart-Pole* and *Inverted Pendulum*, rather than complex real-world environments. A recent work [134] applies DRL for DNN controller training.

3.3 CPS TESTING AND VERIFICATION

Currently, most of the research efforts are devoted to formal verification of such systems, since it can give rigorous proofs on their safety. For example, reachability analysis [69, 145, 157] has been extensively studied and considered as one of the most effective ways to verify AI controllers. The other line of formal verification of AI controllers is based on constraint solving, such as DLV [70], *etc.* Due to the intrinsic scalability problem of verification, their evaluations are usually on simple benchmarks. Falsification is considered as a method that suffers much less from the scalability issue than verification, and this is confirmed by the empirical study [119], in which they compared the effectiveness of model checking and testing on CPS. However, existing falsification research mostly focuses on CPS with traditional controllers, and does not consider the specific structure of neural networks.

3.3.1 COVERAGE-GUIDED TESTING

Coverage-Guided Testing (CGT) is an established methodology in software testing. The central idea of CGT is to generate test cases, guided by coverage criteria (e.g., statement coverage, function coverage, branch coverage, etc.), to achieve high testing adequacy and increase testing confidence [28, 53, 84, 124].

More recently, with the fast development of machine learning, especially *deep neural network (DNN)*, a great number of works emerged on DNN testing [21, 41, 81, 105, 126, 138, 140,

158]. Due to the complexity and uninterpretable decision logic, testing of DNN systems is much more challenging than that of traditional software systems. Notably, CGT is one of the most successfully adopted methods. We discuss some representative works: *DeepXplore* [126] is the first work that proposes *neuron coverage*, which inspires the coverage criteria such as NC and TNC in this work; *DeepGauge* [105] proposes a family of more fine-grained coverage criteria, and shows their potential use cases; *DeepTest* [140], *TensorFuzz* [123] and *DeepHunter* [158] propose coverage-guided fuzz testing frameworks, which apply to a set of well-known DNN models for bug detection. *DeepConcolic* [138] leverages similar neuron coverage criteria as feedback to guide concolic testing of DNNs. Unlike the above coverage notions defined for test suites, *surprise adequacy* [81] is a criterion that focuses on the novelty of individual inputs, based on the heuristic that these inputs are more likely to expose system failures. Application of surprise adequacy to the systems considered in this work is not straightforward, because we do not have control on the plant state y that is part of the input of the DNN. More relevant work along this direction regarding DNN testing could be referred to the recent survey [168].

However, all these existing works mentioned above deal with non-dynamical applications, such as picture classification, and, therefore, they did not take the temporal behaviors of neural network systems into consideration. In the domain of hybrid control systems, neural network controller gives a series of decisions over time to control the plant, and hence, compared to an instant decision at a moment, their temporal performance is of much greater importance.

3.3.2 VERIFICATION & FALSIFICATION FOR AI-ENABLED CONTROL SYSTEMS

Quality assurance of hybrid control systems has been one of the long standing industrial challenges in the past decade, whose importance is further elevated in the era of *Industry 4.0*. The applications of hybrid control systems span over autonomous driving, avionics, medical systems, etc.—most of these domains are safety-critical. Reasoning about these systems is challenging, because of their hybrid nature that combines continuous and discrete dynamics in single systems.

Formal verification of such systems is, in general, often impossible, due to their infinite state space. Hence, existing research in verification mostly relies on their highly abstracted models, e.g., hybrid automata [66], timed automata [7], or some novel formal languages [127, 136]. Recently, research on verification of AI-enabled control systems emerges, such as [69, 143]. These works adopt *reachability analysis* to approximate the reachable set of the system states. Compared to these works, our work is different in the following two aspects: first, [69, 143] consider white-box systems with limited types of dynamics, e.g., linear systems, etc. (in the form of analytical models), while our technique works for black-box models having any type of complex dynamics (not requiring any analytical model); second, [69, 143] consider system reachability (i.e., safety properties) as their specification, while our framework can evaluate arbitrary temporal properties expressed in STL. Note that the benchmarks considered in our experiments (see §5.5.3) are not supported by verification approaches such as [69, 143]: AFC and DPC do not provide any analytical model; ACC, instead, is technically handled by [143], but our changes to the model made the input space much larger, making the application of verification infeasible.

Falsification [4, 10, 33, 38, 113, 161, 171–174, 177] emerged as an alternative to verification, by solving its dual problem—finding one state that violates the specification. This gives better flexibility to the method design, and so there comes the classic optimization-based falsification technique, as introduced in §2.3. No much attention has been paid to falsification of AI-enabled control systems as we do, among which [159] is an early attempt. The technical fundamental of [159] is to obtain an approximated model based on the linearization information of the system under test; in comparison, our approach works for black-box plants, whose dynamics can be highly complex or even inaccessible (if they are confidential products from third-party providers). VerifAI [39] provides a platform that integrates testing, data augmentation, fault analysis, etc. of AI-enabled systems. In this platform, falsification is the major component in charge of system testing. However, the falsification technique adopted by VerifAI is an existing one [37], which targets a system model with DNN for environment perception; in comparison, the DNN in our

system model is used for control decision making, and our technique considers the outputs of the intermediate layers of DNNs and use the information as a guidance to explore the system behavior. Using coverage as a search guidance has been studied in falsification [4, 32, 38] of classic controllers. The main challenge is how to properly define coverage criteria. In [32], the authors define coverage based on the blocks of Simulink models, and integrate it as a part of the objective function; in [38], the authors use “star discrepancy” as a measure of input space coverage; in [4], the authors classify inputs according to their robustness values, and define coverage based on the clustered inputs. The aim of using coverage in these works is to explore the system space, so that falsification is not biased to exploitation.

However, simply defining coverage in terms of input space may not be enough for DNN controllers: first, the input space, identified by continuous input signals, is too huge to be covered adequately; moreover, the relation between exploration of input space and safety violation is weak. Considering the complexity of DNN-controlled hybrid systems, coverage criteria defined in terms of their internal states can give better guidance as confirmed in our study.

3.4 FAULT LOCALISATION

Fault localisation is an established technique in software engineering domain, which aims to find the root cause of a system failure. In this section, we discuss related work from two perspectives: fault localisation approaches for single-inference DNNs (§3.4.2), and fault localisation approaches for classic CPSs (§3.4.2).

3.4.1 DNN FAULT LOCALISATION

Different works have been proposed for DNN fault localisation. A first group of approaches target neurons and weights of the DNN model in the localisation. Ma et al. [106] introduce *MODE*, which utilises state-differential analysis between misclassification and correct classification to

identify faulty neurons. Eniser et al. [44] propose *DeepFault* that localises suspicious neurons, by building their spectra based on the activation status and using SBFL metrics (Tarantula, Ochiai, and D*) to assign a suspiciousness score. Duran et al. [43] introduce *DeepFaultGr* that extends *DeepFault*, and show that doing fault localisation for different misclassifications together can produce sub-optimal results, because of the masking effect between different misclassifications. Sohn et al. [133] propose *Arachne* for repair of DNN models, in which they also handle the fault localisation problem using gradient loss and forward impact to identify the faulty neuron weights. With the previous approaches, SPECTACLE shares the fact that its target weights (as *Arachne*), and that uses SBFL (as *DeepFault* and *DeepFaultGr*). However, there are three main differences: they rely on ground-truth labels, while SPECTACLE relies on the system level specification; they consider single inferences of the DNN, while SPECTACLE considers sequences of inferences; SPECTACLE gives more or less importance to an inference in the spectrum depending on the time when the inference occurred.

A second group of approaches, instead, target faults in the configuration of the DL program, like wrong activation function or missing/redundant/wrong layer. For example, Wardat et al. [149] propose *DeepLocalize* that targets the faults in DNN model structures (e.g., faulty hyperparameter), by analysing the traces produced by the DNN over training. Wardat et al. [150] propose *DeepDiagnosis* that extends *DeepLocalize* to detect more faults during training and, moreover, it also suggests possible actions to fix the fault. Cao et al. [26] introduce *DeepFD*, that, similarly to *DeepLocalize* and *DeepDiagnosis*, targets faults in DL programs. However, *DeepFD* does not use a fixed set of detection rules, but adopts a learning-based fault localisation framework which monitors the DNN model training and infers the suspicious fault types. Ghanbari et al. [57] propose *deepmufi*, which employs *mutation-based fault localisation (MBFL)* to localise the faults in DNN models introduced by wrong DL program configuration. Specifically, it mutates the faulty DNN models and implements two commonly-used approaches in MBFL, Metallaxis and MUSE, and then filters out the most suspicious elements based on suspiciousness values of

different mutants. These approaches are different from SPECTACLE, as their assumption is that the fault is in the DL program; SPECTACLE, instead, should be used to detect faults in the setting of the parameters of the model. In this sense, the approaches are complementary.

In comparison to all previous works of both groups, fault localisation in AI-enabled CPS faces new challenges, as stated in §1.2, including the lack of ground truth for DNN controllers, and the fact that a system execution involves a sequence of DNN controller inferences. These challenges make it impossible to apply the existing DNN fault localisation approaches in our context.

3.4.2 FAULT LOCALISATION OF CLASSIC CPSs

For classic CPS, there are several lines of work on their fault localisation. First, Liu et al. [95] present an iterative approach for localising faults in Simulink models by utilising decision trees; the approach uses decision trees to cluster similar failures, and adopts two selection criteria to select the best cluster for fault localisation. Liu et al. [96] propose a fault localisation approach based on dynamic model slicing and statistical debugging; dynamic slicing identifies, for each test case, the blocks that are executed; then, classical suspiciousness metrics are used to rank the blocks in terms of suspiciousness. Liu et al. [94] propose a search algorithm to generate small and diverse test suites for improving fault localisation accuracy.

Second, Bartocci et al. [18] integrate STL monitoring into CPS fault localisation to refine the results. Later, they [17] develop CPSDebug to detect failures in Simulink models. Recently, Bartocci et al. [19] propose a search-based method to generate a passing test case that closely resembles the original failing test, which can extract precise information about the location of faults in the system.

All these works target CPS models (e.g., Simulink) that contain classic CPS blocks only, and so they are not applicable to AI-enabled CPSs. Indeed, the uninterpretable decision logic of DNNs makes it difficult to determine which DNN components are responsible for an inference. In our work, we propose using forward impact as a measurement for the influence of each neuron

weights to the DNN inference.

3.5 REPAIR TECHNIQUES FOR DNNs AND CPSs

In this section, we review repair techniques for DNNs and CPSs.

3.5.1 DNN REPAIR

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

Adversarial training refers to the approaches that first collect adversarial examples and then use these examples with corrected labels to retrain the model. It has been extensively studied in repairing image classifiers, e.g., [55, 61, 106, 132, 153, 163]. While our work could be treated as a specific practice of adversarial training, we differ from existing works in the way we obtain the labels for the adversarial examples, namely, due to the high complexity of AI-enabled CPS, we search for the corrections for the control signals by optimization, such that safety is achieved at the system level.

Parameter alteration refers to the approaches that directly modify DNNs’ parameters to change the inference of adversarial examples [25, 133, 148, 167]. For instances, [133, 167] attempt to localize the suspicious DNN parameters and then optimize them for DNN repair.

3.5.2 REPAIR OF DNN CONTROLLERS

There have been several attempts to repair DNN controllers. Yang et al. [162] propose to first attack a DNN controller and then retrain with the identified adversarial examples. Majd et al. [107] presents a controller repair technique based on Mixed Integer Quadratic Program that encodes temporal logic requirements. However, compared to [162] and [107] in which an isolated

controller is considered, we target a combined system including a physical system controlled by a DNN controller and moreover, our safety requirement is defined based on states of the physical plant, rather than the DNN outputs. Another related work is [179], in which Zhou et al. repair a learning-based controller using an assumptive *safe* oracle. Compared to [179], our approach automates the controller repair process, by searching in the control decision space for corrections.

Safe reinforcement learning. *Reinforcement learning (RL)* is a typical approach for training controllers. In RL, there is a line of work [6, 9, 22, 180] that aims to ensure the safety of the trained controllers; in particular, *shielding* [6] provides a similar mechanism to our framework, which repairs the actions that trigger unsafe system behavior. However, there are several major differences between safe RL and our framework. First, many of the problem settings in RL are bounded to *Markov decision processes (MDP)*, which involve finitely many states and transitions; in contrast, we consider the control of non-linear continuous plants with dense-time real-valued dynamics, which are intrinsically hard to verify. Moreover, the safety mechanisms like shielding can be done at runtime—when an unsafe action is detected, a shield can be activated to stop it; in our case, banning a single action at runtime does not prevent unsafety from happening, due to the systems’ *dense-time* nature, and this can also explain why we analyze and repair complete signals in an offline manner.

3.5.3 CPS REPAIR

Different works have been proposed to test machine learning systems and systems empowered by machine learning components; please refer to [130, 169] for recent surveys. In this work, we focus on the activities that follow testing, specifically DNN repair.

DNN repair has been inspired by automatic program repair for classic code; we refer the interested reader to recent surveys [56, 86, 116]. Here we review related work on DNN repair.

A simple way of repairing the DNN consists of *retraining* it by using additional inputs. For example, Zhang et al. [166] introduced the approach *Apricot* which retrains the DNN model by

using different subsets of the training dataset and then merges them. Fahmy et al. [50] proposed the approach *HUDD*, a repair approach for DNN classifiers; the approach identifies the causes of failures by clustering the heatmaps of the behaviors of each DNN layer, and then uses the images that more likely lead to errors for retraining. Yu et al. [164] introduced *DeepRepair*, which aims to improve a DNN classifier against unknown noise that could be present in the operational environment; it learns unknown failure patterns and introduces them to retraining data via style transfer. Our approach differs from all the previous approaches, as it considers DNN controllers (and not classifiers), it is not based on retraining, and it does not require ground truth for the DNN behaviors.

An issue of retraining is that it modifies all the weights of the network, and so it may not be able to repair the behavior of the failing inputs without introducing other failures. To address this problem, a more popular approach consists of employing search-based approaches to modify a few DNN components, usually identified by some fault localization approach [45, 133]. For example, Sohn et al. [133] introduced *Arachne* that identifies the weights that are responsible for misclassification and then uses differential evolution to do the repair. Sun et al. [137] proposed a similar approach, but using PSO as a search algorithm. Li Calsi et al. [90] proposed *DistrRep* that extends *Arachne* by considering different misclassifications: it repairs each misclassification individually and then merges the different models into a final repaired model. Tokui et al. [141] proposed *NeuRecover* which focuses on the weights that were critical during the training history. Our approach shares with the previous approaches the use of a search-based approach for modifying the weights; however, differently from them, it uses system-level specifications to guide the repair.

A different family of repair approaches [82, 128, 129] considers different types of faults [71] at the architectural level (e.g., wrong choice of the optimizer). These approaches target completely different problems than ours, and so a direct comparison is difficult.

Some works have been proposed for repairing the CPS (i.e., the plant $\mathcal{M}$ of $\mathcal{M}^C$). For ex-

ample, Valle et al. [146] proposed a search-based approach for repairing the configuration of an elevator system. Jung et al. [78] repair misconfigurations of swarm robots. Ben Abdesslem et al. [20], instead, repair feature interaction bugs of autonomous driving systems. These works are complementary to ours, as they repair the physical plant, while we repair the controller. Future work could consider a combined repair of both.

4 | WHEN CYBER-PHYSICAL SYSTEMS MEET AI: A BENCHMARK, AN EVALUATION, AND A WAY FORWARD

In this chapter, we present a public benchmark of industry-level CPSs in seven domains and build AI controllers for them via state-of-the-art deep reinforcement learning (DRL) methods. Based on that, we further perform a systematic evaluation of these AI-enabled systems with their traditional counterparts to identify current challenges and future opportunities. Our key findings include (1) AI controllers do not always outperform traditional controllers, (2) existing CPS testing techniques (falsification, specifically) fall short of analysing AI-enabled CPSs, and (3) building a hybrid system that strategically combines and switches between AI controllers and traditional controllers can achieve better performance across different domains. Our results highlight the need for new testing techniques for AI-enabled CPSs and the need for more investigations into hybrid CPSs to achieve optimal performance and reliability.

4.1 INTRODUCTION

Cyber-Physical Systems (CPSs) refer to the combination of mechanical and computer systems, in which computer systems actively monitor and control the behaviour of mechanical systems

according to system states and external environments. The integration of computer systems significantly improves the performance of mechanical systems. Nowadays, CPSs have been widely deployed in diverse industrial domains, such as automotive systems, energy control, avionics, medical devices, *etc.*

With recent advances in Artificial Intelligence (AI), there has been increasing demand, from both industry and academia, in enhancing or even replacing traditional controllers with AI controllers (*e.g.*, deep neural networks), in order to achieve more optimized and flexible control. Given the great learning and generalization capabilities of deep neural networks, AI controllers have been increasingly deployed in various domains to handle complex situations in the physical world [97, 121].

Despite the rapid development of AI-enabled CPSs, there is a lack of comprehensive analysis of their performance (*e.g.*, safety, reliability, robustness) in different domains. The main reason is that unlike the large and growing open-source community in conventional software, CPSs are often kept as private intellectual properties by industrial practitioners, in which strong domain knowledge and know-how are encoded. Therefore, even up to the present, very few benchmarks of AI-enabled CPSs are publicly available for such analysis. Furthermore, since existing quality assurance methods such as *falsification* [4, 11, 34, 37, 161, 178] are mostly designed for traditional CPS controllers, it is unclear to what extent these methods are still effective in analysing AI controllers.

To bridge this gap, in this paper, we take the first step of benchmarking AI-enabled CPSs in multiple industrial domains and performing a systematic analysis of their performance in comparison to their traditional counterparts. Fig. 4.1 shows our high-level study workflow. In particular, we investigate three main research questions to identify the challenges and potential opportunities for building safe and reliable AI-enabled CPSs:

- **RQ1. How well do the DRL-based AI controllers perform compared with the traditional controllers?** This RQ aims to establish a comprehensive understanding of the advan-

tages and limitations of both traditional and AI controllers in CPSs. Our experiment shows that AI controllers do not always outperform their traditional counterparts. In several cases, AI controllers have weaknesses in handling multiple control outputs and fail to balance among multiple requirements.

- **RQ2: To what extent are existing CPS testing methods still effective on AI-enabled CPSs?** Falsification is a commonly used technique to detect defects in traditional CPSs. Although many falsification techniques have been proposed, it is still unclear whether they are still effective in the context of AI-enabled CPSs. This RQ aims to establish a testing benchmark of different falsification methods on both traditional and AI-enabled CPSs. Our comparative study finds that existing falsification methods are mostly designed for traditional controllers, and are not effective enough for AI-enabled CPSs.

- **RQ3: Can the combination of traditional and AI controllers bring better performance?** In practice, building a hybrid controller that strategically switches between AI controllers and traditional controllers can be a promising direction for better performance (*e.g.*, similar to component redundancies for safety in ISO 26262 [72]). This RQ aims to make an early investigation on whether this can be a promising direction in the context of AI-enabled CPSs. Overall, we find that among the three types of hybrid controllers we explored, a scenario-dependent approach outperforms the other two in most of the cases. This result confirms that strategically combining traditional and AI controllers is a promising direction for further research.

In summary, this work makes the following contributions:

- We create the first public dataset and benchmark of AI-enabled CPSs that span over various industrial domains, such as driving assistant, chemical reactor, aerospace, powertrain, *etc.* This provides a common ground for evaluating AI-enabled CPSs and also enables further research along this direction.
- We perform a systematic analysis of the performance (*e.g.*, safety, reliability, robustness) of AI-enabled CPSs, and benchmarks the current techniques (*e.g.*, falsification, system enhancement)

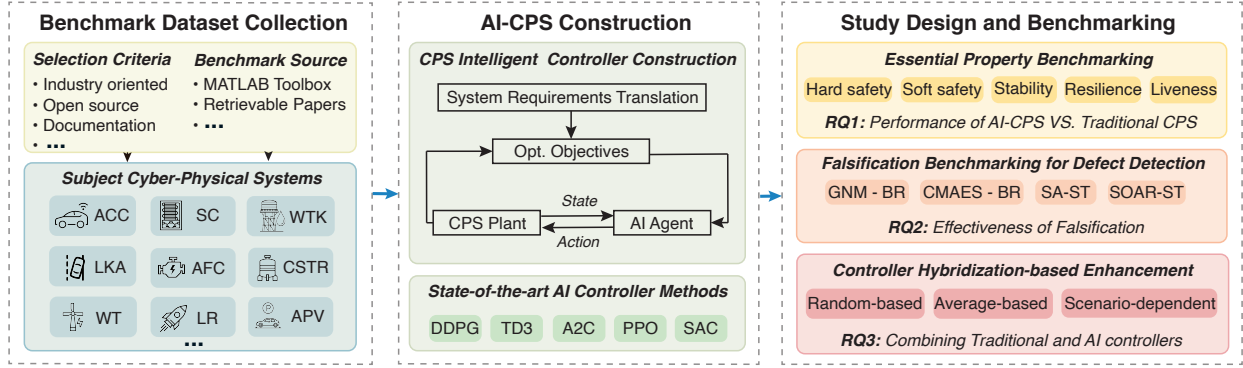


Figure 4.1: Workflow summary of AI-enabled CPSs dataset and benchmark construction, and high-level empirical study design.

as the basis for further investigation.

- Based on the analysis results, we further pose discussions on the future directions of AI-enabled CPSs, including effectiveness of AI controllers, testing tools for AI-enabled CPSs, and methods to construct hybrid control systems.

To the best of our knowledge, this is the very first paper that establishes a publicly available dataset and benchmark for industry-level CPSs with AI controllers. The benchmark and our empirical study results demonstrate the potential research opportunities around AI-enabled CPSs to meet the growing industrial demands. Our work enables better understanding, establishes the basis, and paves the path towards further quality assurance research to build safe and reliable AI-enabled CPSs.

4.2 BENCHMARK COLLECTION

4.2.1 BENCHMARK COLLECTION

As most industrial CPSs are treated as private intellectual properties and are kept confidential, it is challenging to find a large collection of practical and open-sourced CPSs. We focus on two sources that potentially release open-sourced systems with documentations: (1) the distribu-

Table 4.1: The collected subject CPSs

Subject CPS	Domain	Description	Type of Controller	#Blocks
Adaptive Cruise Control (ACC)	Driving assistant	Maintain a safety distance from a lead car	Model predictive control	297
Lane Keeping Assistant (LKA)	Driving assistant	Keep a vehicle in the center of a lane	Model predictive control	427
Automatic Parking Valet (APV)	Driving assistant	Park a vehicle on a target spot	Model predictive control	224
Exothermic Chemical Reactor (CSTR)	Chemical reactor	Promote a reactor transition of conversion rate	Model predictive control	316
Land a Rocket (LR)	Aerospace	Land a rocket on a target position	Model predictive control	175
Abstract Fuel Control (AFC)	Powertrain	Maintain the reference air-to-fuel ratio	PI & Feedforward control	281
Wind Turbine (WT)	Power grid	Maintain demanded values for key components	PID & Lookup table	161
Steam Condenser (SC)	Thermal systems	Maintain the reference pressure	PID control	56
Water Tank (WTK)	Water storage	Keep the water level at a reference value	PID control	919

tion of MATLAB control-related toolboxes, such as the *model predictive control* toolbox¹, and (2) CPS-related literature, such as the papers from software engineering and cyber-physical systems. Eventually, we selected nine industry-level CPSs in seven domains based on the following criteria (see Table 4.1).

- **Open-source.** A CPS must be open-sourced so that we can modify the system to replace the traditional controller in it with AI controllers.
- **Documentation.** A CPS must have comprehensive documentation so that we can configure the system properly. Furthermore, to train AI controllers using DRL, we need to understand the system requirements to design proper reward functions in DRL.
- **Complexity.** A CPS must reflect the industrial complexity in order to get useful insights and implications from the experiments.
- **Simulink.** A CPS must use Simulink as its modeling platform. Simulink is a MATLAB-based modeling environment developed by *Mathworks* and is widely adopted in industry. Having a consistent platform makes it easier to set up and run the benchmarks.

Table 4.1 gives an overview of the nine CPSs in the benchmark. Specifically, Column #Blocks shows the number of blocks in each system, which is often used to measure the complexity of a CPS. Each system ships with a built-in traditional controller. We experiment with different types of learning algorithm, various agent configurations, and diverse reward functions to explore the capability of AI controllers. Among all variants, we select the best one as the final AI controller

¹<https://www.mathworks.com/products/model-predictive-control.html>

for each system. Due to the page limit, we elaborate on three representative systems (of different types) and their traditional controllers and AI controllers in the following section.

4.2.2 THREE REPRESENTATIVE CPS EXAMPLES

Adaptive Cruise Control (ACC). ACC is released by Mathworks [111]. ACC is deployed in a driving environment with an ego car and a lead car. The goal of this system is to make the ego car move at a user-set velocity v_{set} as long as the relative distance d_{rel} between two cars is greater than the safety distance d_{safe} . The external input of the whole system is the acceleration of the lead car. The outputs include the velocity v_{ego} and the position of the ego car.

The traditional controller used in this system is *model predictive control (MPC)*. MPC achieves the optimal control at each moment by predicting the motions of the two cars in a finite time-horizon, and optimizing the acceleration of the ego car to maintain the safe distance. Specifically, MPC uses a linear model to predict the acceleration and velocity of both cars.

The DRL controller in ACC collects the system environment information to generate observation states. By evaluating the current state and computing a corresponding state value, the DRL controller outputs an acceleration command to the ego car. Unlike MPC, the DRL controller uses a reward function to evaluate the agent performance from two aspects: velocity and distance. While the safety distance d_{safe} is secured, the ego car should approach to the cruise velocity v_{set} . Otherwise, it follows the lead car velocity to avoid collision. The reward function penalizes the agent by the violation of the safety distance requirement and rewards the agent based on how close the ego car velocity v_{ego} is to the target speed.

Steam Condenser (SC). This system is collected from [159]. As an indispensable part of modern steam power plants, SC is a sealed container where the steam is condensed by cooling water. The goal of the system is to maintain the pressure of condenser P at a desired level P_{ref} . The external input of the system is the steam mass flow rate F_s (kg/s). The output of the system is the internal

pressure of the condenser.

The traditional controller used in SC is a PID controller which outputs a cooling water flowrate. A typical PID controller includes three parameters: proportional (P), integral (I) and Derivative (D). P reflects the current deviation of the system, I mirrors the accumulations of past errors and D represents on the anticipatory control. Through on-demand combination of the above three control parameters, the system error can be corrected.

The DRL controller in SC outputs a cooling water flowrate to maintain the condenser pressure at a desired level. It uses a reward function to make the condenser reach the desired pressure level as soon as possible and maintain this pressure until a new desired pressure is set. When the SC system moves to a steady state, the error signal may fluctuate at a small range. Then the reward function amplifies the error to improve the agent performance.

Abstract Fuel Control (AFC). AFC is a complex air-fuel control system released by Toyota [73]. The whole system takes two input signals from the outside environment, *PedalAngle* and *EngineSpeed*, and outputs $\mu = \frac{|AF - AF_{ref}|}{AF_{ref}}$, which is the deviation of the air-to-fuel ratio AF from a reference value AF_{ref} . The goal of this system is to control the deviation μ no more than a predefined threshold.

The original control system consists of two parts: (1) a PI controller, and (2) a feed-forward controller. The former regulates the air-to-fuel ratio AF in a closed loop, using the measured AF to compute the fuel command. The latter estimates the rate of air flow into the cylinder by measuring of the inlet air mass flow rate.

The DRL controller in AFC gathers the information about engine dynamics and outputs a fuel command to achieve the reference AF ratio. It uses a reward function to guide the agent to reduce the deviation μ . Specifically, a positive reward is given based on how smaller μ is and a negative feedback is generated if μ exceeds certain threshold. A small penalty is added based on the DRL action value from the last time step to acquire a stable control output.

Table 4.2: The STL specifications of the benchmark CPSs. S1 follows the pattern: $\Box_I(\varphi_1)$; S2 follows the pattern: $\Box_I(\varphi_2)$; S3 follows the pattern: $\Diamond_I\Box_{I'}(\varphi_3)$; S4 follows the pattern: $\Box_I(\varphi_{41} \rightarrow \Diamond_{I'}\varphi_{42})$; S5 follows the pattern: $\Diamond_I(\varphi_5)$;

Systems	S1-hard safety	S2-soft safety	S3-steady state	S4-resilience	S5-liveness
ACC	$I = [0, 50]$ $\varphi_1 \equiv d_{rel} \geq d_{safe}$	$I = [0, 50]$ $\varphi_2 \equiv v_{ego} \leq v_{set}$	$I = [0, 50], I' = [0, 40]$ $\varphi_3 \equiv d_{rel} \geq d_{safe} + 0.2$	$I = [0, 50], I' = [0, 1]$ $\varphi_{41} \equiv \neg\varphi_3, \varphi_{42} \equiv \varphi_3$	$I = [0, 50]$ $\varphi_5 \equiv v_{ego} \geq 1$
LKA	$I = [0, 15]$ $\varphi_1 \equiv error_1 \leq 0.85$	$I = [0, 15], \varphi_2 \equiv \varphi_{21} \wedge \varphi_{22}$ $\varphi_{22} \equiv error_1 = 0$ $\varphi_{21} \equiv error_2 = 0$	$I = [0, 15], I' = [0, 10]$ $\varphi_3 \equiv error_1 \leq 0.5$	$I = [0, 15], I' = [0, 1]$ $\varphi_{41} \equiv \neg\varphi_3, \varphi_{42} \equiv \varphi_3$	$I = [0, 15]$ $\varphi_5 \equiv v \geq 1$
APV	$I = [0, 12]$ $\varphi_1 \equiv error_1 \leq 1$	$I = [0, 12], \varphi_2 \equiv \varphi_{21} \wedge \varphi_{22}$ $\varphi_{21} \equiv error_1 = 0$ $\varphi_{22} \equiv error_2 = 0$	$I = [0, 12], I' = [0, 10]$ $\varphi_3 \equiv error_1 \leq 0.5$	$I = [0, 12], I' = [0, 1]$ $\varphi_{41} \equiv \neg\varphi_3, \varphi_{42} \equiv \varphi_3$	$I = [0, 12]$ $\varphi_5 \equiv v \geq 0.1$
CSTR	$I = [25, 30]$ $\varphi_1 \equiv error \leq 0.5$	$I = [25, 30]$ $\varphi_2 \equiv error = 0$	$I = [25, 30], I' = [0, 4]$ $\varphi_3 \equiv error \leq 0.4$	—	—
LR	$I = [14.8, 15]$ $\varphi_1 \equiv error \leq 0.5$	$I = [14.8, 15]$ $\varphi_2 \equiv error = 0$	—	—	$I = [0, 15]$ $\varphi_5 \equiv v \geq 0.1$
AFC	$I = [0, 30]$ $\varphi_1 \equiv \mu \leq 0.2$	$I = [0, 30]$ $\varphi_2 \equiv \mu = 0$	$I = [0, 30], I' = [0, 20]$ $\varphi_3 \equiv \mu \leq 0.1$	$I = [0, 30], I' = [0, 1]$ $\varphi_{41} \equiv \neg\varphi_3, \varphi_{42} \equiv \varphi_3$	—
WT	$I = [30, 630]$ $\varphi_1 \equiv \theta \leq 14.2$	$I = [30, 630]$ $\varphi_2 \equiv \theta = 0$	$I = [30, 630], I' = [0, 500]$ $\varphi_3 \equiv \theta \leq 14$	$I = [30, 630], I' = [0, 5]$ $\varphi_{41} \equiv \neg\varphi_3, \varphi_{42} \equiv \varphi_3$	—
SC	$I = [30, 35]$ $\varphi_1 \equiv error \leq 0.5$	$I = [30, 35]$ $\varphi_2 \equiv error = 0$	$I = [30, 35], I' = [0, 4]$ $\varphi_3 \equiv error \leq 0.4$	—	—
WTK	$I = [5, 6] \cup [11, 12] \cup [17, 18]$ $\varphi_1 \equiv error \leq 0.2$	$I = [5, 6] \cup [11, 12] \cup [17, 18]$ $\varphi_2 \equiv error = 0$	$I = [5, 6] \cup [11, 12] \cup [17, 18]$ $I' = [0, 0.8]$ $\varphi_3 \equiv error \leq 0.15$	—	—

4.3 EXPERIMENT DESIGN

As illustrated in Fig. 4.1, we perform experiments to answer three research questions. In RQ1, we evaluate the performance of AI-enabled CPSs vs. traditional CPSs. In RQ2, we evaluate the effectiveness of the existing falsification methods. In RQ3, we investigate the possibility of combining traditional and AI controllers. All experiments are based on system specifications from the official documentation (§4.3.1). These specifications further derive different metrics and experiment settings for each RQ, which we detail in §4.3.2.

4.3.1 SYSTEM SPECIFICATIONS

In this work, we adopt STL (introduced in §2.2) as our specification language to evaluate the temporal properties of CPSs. Specifically, we extract the temporal properties of the models from

their documents, and summarize them in Table 4.2. We classify these properties into 5 categories, namely **S1-S5**, according to their semantics:

- **S1: *Hard safety*.** S1 must be strictly satisfied by the systems, any violation of S1 can lead to safety problems. S1 follows the pattern $\Box_I(\varphi_1)$, where φ_1 is a system invariant during the simulation.
- **S2: *Soft safety*.** S2 follows the similar pattern $\Box_I(\varphi_2)$ as S1, but the satisfaction of φ_2 is not demanded. Instead, S2 is used to measure the average and maximum deviations of the outputs from the reference values. Based on that, we can understand the average and boundary behaviors of the controllers.
- **S3: *Steady state*.** S3 follows the pattern $\Diamond_I\Box_{I'}(\varphi_3)$. It requires the system to satisfy φ_3 for the interval I' at some point in I . S3 is used to evaluate if the system reaches the steady state φ_3 and stays there for the interval I' .
- **S4: *Resilience*.** S4 follows the pattern $\Box_I(\varphi_{41} \rightarrow \Diamond_{I'}\varphi_{42})$. It requires that, during the interval I , whenever an event φ_{41} happens, the system should react by satisfying φ_{42} within I' . S4 is used to evaluate the system's ability to recover from fluctuations.
- **S5: *Liveness*.** S5 follows the pattern $\Diamond_I(\varphi_5)$, where φ_5 should be eventually satisfied during I . S5 is used to inspect the system to avoid the case when the controller makes conservative control decisions to passively meet the safety requirements.

4.3.2 RESEARCH QUESTIONS

RQ1. How well do the DRL-based AI controllers perform compared with the traditional controllers?

While many AI controllers have been proposed and used, there has not been a systematic study on how well AI controllers perform compared with traditional controllers. RQ1 aims to compare the performance of these two kinds of controllers and understand their *pros and cons*. Since DRL-based approaches are the state of the art among current AI controllers, we mainly

focus on DRL-based controllers in this study. In the experiment, we first randomly generate 100 input signals. Then we run simulations on each CPS with the traditional controller and the DRL-based controller respectively. We compare the performances of these controllers according to multiple properties introduced in §4.3.1. To better understand the quality of the controllers, we not only consider Boolean satisfactions to the STL formulas in §4.3.1, but also propose a series of more fine-grained metrics that consider the semantics of different formula patterns. These fine-grained metrics for **S1-S5** are listed as follows:

- **S1:** We record the number of satisfying simulations n_s and compute the satisfaction ratio over 100 rounds of simulation since S1 are safety properties required to be satisfied strictly.
- **S2:** In addition to the Boolean satisfaction of S2, we are also interested in how much S2 is violated in each case. Hence, given an output signal s , we first collect all the moments T when φ_2 is violated. Then, we present two metrics regarding S2: 1) **MAE**—the mean value of the average absolute error over 100 simulations, and 2) **MAXERR**—the mean value of the maximum absolute error over 100 simulations. With these two metrics, we can understand the average and the extreme violations, respectively.
- **S3:** We use S3 to extensively explore how stable the system is during each round of simulation. For 100 simulations, we record how many moments T are there when $s(t)$ falls into a steady state defined by φ_3 . Then we compute the percentage of these steady state moments by $\frac{\#T}{\#T_{total}}$ in each round of simulation. We take the average value over 100 simulations as the fine-grained metric to evaluate S3.
- **S4:** We use $\frac{n_r}{n_v}$ to measure how good the systems satisfy S4 during one round of simulation, where n_v is the number of cases when the system falls into the fluctuating state defined by φ_{41} , and n_r is the number of the cases when the system can react by doing φ_{42} . We then take the average value of $\frac{n_r}{n_v}$ over 100 simulations.
- **S5:** We record the number of simulation rounds n_s satisfying S5 in §4.3.1. Then we calculate the ratio of satisfaction over 100 rounds of simulation.

By analysing the evaluation results with the metrics above, we can obtain more comprehensive information to understand the advantage and limitations of DRL-based AI controllers and traditional controllers, from multiple perspectives.

RQ2. To what extent are existing CPS testing methods still effective on AI-enabled CPSs?

For RQ2, we focus on an established CPS testing methodology called falsification (introduced in § 2.3). Falsification has proved to be effective on traditional CPSs [4, 161, 178], but few studies have evaluated it on AI controllers. Since falsification methods are guided by logic semantics, it is unclear whether it is still effective on AI controllers, which are essentially statistical methods with high uncertainty and low interpretability.

In the experiment, we select two widely-used falsification tools, Breach [34] and S-TaLiRo [11]. Note that both Breach and S-TaLiRo integrate several different back-end optimization solvers, and apply different tricks to improve the performance. We select Global Nelder-Mead (GNM) and CMAES for Breach, and Simulated Annealing (SA) and stochastic optimization with adaptive restart (SOAR) [110] for S-TaLiRo according to the findings in [47].

Due to the stochasticity of the falsification algorithms, for each experiment, we repeatedly run the falsification algorithms, and obtain a falsification rate $\frac{\# \text{successful trials}}{\# \text{total trials}}$ as the evaluation metric. We also report the average time consumption, and the average number of successful falsification trials as complementary metrics.

In this experiment, we only run falsification on the systems that never violate S1 (hard safety) during random simulations in the experiment of RQ1. Because these systems are hard to be evaluated in RQ1, which can be used as the changeable samples to measure the performance difference among different falsification approaches.

Our results of RQ1 and RQ2 form a basic benchmark quality and reliability analysis of AI-enabled CPSs.

RQ3. Can the combination of traditional and AI controllers bring better performance?

As suggested by international standards *ISO26262* [72] and *ISO/PAS 21448 (SOTIF)* [1], modular

redundancy (e.g., doubling or tripling) is an important way to improve system quality and reliability. Therefore, in RQ3, we aim to investigate the performance of hybrid controllers—a novel type of controllers which combines the traditional and DRL-based ones. We perform an exploration on three typical approaches for such combinations: (1) a random-based approach, (2) an average-based approach, and (3) a scenario-dependent approach, with the purpose to understand whether this could be a promising direction for further research.

- ***Random-based.*** This method chooses a signal randomly from two controllers and passes it to the subsequent components. Here, *sample time* is a hyperparameter that indicates how frequently the controller switches. We use 2 different sample times in our experiments: (1) 0.1 sec, which is the minimum system step time, and (2) 1 sec, which is a log scale increase for comparison.
- ***Average-based.*** This method takes the average value of the outputs of two controllers as the final output.
- ***Scenario-dependent.*** Based on the insights from the experiments of RQ1 and RQ2, we summarize the scenarios where different types of controllers perform well and design a dynamic controller switch logic to achieve the optimal control strategy. For instance, for a system with controller A and B, A may have smaller value on S2 (averaged error signal) while B may have better performance on S4 (resilience). A switch logic can be: if the error signal exceeds a specified threshold which indicates the system falls into an unsteady state, the control will be granted to B to quickly recover to the steady state, and A may take in charge to maintain a small error value under the steady state.

In RQ3, we use the same metrics from RQ1 and RQ2 to evaluate these hybrid controllers under the same experimental settings.

Hardware & Software Dependencies. DRL training is computationally intensive, so we use a server with 3.5GHz Intel i9-10920X CPU, 15GB RAM, and an NVIDIA TITAN V GPU. Other experiments (e.g., falsification) were conducted using Breach 1.9.0 with GNM and CMAES as

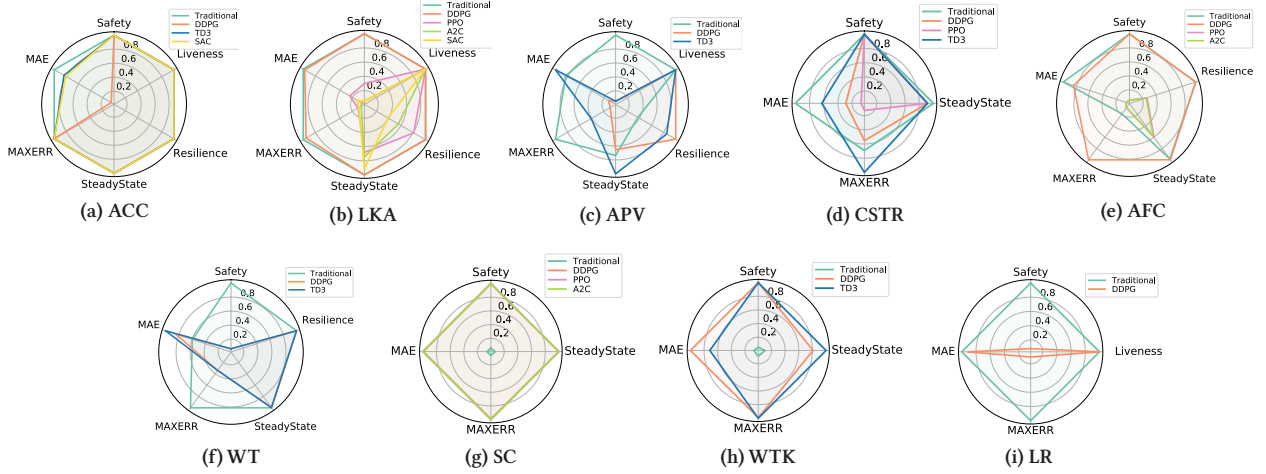


Figure 4.2: Performance comparison on systems with traditional and DRL controllers (RQ1)

solvers and S-TaLiRo 1.6.0 with SA and SOAR as solvers on an Amazon EC2 c4.xlarge server with 2.9GHz Intel Xeon E5-2666 CPU, 4 virtual CPU cores, and 8GB RAM.

4.4 EXPERIMENTAL RESULTS

4.4.1 RQ1. PERFORMANCE OF AI-ENABLED CPSS VS. TRADITIONAL CPSS

Fig. 4.2 summarizes the evaluation results of traditional and AI-enabled CPSSs via radar charts. Each axis in a radar chart represents an evaluation metric from §4.3.1. We normalize the result of each metric into $[0, 1]$. A higher value indicates a better performance.

- *S1: Hard safety.* Not surprisingly, traditional controllers have better or at least comparable performance on S1 in 7 of the 9 systems (ACC, LKA, APV, CSTR, LR, AFC, and WT), compared to DRL controllers. This is because safety always has the highest priority in the design of a traditional controller. However, in WTK and SC, the traditional PID controllers fail to regulate the error signals within a limited range. In particular, an oscillated signal can be generated by PID controllers and the maximum error threshold is therefore violated by an instantaneous overshoot.

DRL controllers also have good performance on S1 in 6 systems (ACC, LKA, CSTR, AFC, SC, and WTK). However, DRL controllers are better than traditional controllers in SC and WTK, while worse in WT and APV. Unlike the behaviour of PID controllers in SC and WTK, DRL controllers output stable control signals with no instant overshoot. Thus, the safety requirements are held. Overall, for S1, traditional controllers and DRL controllers have similar performance in systems like ACC, LKA, CSTR, and AFC. Also, for all systems, there is always at least one controller that never violates S1. These results indicate that, the random sampling method cannot effectively evaluate systems on S1, and therefore, we use the more advanced testing method, namely falsification, in RQ2 to acquire more in-depth insights.

- *S2: Soft safety.* The traditional controllers have good MAE performance in 5 of the 9 systems (ACC, LKA, CSTR,LR, and AFC) and decent MAXERR performance in 5 systems (ACC, LKA, APV,LR, and WT). Compared with the results on S1, there are fewer traditional CPSs that outperform AI-enabled CPSs on S2. Moreover, a traditional controller with a good MAE result may not have a similarly good MAXERR result, *e.g.*, AFC and CSTR. This indicates that the output signals from traditional controllers could be unstable, compared with DRL controllers.

DRL controllers bring good MAE results in 7 systems (ACC, LKA, APV, WT, SC, LR, and WTK) and good MAXERR results in 6 systems (ACC, LKA, CSTR, AFC, SC, and WTK). According to these results, if a system has too many performance requirements or multiple control outputs, such as APV and WT, a standalone DRL controller may not handle these situations well.

- *S3: Steady state.* Most traditional controllers can provide relatively stable outputs in all systems except SC and WTK. The poor performances of these two systems can be attributed to their oscillated PID outputs. In contrast, DRL controllers show good performance in all of the systems. Moreover, we find that different DRL controllers of the same system may have a variance in their performance. For instance, in WTK, while TD3 does not perform as well as DDPG in MAE, it outperforms DDPG in S3.
- *S4: Resilience.* Both DRL and traditional controllers show good performances regarding re-

Table 4.3: Falsification performance comparison between four existing falsification algorithms (RQ2)

	ACC-T			ACC-DDPG			ACC-TD3			ACC-SAC			CSTR-T			CSTR-DDPG			CSTR-PPO			CSTR-TD3		
	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim
GNM-BR	3	269.9	187.7	1	250.6	144.0	25	162.6	107.9	1	313.8	159.0	28	292.0	56.7	0	-	-	0	-	-	0	-	-
CMAES-BR	16	142.4	101.5	0	-	-	16	76.9	30.8	9	263.8	98.9	30	190.8	35.8	0	-	-	0	-	-	0	-	-
SA-ST	2	148.1	148.0	1	218.6	147.0	0	-	-	1	433.7	272.0	30	200.29	47.57	0	-	-	0	-	-	0	-	-
SOAR-ST	0	-	-	0	-	-	0	-	-	0	-	-	30	214.0	50.5	0	-	-	0	-	-	0	-	-

	AFC-T			AFC-DDPG			AFC-PPO			AFC-A2C			SC-T			SC-DDPG			SC-PPO			SC-A2C		
	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim
GNM-BR	0	-	-	0	-	-	30	9.5	6.8	30	9.4	6.1	30	0.3	1.0	0	-	-	0	-	-	0	-	-
CMAES-BR	0	-	-	0	-	-	10	37.1	28.9	4	77.3	61.3	30	0.3	1.0	0	-	-	0	-	-	0	-	-
SA-ST	9	295.2	148.4	0	-	-	0	-	-	0	-	-	30	0.2	1.0	0	-	-	0	-	-	0	-	-
SOAR-ST	11	411.3	157.4	0	-	-	0	-	-	0	-	-	30	0.2	1.0	0	-	-	0	-	-	0	-	-

	LKA-T			LKA-DDPG			LKA-PPO			LKA-A2C			LKA-SAC			APV-T			APV-DDPG			APV-TD3		
	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim
GNM-BR	30	636.9	75.4	0	-	-	30	10.6	10.9	30	5.0	5.0	30	60.5	54.0	0	-	-	30	41.5	1.0	30	42.8	1.0
CMAES-BR	27	414.9	47.8	0	-	-	28	21.6	16.6	30	9.4	7.5	28	82.3	44.4	0	-	-	30	51.6	1.0	30	59.7	1.0
SA-ST	24	1006.2	114.8	0	-	-	29	28.4	33.0	29	62.8	64.0	30	20.7	25.1	0	-	-	30	112.5	1.0	30	119.2	1.0
SOAR-ST	30	262.4	30.8	0	-	-	30	5.2	7.1	30	3.3	5.2	30	50.2	27.6	0	-	-	30	115.9	1.0	30	126.2	1.0

	WT-T			WT-DDPG			WT-PPO			WTK-T			WTK-DDPG			WTK-TD3			LR-T			LR-DDPG		
	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim
GNM-BR	30	197.8	42.9	0	-	-	0	-	-	30	3.3	5.8	0	-	-	0	-	-	20	4224.7	80.8	30	68.5	1.0
CMAES-BR	0	-	-	0	-	-	0	-	-	30	9.8	17.6	0	-	-	0	-	-	16	4341.1	78.4	30	54.9	1.0
SA-ST	20	335.5	65.4	0	-	-	0	-	-	30	8.8	20.6	0	-	-	0	-	-	13	6532.2	92.9	30	78.4	1.0
SOAR-ST	30	255.7	50.9	0	-	-	0	-	-	30	4.0	9.1	0	-	-	0	-	-	9	7809.1	93.2	30	83.7	1.0

silience in the 5 applicable systems, except the traditional and TD3 controllers in APV. This reveals their weaknesses in handling fluctuations. In contrast, although DDPG performs strongly in resilience, it does not conform to S1, the hard safety requirements. For that reason, it can not be considered as a reliable system.

- *S5: Liveness.* All the controllers show positive results on S5. This indicates that none of our controllers takes conservative strategies to passively satisfy the safety requirements.

Regarding the comparison between the traditional controller and DRL controllers in each system, DRL controllers have better or similar performance in most of the systems. However, this is not always the case. For example, in APV and WT, the DRL-based controllers do not perform as well as their traditional counterparts.

Answer to RQ1: Though AI controllers achieve better or comparable performance in many cases, they also fail to properly function in several complex cases, implying that falling back to traditional controllers can be a safer choice in some complex scenarios that involve multiple control requirements.

4.4.2 RQ2. EFFECTIVENESS OF FALSIFICATION

Table 4.3 shows the experimental results of RQ2, where we evaluate the performances of 4 falsification approaches using the metrics of FR (/30), time (secs), and #sim (see §4.3.2). For each benchmark, we select their S1, namely hard safety, as the target system specification. The reason is that, according to RQ1, we know that most of the systems do not violate their hard safety properties. Therefore, it makes sense to further investigate the satisfaction of each system to S1. In our experiments, we set the budget (B_G in Alg. 1) as 300. We highlight the best performer for each CPS model in Table 4.3, according to their FR.

The results presented in Table 4.3 reveal apparent differences in the abilities of different falsification algorithms for specific cases. For example, CMAES-BR performs well in ACC-T, but does not perform well in WT-T or AFC-A2C; SOAR-ST performs well in AFC-T, but does not perform well in ACC-TD3 or AFC-A2C.

We identify a model as *falsifiable* once there exists a falsification algorithm that manages to falsify the model. Based on our observations, we find that, for any falsification algorithm A, there always exists a falsifiable benchmark that can not be falsified by A. For instance, GNM-BR cannot falsify the falsifiable model AFC-T, and performs poorly on ACC-SAC; CMAES-BR cannot falsify the falsifiable model ACC-DDPG or WT-T; SA-ST cannot falsify the falsifiable model AFC-PP0 or AFC-A2C; SOAR-ST cannot falsify the falsifiable model AFC-PP0 or AFC-A2C. This result proves the famous *no free lunch theorem* in optimization [152], and also motivates us to develop more effective algorithms to handle these emerging AI-enabled CPSs.

Answer to RQ2: Falsification, an established testing method, fails to handle several AI-enabled CPSs, highlighting the opportunity to develop new AI-aware testing methods for CPSs.

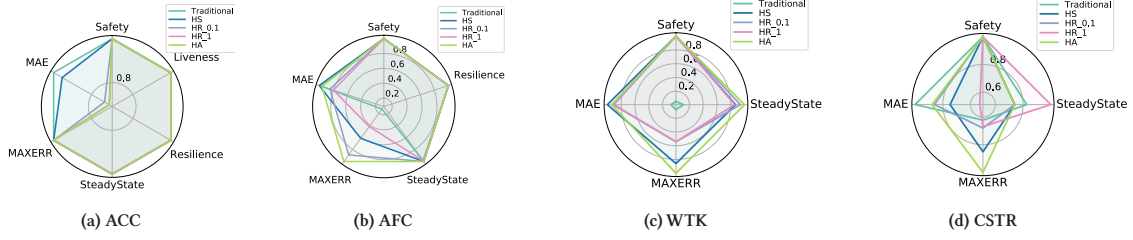


Figure 4.3: Performance comparison on systems with hybrid controllers (RQ3)

Table 4.4: Performance comparison on 4 hybrid benchmark CPSs with falsification tools (RQ3)

Tool	GNM-BR			CMAES-BR			SA-ST			SOAR-ST		
Benchmark	FR	time	#sim	FR	time	#sim	FR	time	#sim	FR	time	#sim
ACC-HS	0	-	-	11	177.9	71.5	2	465.4	188.0	0	-	-
ACC-HR_0.1	1	123.9	48.0	0	-	-	1	392.1	183.0	0	-	-
ACC-HR_1	3	313.1	121.0	1	66.2	26.0	1	457.6	209.0	0	-	-
ACC-HA	4	437.1	165.0	1	178.9	72.0	1	372.3	149.0	0	-	-
AFC-HS	0	-	-	0	-	-	0	-	-	0	-	-
AFC-HR_0.1	0	-	-	0	-	-	0	-	-	0	-	-
AFC-HR_1	29	148.5	87.4	30	105.1	61.0	0	-	-	0	-	-
AFC-HA	0	-	-	0	-	-	0	-	-	0	-	-
WTK-HS	0	-	-	0	-	-	0	-	-	0	-	-
WTK-HR_0.1	0	-	-	0	-	-	0	-	-	0	-	-
WTK-HR_1	0	-	-	0	-	-	0	-	-	0	-	-
WTK-HA	0	-	-	0	-	-	0	-	-	0	-	-
CSTR-HS	3	835.8	155.0	1	1310.6	292.0	10	997.7	213.4	11	973.5	172.1
CSTR-HR_0.1	0	-	-	0	-	-	0	-	-	0	-	-
CSTR-HR_1	22	411.2	87.6	20	155.8	33.2	18	578.7	124.0	30	369.0	61.9
CSTR-HA	0	-	-	0	-	-	0	-	-	1	1940.4	189.0

4.4.3 RQ3. PERFORMANCE OF HYBRID CONTROLLERS

From the results of RQ1 and RQ2, we find that traditional and AI controllers have their own advantages in satisfying different requirements. For example, in CSTR (Fig. 3d), although the traditional controller performs well in MAE, it does not perform well in MAXERR. In contrast, the TD3 controller complements the situation, holding a high MAXERR but low MAE. This inspires us to combine traditional and AI controllers to obtain hybrid controllers, which may outperform each of them alone. We explore three combination methods, namely the random-based, the average-

based, and the scenario-dependent ones, introduced in §4.3.2. For the random-based methods, we evaluate on multiple instances varied by their sampling time. Our results include Fig. 4.3 that uses the same evaluation metrics as RQ1, and Table 4.4 that applies the 4 falsification methods to those hybrid controllers.

- ACC. From the Fig. 4.3 and Table 4.4, in ACC, all the hybrid controllers perform better in falsification than the traditional controllers, and the scenario-dependent hybrid controller performs well in MAE compared to the traditional controller. In contrast, the performances of the random-based or the average-based hybrid controllers are not as good as the scenario-dependent ones.
- AFC. Among the 4 types of hybrid controllers we deployed, 3 of them have similar or better performance than the traditional one.
- WTK. The scenario-dependent and the average-based hybrid controllers are significantly better than the random-based controllers, and none of the hybrid controllers have been falsified.
- CSTR. We find that, the random-based method controller HR_0.1 and the average-based controller are not falsified. Moreover, the average-based controller also performs well in MAXERR as its constituent TD3 controller does.

In summary, the hybrid controllers can take advantage of their constituent controllers. The results reflect that the random-based combination method with large sample time can lead to an inconsistent control logic, and it can cause serious safety issues. The average-based method can be applied if the system is too complicated to design a controller switching logic and/or the constituent controllers have distinct advantages in different aspects. The scenario-dependent approach is recommended as it can switch between different controllers based on real-time system states. Moreover, the controller switching logic should be exclusively designed to fit the characteristics of the candidate controllers.

Answer to RQ3: Building a hybrid controller which strategically switches between AI controllers and traditional controllers is a promising direction and can significantly improve the performance in our evaluated cases. For three types of hybrid controllers we explored, a scenario-dependent approach outperforms the other two in most of the cases.

4.5 DISCUSSIONS, FUTURE DIRECTIONS AND THREATS TO VALIDITY

Discussions. According to our insights from RQ1, DRL controllers may fall short of handling multiple control outputs or balancing multiple requirements simultaneously, compared to their traditional counterparts. Indeed, the design of reward function may be too complicated to compensate the balance among different requirements, since rewarding only a portion of the requirements may overshadow others. In industry, it is a common scenario to handle multiple requirements simultaneously. Therefore, applying DRL-based AI controllers in these cases requires further research.

Based on our evaluation of existing falsification algorithms, we find that falsification may not work effectively for AI-enabled CPSs. AI controllers have their specific structure and unique decision logic, which is quite different from their traditional counterparts. Therefore, taking into account the characteristics of this specific formalism is important for effective testing.

The combination of different types of controllers offers a new direction of improving the safety and performance of the controllers, as demonstrated by our evaluation. Specifically, the scenario-dependent approach outperforms others, showing that strategic combination is necessary to achieve superior performances.

Future Directions. Based on the insights from our evaluation, we propose the following three future directions for AI-enabled CPSs.

- First, there is a need for more research efforts on benchmarks and empirical studies in this direction. Moreover, more complicated system requirements that reflect industrial standards

or demands should also be considered for further evaluation;

- Second, according to RQ2, existing falsification tools are not fully effective in detecting requirement violations in AI-enabled CPSs. This offers a research opportunity of developing more advanced testing techniques for AI-enabled CPSs, e.g., by exploiting the specific structure of neural networks for more effective testing;
- Third, besides testing, analysis techniques should be developed to understand the root cause of the violations. To achieve this, more research efforts on fault localization and repair are necessary.

Threats to Validity. In terms of *construct validity*, one potential threat is that the evaluation metrics may not fully describe the performance of controllers. To mitigate this threat, we used five evaluation metrics and two falsification tools to comprehensively measure and analyze the performance and reliability of CPSs in our benchmark. In terms of *internal validity*, one potential threat is that the behaviour of a CPS can vary when using different environment parameters. To mitigate this threat, we chose to use the same parameters as described in the documentation of each CPS to keep consistency. Further, we confirmed that our simulation results are consistent with the source descriptions and demos. In terms of *external validity*, one potential threat is that our analysis results may not be generalized to other CPSs. To mitigate this threat, we tried our best to collect a diverse set of CPSs with different functionalities, system environments, and control tasks.

4.6 CONCLUSION

This paper presents a public benchmark of AI-enabled CPSs in various domains, which can serve as a fundamental evaluation and testing framework for enhancing the understanding and development of AI-enabled CPSs. Based on this benchmark, we collected a series of evaluation metrics and measured the performance and reliability of state-of-the-art deep reinforcement

learning (DRL) controllers on various types of CPSs. Our findings reveal some strengths and weaknesses of AI-enabled CPSs and highlights an opportunity of strategically combining AI-enabled CPSs with traditional CPSs. Furthermore, our analysis of two widely used falsification techniques on AI-enabled CPSs motivates further improvement of these techniques to account for the unique characteristics of AI controllers, in order to build safe and reliable CPSs in the age of AI.

5 | FalsifAI: FALSIFICATION OF AI-ENABLED HYBRID CONTROL SYSTEMS GUIDED BY TIME-AWARE COVERAGE CRITERIA

5.1 INTRODUCTION

Modern Cyber-Physical Systems (CPSs) are constantly increasing their complexity and capabilities. Systems like autonomous vehicles [23] or air traffic collision avoidance systems [77] perform extremely complex control tasks that require advanced techniques such as intelligent autonomy. To this aim, they are increasingly empowered by AI components in practice, mainly based on deep neural networks (DNNs); we refer to this class of systems as *AI-Enabled Hybrid Control Systems* [69, 74, 75, 143, 144]. In this type of systems, the neural network controls, in a feedback loop manner, a plant (e.g., an autonomous vehicle), i.e., it takes as input the state of the plant and produces as output the control command for it. At the same time, the neural network is also fed with continuous input signals from the environment.

Safety assurance of these systems is of paramount importance, since many of these are applied under safety-critical scenarios. For classical CPSs, *falsification* is the state-of-the-art validation

method [33, 113, 161, 177], with several industrial applications [60, 118]. Given a temporal specification, usually given in *Signal Temporal Logic* (STL), falsification consists in seeking to find a temporal signal to the system that violates the specification. To do this, it exploits the quantitative *robust semantics* of temporal formulas [35, 51]. Robust semantics extends the classical Boolean satisfaction relation in the following sense: given a formal specification φ and an input signal $\mathbf{u}$, it tells not only whether φ is satisfied or violated, but also *how robustly* the formula is satisfied or violated. Falsification is usually cast as an optimization approach that generates input signals with the aim of minimizing robustness. The approach terminates when an input with negative robustness, i.e., a *falsifying input*, is found, witnessing the violation of the specification. Up to the present, different optimization-based falsification methods have been proposed (see [79] for a survey), and some well-known established tools (e.g., Breach [33] and S-TaLiRo [10]) have also been developed.

However, most of the current falsification approaches treat the system as a black box and guidance for the falsification search is only provided by robustness [33, 113, 161, 177], or coverage of the input signal space [4, 38]. Pure guidance by robustness is too greedy, as it only focuses on *exploitation* [30] (i.e., the detailed search in a specific area of the search space), and can be easily trapped in local optima. On the other hand, only considering the coverage of the huge input signal space enables more *exploration* [30] (i.e., the visit of different areas of the search space) to some extent, but may not scale to large systems; indeed, different inputs may lead to the same system behaviours, and it is challenging to efficiently find different inputs exhibiting as diverse system behaviours (i.e., the response of the system to the environmental conditions) as possible, being these infinite.

These problems are further exacerbated when considering falsification of AI-Enabled Hybrid Control Systems. In such systems, the neural network controller is the central component responsible for the system evolution, and the different types of behaviours that can be exhibited by the system depend on the *sequences of decisions* of the neural network (i.e., the control commands

provided to the plant). Hence, in order to obtain more diverse exploration in the falsification search, we need to seek diverse signals that can trigger all the possible types of sequences of control decisions of the neural network over time.

In traditional software testing, coverage criteria are the classical and *de facto* means to try to obtain a sufficient exploration of diversity of software behaviours [8]. For DNNs, some coverage criteria also have been recently proposed [40, 81, 104, 105, 126, 138, 168] that consider the neuron activation status coverage for a specific set of inputs. In our context of CPS falsification under temporal signals, these criteria can be promising for assessing whether a particular type of decision (i.e., a control command) has been taken by the DNN at a particular timestamp; however, they are not able to quantitatively measure the sufficiency on how many diverse behaviours of the DNN (controlling the plant over time) are observed. Let us consider the example of an Adaptive Cruise Control (ACC) system governed by a DNN. The scenario in which the DNN provides a single command of acceleration at the beginning, is very different from the scenario in which the DNN keeps on sending the same acceleration command over time. For classical DNN coverage criteria, the inputs used in these two scenarios could obtain the same level of coverage.¹

Therefore, in order to better account for the influence of the neural network controller on the plant over time, it is highly desirable to design “time-aware” coverage criteria of DNN controllers to capture the diverse DNN internal behaviours over time. To this aim, we originally propose three categories of time-aware coverage criteria (for a total of eight criteria) that consider different aspects of the *temporal behaviour* of the DNN (i.e., due to the repeated execution of the DNN in the AI-Enabled hybrid control system): 1) the activation of neurons *at a particular timestamp*, 2) the *continuous* activation of neurons over a specific time window, 3) and the *change over time* (differential behaviour) of the activation of neurons. Such criteria enable to better characterize the behaviours of a given DNN controller and, consequently, also the behaviour of the AI-enabled

¹Note that we consider feed-forward networks in this work; these observations do not in general hold for other networks such as recurrent neural networks.

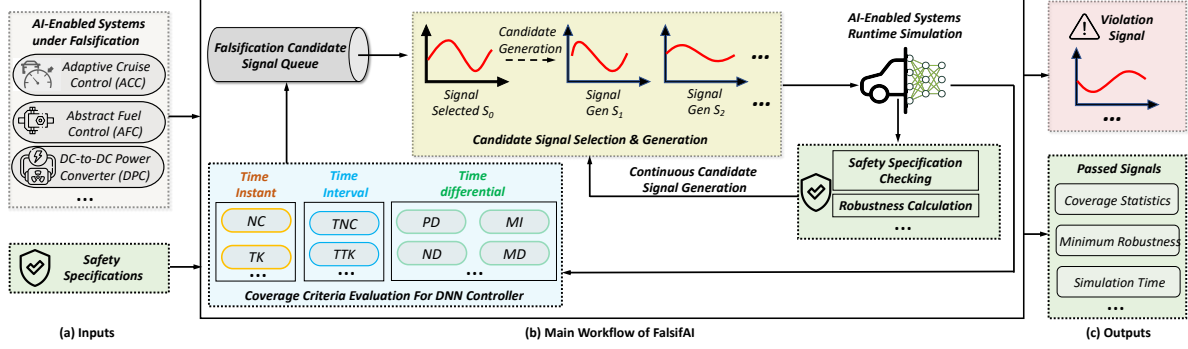


Figure 5.1: FalsifAI workflow: A coverage-guided falsification framework for DNN-enabled control systems

hybrid control system controlled by the DNN controller.

Furthermore, we propose a general-purpose coverage-guided falsification framework for AI-enabled CPS, named FalsifAI, which we used to perform a large-scale evaluation to investigate the usefulness of our proposed time-aware coverage criteria as the guidance for falsification, on a total of 3 typical AI-enabled CPS tasks, 6 system specifications, 18 DNN models, and over 12,000 experiment runs. We also compared FalsifAI with two state-of-the-art falsification methods. The evaluation results and our in-depth study reveal that: 1) with the guidance of proposed time-aware coverage, FalsifAI can indeed outperform the two state-of-the-art falsification approaches, and a falsification approach that achieves exploration by the coverage of the input space; 2) the time-aware coverage exhibits much better performance to characterize the temporal behaviours of a DNN controller repeatedly executed in a hybrid control system; 3) being able to falsify the specification usually correlates with high coverage, showing that the criteria do indeed provide good guidance during the falsification search; 4) the effectiveness of the proposed criteria varies with different parameterizations, and parameter tuning can be used to find a configuration that provides a good guidance in falsification.

The main contributions (also visualized in Fig. 5.1) of this paper are summarized as follows:

- 8 new *time-aware* coverage criteria for DNNs, especially designed for the context of CPS by taking the temporal behaviour of a DNN into account, e.g., the sequence of activation status of

the DNN neurons over time;

- A general-purpose coverage-guided falsification framework, FalsifAI, guided by our time-aware coverage criteria in the falsification exploration of the system behaviours;
- A large-scale evaluation over 3 typical AI-enabled hybrid control systems, 18 DNN models, 10 criteria (including one robustness-based and one based on random input space exploration), with a total of more than 12,000 configuration simulation runs. The results confirm that FalsifAI significantly outperforms the state-of-the-art falsification methods. Our proposed time-aware coverage is indeed helpful for falsification guidance towards better effectiveness.

To the best of our knowledge, this is the first paper that tackles the problem of characterizing the temporal behaviour of a neural network controller, and that exploits this for efficient falsification of AI-enabled hybrid control systems. With the recently increasing trend of DNN component adoption in CPS industrial applications, we believe this work has the great potential to pave the path with very broad impacts, towards providing better quality assurance methods in the era of many upcoming AI-enabled hybrid control systems.

Paper structure. §5.2 introduces the necessary background. §5.3 introduces the proposed time-aware coverage criteria, and §5.4 presents the proposed falsification approach guided by the criteria. Design of the experiments is reported in §5.5, while experimental results are reported in §5.6. §5.7 discusses some threats that may affect the validity of the approach, and §5.8 concludes the paper.

5.2 PRELIMINARIES

In this section, we provide essential background introduction, with definition and formulation of our considered type of system model (§5.2.1) and formal specification (§5.2.2), to facilitate the understanding of our proposed techniques.

5.2.1 SYSTEM MODEL

Definition 1 (Deep Neural Network²). Let $x_1, \dots, x_M$ be M variables. A *deep neural network* (DNN) $C: \mathbb{R}^M \rightarrow \mathbb{R}$ is a function that maps a vector $\vec{x} \in \mathbb{R}^M$ (an assignment of $x_1, \dots, x_M$) to a real number $C(\vec{x}) \in \mathbb{R}$.

A DNN C is composed of a set $N = \{\mathbf{n}_{ij}\}$ of neurons that lie in different hidden layers³, where $i \in \{1, \dots, L\}$ is the index of the hidden layers, and $j \in \{1, \dots, J_i\}$ is the index of the neuron in the i -th layer. Given $\vec{x}$ as the input for the DNN C , each neuron $\mathbf{n}_{ij} \in N$ outputs a real $\varphi^{\mathbf{n}_{ij}}(\vec{x}) \in \mathbb{R}$; the outputs of neurons in the i -th or previous layers are then fed as the input to the neurons in the $(i + 1)$ -th layer:

$$\varphi^{\mathbf{n}_{(i+1)k}}(\vec{x}) = F\left(b + \sum_{j=1}^{J_i} \omega_j \cdot \varphi^{\mathbf{n}_{ij}}(\vec{x})\right)$$

where F is an activation function, $\omega_j \in \mathbb{R}$ ($j = 1, \dots, J_i$) are weights, and $b \in \mathbb{R}$ is a bias. Common choices of activation function F include *ReLU*, *Sigmoid*, *tanh*, etc.

Next, we give the definition of AI-enabled hybrid control system used in this paper, which is based on the ones given in previous work [69, 74, 75, 143]. In the following, we refer to an M -dimensional *signal*, w.r.t. variables $w_1, \dots, w_M$ and a time horizon $T \in \mathbb{R}_{\geq 0}$, as a time-variant function $\mathbf{w}: [0, T] \rightarrow \mathbb{R}^M$.

Definition 2 (AI-Enabled Hybrid Control System). An *AI-enabled hybrid control system* $\mathcal{M}^C$ is depicted in Fig. 5.2. It consists of a neural network controller C and a plant, which works in the following way:

- The *plant* is a system whose dynamics is given by a black-box function $\mathcal{M}$. Its inputs are the plant state $\mathbf{y}(t)$ at time t , and the control command $\mathbf{c}(t)$ at time t ;

²As an early step, we consider feed-forward neural network controllers.

³In general, layers can be characterized as input-, hidden- and output-layers. Similar to previous works, in this paper, we also mainly consider hidden layers for coverage calculation to characterize DNN internal behaviours.

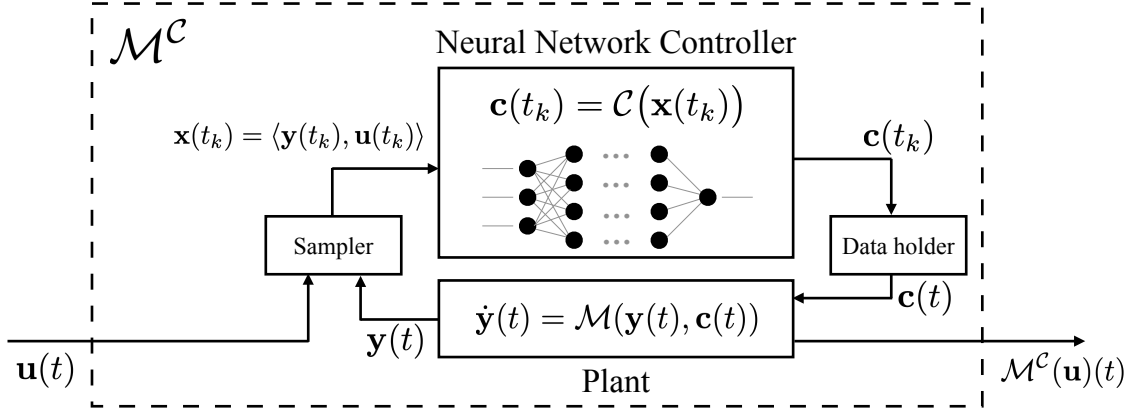


Figure 5.2: AI-enabled hybrid control system

- A *sampler* samples, at regular time points, the state $\mathbf{y}$ of the plant, and the external input signals $\mathbf{u}$ from the environment; at a given time t_k , it produces input $\mathbf{x}(t_k) = \langle \mathbf{y}(t_k), \mathbf{u}(t_k) \rangle$ (i.e., a tuple composed of $\mathbf{y}(t_k)$ and $\mathbf{u}(t_k)$) for the neural network controller $\mathcal{C}$;
- The *neural network controller* $\mathcal{C}$ takes the input $\mathbf{x}(t_k)$ consisting of plant state $\mathbf{y}(t_k)$ and external input $\mathbf{u}(t_k)$, and outputs a control command $\mathbf{c}(t_k) \in \mathbb{R}$;
- A *data holder* transforms the sequence of control commands $\mathbf{c}(t_k)$ into a continuous signal $\mathbf{c}$, and feeds it to the plant for control.

From a black-box view (dashed box in Fig. 5.2), the system takes input signal $\mathbf{u}$, and produces output signal $\mathcal{M}^C(\mathbf{u})$.

Example 1. An Adaptive Cruise Control (ACC) [74, 75] is a typical system of this type. ACC is responsible for keeping a safe distance of the car (called *ego car*) from a lead car; the ego car is the plant in this system. The neural network controller computes control actions for the ego car for keeping a safe distance. The system inputs $\mathbf{u}$ are the position and velocity of the lead car. The state dynamics $\mathbf{y}$ of the ego car (plant), are its position and velocity. The output $\mathbf{c}$ of the DNN is the acceleration of the ego car. The system outputs $\mathcal{M}^C(\mathbf{u})$ are the distance and relative velocity of the ego car with respect to the lead car.

5.2.2 FORMAL SPECIFICATION

Specifications are used to describe the expected system behaviour regarding aspects such as safety, performance, comfort, *etc.* In order to support automatic system validation, they are usually expressed in formal languages. In this work, we select *Signal Temporal Logic (STL)* [35] as our specification language, which is widely adopted to formulate temporal properties of hybrid control systems. In the following, we first present the syntax, and then introduce the quantitative *robust semantics*, which constitutes the core of state-of-the-art black-box falsification techniques.

Definition 3 (STL Syntax). In Signal Temporal Logic (STL), *atomic propositions* and *formulas* are defined as follows, respectively: $\alpha ::= f(x_1, \dots, x_N) > 0$, and $\varphi ::= \alpha \mid \perp \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathcal{U}_I \varphi$. Here, f is an N -ary function $f : \mathbb{R}^N \rightarrow \mathbb{R}$, $x_1, \dots, x_N$ are variables, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e., $I = [a, b]$ or $[a, \infty)$ where $a, b \in \mathbb{R}$ and $a < b$.

We omit subscripts I for temporal operators if $I = [0, \infty)$. Other common connectives such as $\vee, \rightarrow, \top, \Box_I$ (always) and $\Diamond_I$ (eventually), are introduced as abbreviations: $\top \equiv \neg\perp$, $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond_I \varphi \equiv \top \mathcal{U}_I \varphi$ and $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$. An atomic formula $f(\vec{x}) \leq c$, where $c \in \mathbb{R}$, is accommodated using $\neg$ and the function $f'(\vec{x}) := f(\vec{x}) - c$.

Definition 4 (STL Robust Semantics). Let $\mathbf{w} : [0, T] \rightarrow \mathbb{R}^N$ be an N -dimensional signal. Given $t \in [0, T)$, we denote by $\mathbf{w}^t$ the t -shift of $\mathbf{w}$, which is a signal $\mathbf{w}^t : [0, T - t] \rightarrow \mathbb{R}^N$ that holds $\mathbf{w}^t(t') := \mathbf{w}(t + t')$.

Let φ be an STL formula. We define the *robustness* $\llbracket \mathbf{w}, \varphi \rrbracket \in \mathbb{R} \cup \{\infty, -\infty\}$ as follows, by induction on the construction of formulas. $\sqcap$ and $\sqcup$ denote infimums and supremums of real numbers, respectively. $\sqcap$, the binary version of $\sqcap$, denotes minimum.

$$\begin{aligned} \llbracket \mathbf{w}, f(x_1, \dots, x_N) > 0 \rrbracket &:= f(\mathbf{w}(0)(x_1), \dots, \mathbf{w}(0)(x_N)) \\ \llbracket \mathbf{w}, \perp \rrbracket &:= -\infty & \llbracket \mathbf{w}, \neg\varphi \rrbracket &:= -\llbracket \mathbf{w}, \varphi \rrbracket \end{aligned}$$

$$\begin{aligned} \llbracket \mathbf{w}, \varphi_1 \wedge \varphi_2 \rrbracket &:= \llbracket \mathbf{w}, \varphi_1 \rrbracket \cap \llbracket \mathbf{w}, \varphi_2 \rrbracket \\ \llbracket \mathbf{w}, \varphi_1 \mathcal{U}_I \varphi_2 \rrbracket &:= \bigsqcup_{t \in I \cap [0, T]} (\llbracket \mathbf{w}^t, \varphi_2 \rrbracket \cap \prod_{t' \in [0, t)} \llbracket \mathbf{w}^{t'}, \varphi_1 \rrbracket) \end{aligned}$$

The original STL semantics is Boolean, given by a binary relation $\models$ between signals and formulas. The robust semantics refines the Boolean one in the following sense: $\llbracket \mathbf{w}, \varphi \rrbracket > 0$ implies $\mathbf{w} \models \varphi$, and $\llbracket \mathbf{w}, \varphi \rrbracket < 0$ implies $\mathbf{w} \not\models \varphi$.

5.2.3 THE CLASSIC BLACK-BOX FALSIFICATION METHOD

In the classic falsification method, the system model $\mathcal{M}^C$ is treated as a black box (dashed box in Fig. 5.2), i.e., the behaviours of the system are only observed from its input signal $\mathbf{u}$ and the corresponding output signal $\mathcal{M}^C(\mathbf{u})$. Falsification aims at finding an input signal $\mathbf{u}$, such that the output signal $\mathcal{M}^C(\mathbf{u})$ violates a specification φ ; we call such an input a *falsifying input*. With the support of STL robust semantics, falsification can be solved by turning it into an optimization problem that minimizes the robustness $\llbracket \mathcal{M}^C(\mathbf{u}), \varphi \rrbracket$ of the specification φ —once a negative robustness result $\llbracket \mathcal{M}^C(\mathbf{u}), \varphi \rrbracket$ is observed, it means that a falsifying input $\mathbf{u}$ has been found.

The inputs for the optimization problem, i.e., input signals $\mathbf{u}: [0, T] \rightarrow \mathbb{R}^M$, are time-variant continuous functions. In practice, synthesizing such a continuous signal is infeasible. As it suffices to show the existence of a falsifying input, practitioners employ parametrized representations to characterize these signals [33, 46, 73]. A commonly adopted parametrized representation is *piecewise constant*. A piecewise constant signal $\mathbf{u}: [0, T] \rightarrow \mathbb{R}^M$ has a hyperparameter c , and it is defined as such: during each interval $[\frac{(i-1)T}{c}, \frac{iT}{c}]$ ($i \in 1, \dots, c$), $\mathbf{u}(t)$ is a constant. In that way, a signal $\mathbf{u}$ is identified by a finite number $c \cdot M$ of parameters, and so synthesizing such a signal becomes feasible.

The methodology of optimization-based falsification is summarized in Alg. 2. The key to solve the problem is using *stochastic optimization* algorithms, such as *hill climbing* (as shown in Line 6),

Algorithm 2 The classic falsification algorithm

Require: A hybrid control system $\mathcal{M}^C$, an STL specification φ , a budget $B_G \in \mathbb{N}_{>0}$.

```

1: function HILL-CLIMB-FALSIFY( $\mathcal{M}^C, \varphi, B_G$ )
2:   initialize a placeholder  $\mathbf{u}$                                  $\triangleright$  best input so far
3:    $\text{rb} \leftarrow \infty$                                           $\triangleright$  minimum robustness so far
4:    $i \leftarrow 0$                                                 $\triangleright$  budget counter
5:   while  $i < B_G$  and  $\text{rb} > 0$  do
6:      $\mathbf{u}_i \leftarrow \text{HILL-CLIMB}(\langle \mathbf{u}_l, \text{rb}_l \rangle_{l=0, \dots, i-1})$ 
7:      $\text{rb}_i \leftarrow \llbracket \mathcal{M}^C(\mathbf{u}_i), \varphi \rrbracket$                       $\triangleright$  compute robustness
8:     if  $\text{rb}_i < \text{rb}$  then
9:        $\text{rb} \leftarrow \text{rb}_i, \mathbf{u} \leftarrow \mathbf{u}_i$ 
10:     $i \leftarrow i + 1$ 
11:  return  $\begin{cases} \mathbf{u} & \text{if } \text{rb} < 0 \\ \text{Failure} & \text{otherwise, no violation is found} \end{cases}$ 

```

because they work effectively with black box objective functions, e.g., $\llbracket \mathcal{M}^C(\mathbf{u}), \varphi \rrbracket$ in our case. These algorithms adopt various meta-heuristic strategies so that they can efficiently explore the search space and achieve optimal solutions. Alg. 2 starts with initializing two placeholders, $\mathbf{u}$ and rb , for storing the best input signal and its robustness (Line 2 and 3); it then enters the main loop, where hill climbing is employed to search⁴ for the best input signal (Line 6), and $\mathbf{u}$ and rb are updated on the observation of lower robustness (Line 9); the algorithm is terminated either when a falsifying input is found or the budget is reached (Line 11). We refer interested readers to [79] for a more comprehensive survey of different optimization-based falsification algorithms. Also mature tools, such as Breach [33] and S-TaLiRo [10], have been developed; in this work, we select these two tools as baselines to compare with.

5.3 TIME-AWARE COVERAGE CRITERIA

Existing works (such as [81, 104, 105, 126, 138, 168] on DNN coverage) define criteria based on the level of activation of one neuron or of combinations of neurons. However, such existing

⁴Note that the search is bounded by a hyper-rectangle in which the inputs are known to be realistic. In case more complex spaces define the realism, approaches to handle input constraints [172] can be adopted on the top of the falsification algorithm.

criteria are not suitable for assessing the coverage of the DNNs used in AI-enabled hybrid control system $\mathcal{M}^C$. In $\mathcal{M}^C$, the neural network controller is the central component responsible for the control of the plant *over time*. Therefore in this paper, we hypothesize that characterizing the temporal behaviour of the DNN controller “over time”, when it is executed together with the whole control system, is useful to understand and characterize the overall behaviour of $\mathcal{M}^C$. Moreover, the behaviour of the DNN controller is better characterized by its internal status of neuron activation, rather than just its output; this idea is similar to using structural coverage criteria in testing of classical programs.

So, we propose novel coverage criteria that are *time-aware*, as they take the *temporal behaviour* of the DNN into account. The temporal behaviour of the DNN is defined as the outcome of its neurons during the executions over time. For example, from the temporal perspective, instead of stating “a neuron is covered if its output is larger than h ” (by considering only one DNN execution) [126], we state that “a neuron is covered if its output is larger than h **for τ time units**” (by considering consecutive executions of the DNN within a period of time).

Namely, we design criteria from three perspectives, that inspect different characteristics of the DNN temporal behaviour: the activation of neurons at some timestamp during the DNN temporal behaviour (§5.3.1), the continuous activation of neurons for some time intervals (§5.3.2), and the change (i.e., differential behaviour) of activation values over time (§5.3.3).

In the following, let $\mathbf{x} = \langle \mathbf{x}(t_0), \mathbf{x}(t_1), \dots, \mathbf{x}(t_n) \rangle$ be a *temporal input*, i.e., a sequence of inputs provided to the DNN over the time horizon $[0, T]$, and X be a set of temporal inputs. Our coverage criteria are defined based on the neuron outputs of the intermediate layers, as previous works [165] indicate that these intermediate outputs can better characterize the behavioral features of DNNs.

5.3.1 TIME INSTANT COVERAGE CRITERIA

The criteria of this category require that neurons are activated *at any particular timestamp* in the time horizon, to show the existence of the activation state for a neuron. The time instant neuron coverage and instant top- k neuron coverage are inspired by [126] and [105], respectively. In [105, 126], the neuron activation status is defined for a single inference of a DNN; in comparison, our coverage criteria require that the neuron activation status (as defined in [105, 126]) is “eventually” achieved at an instant over the execution time of the system.

Instant Neuron Coverage (NC). The instant neuron coverage NC is defined as follows:

$$\text{NC}(h) := \frac{\left| \{ \mathbf{n} \in N \mid \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T] : \varphi^{\mathbf{n}}(\mathbf{x}(t)) > h \} \right|}{|N|}$$

The criterion is parameterized with constant h , and it checks the ratio of neurons having an activation value higher than h in at least one timestamp.

Instant Top- k Neuron Coverage (TK). We first define a function $\text{top}^k(\mathbf{x}, i)$ that sorts the neurons $\mathbf{n}_{ij}$ ($j = \{1, \dots, J_i\}$) in the i -th layer in decreasing order according to their output $\varphi^{\mathbf{n}_{ij}}(\vec{x})$, and returns the first k neurons. Instant top- k neuron coverage is defined as follows:

$$\text{TK}(k) := \frac{\left| \bigcup_{\mathbf{x} \in \mathcal{X}} \bigcup_{1 \leq i \leq L} \bigcup_{t \in [0, T]} \text{top}^k(\mathbf{x}(t), i) \right|}{|N|}$$

The criterion computes the ratio of neurons that are ever in the top k at some point in at least one temporal input. Note that this criterion does not focus on the value of the activation function (i.e., there is no threshold), but on the relation between the values of the activation functions of the different neurons.

5.3.2 TIME INTERVAL COVERAGE CRITERIA

The criteria of this category not only check whether some neurons are activated at some timestamp, but also whether they are *continuously* activated for a given period of time (i.e., in consecutive invocations of the DNN). Intuitively, this category characterizes constancy of neuron state activation status over time.

Timed Neuron Coverage (TNC). We define the following predicate for a given neuron $\mathbf{n}$ assessing whether it has been continuously activated (more than h) during interval I :

$$P_{\text{TNC}}(\mathbf{n}, \mathbf{x}, h, I) := \bigwedge_{t \in I} \varphi^{\mathbf{n}}(\mathbf{x}(t)) > h$$

Intuitively, a continuous activation of a neuron identifies a situation in which the neuron holds its activation status continuously over a time interval.

The timed neuron coverage is then defined as follows, where the numerator collects all neurons that have been continuously activated for a τ period of time in at least one temporal input.

$$\text{TNC}(h, \tau) := \frac{\left| \left\{ \mathbf{n} \in N \mid \begin{array}{l} \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T - \tau]: \\ P_{\text{TNC}}(\mathbf{n}, \mathbf{x}, h, [t, t + \tau]) \end{array} \right\} \right|}{|N|}$$

Timed Top- k Neuron Coverage (TTK). We first define a function selecting the neurons that are continuously in the top k positions in a given interval I :

$$N_{\text{TTK}}(\mathbf{x}, k, I) := \bigcap_{t \in I} \text{top}^k(\mathbf{x}(t), i)$$

The assumption in top- k neuron coverage in [105] is that the neurons ranked as the top k are considered to be highly influential to the final output; here, we extend this intuition to the tem-

poral execution of the system as follows: if a neuron is continuously in the top k status for some time, it means that it is highly influencing the decision during that period of time.

The timed top- k neuron coverage is defined as follows, where the numerator selects the neurons that, at some point, are continuously in the top k for τ time units in some temporal input.

$$\text{TTK}(k, \tau) := \frac{\left| \bigcup_{\mathbf{x} \in \mathcal{X}} \bigcup_{1 \leq i \leq L} \bigcup_{t \in [0, T-\tau]} N_{\text{TTK}}(\mathbf{x}, k, [t, t + \tau]) \right|}{|N|}$$

5.3.3 TIME DIFFERENTIAL COVERAGE CRITERIA

The coverage criteria of this category assess the relative change in the evaluation of the neurons over time. Intuitively, these criteria characterize inconstancy of neuron activation status over time. In particular, we introduce the following notation to measure the differential behaviour of a neuron between two timestamps: $\varphi_{\Delta(t_1, t_2)}^{\mathbf{n}}(\mathbf{x}) = \varphi^{\mathbf{n}}(\mathbf{x}(t_2)) - \varphi^{\mathbf{n}}(\mathbf{x}(t_1))$.

Positive/Negative Differential Neuron Coverage (PD/ND). We first define two predicates that assess whether the evaluation of a neuron $\mathbf{n}$ respectively increases or decreases by at least h in a time interval I :

$$P_{\text{PD}}(\mathbf{n}, \mathbf{x}, h, I) := \exists t_1, t_2 \in I: t_1 < t_2 \wedge \varphi_{\Delta(t_1, t_2)}^{\mathbf{n}}(\mathbf{x}) > h$$

$$P_{\text{ND}}(\mathbf{n}, \mathbf{x}, h, I) := \exists t_1, t_2 \in I: t_1 < t_2 \wedge \varphi_{\Delta(t_2, t_1)}^{\mathbf{n}}(\mathbf{x}) > h$$

The intuition is that diverse temporal activation patterns in terms of change could be characterized, i.e., a neuron changes its status in a non-negligible way (measured by h).

We collect in two sets all the neurons whose activation functions respectively increased or

decreased by at least h in an interval of τ length, in at least one temporal input:

$$S_{PD}(h, \tau) = \left\{ \mathbf{n} \in N \left| \begin{array}{l} \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T - \tau]: \\ P_{PD}(\mathbf{n}, \mathbf{x}, h, [t, t + \tau]) \end{array} \right. \right\}$$

$$S_{ND}(h, \tau) = \left\{ \mathbf{n} \in N \left| \begin{array}{l} \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T - \tau]: \\ P_{ND}(\mathbf{n}, \mathbf{x}, h, [t, t + \tau]) \end{array} \right. \right\}$$

Then, the positive/negative differential neuron coverage is defined as follows:

$$PD(h, \tau) := \frac{|S_{PD}(h, \tau)|}{|N|} \quad ND(h, \tau) := \frac{|S_{ND}(h, \tau)|}{|N|}$$

Monotonic Increase/Decrease Neuron Coverage (MI/MD). We define two predicates that check whether the activation function of a neuron $\mathbf{n}$ constantly increases or decreases in the interval I :

$$P_{MI}(\mathbf{n}, \mathbf{x}, I) := \forall t_1, t_2 \in I: (t_2 - t_1) \cdot \varphi_{\Delta(t_1, t_2)}^{\mathbf{n}}(\mathbf{x}) > 0$$

$$P_{MD}(\mathbf{n}, \mathbf{x}, I) := \forall t_1, t_2 \in I: (t_2 - t_1) \cdot \varphi_{\Delta(t_1, t_2)}^{\mathbf{n}}(\mathbf{x}) < 0$$

The intuition is that when the activation of a neuron keeps on increasing, the neuron may become more sensitive to the input data and so it possibly influences the decision process more. Similarly, when the activation decreases, the neuron may become less relevant in the decision process.

We collect in two sets all the neurons whose activation functions monotonically increased or

decreased for an interval of τ length in at least one temporal input:

$$S_{MI}(\tau) = \left\{ \mathbf{n} \in N \left| \begin{array}{l} \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T - \tau]: \\ P_{MI}(\mathbf{n}, \mathbf{x}, [t, t + \tau]) \end{array} \right. \right\}$$

$$S_{MD}(\tau) = \left\{ \mathbf{n} \in N \left| \begin{array}{l} \exists \mathbf{x} \in \mathcal{X}, \exists t \in [0, T - \tau]: \\ P_{MD}(\mathbf{n}, \mathbf{x}, [t, t + \tau]) \end{array} \right. \right\}$$

Then, the monotonic increase/decrease neuron coverage criteria are defined as follows:

$$MI(\tau) := \frac{|S_{MI}(\tau)|}{|N|} \quad MD(\tau) := \frac{|S_{MD}(\tau)|}{|N|}$$

5.4 COVERAGE-GUIDED FALSIFICATION

The *exploration-exploitation trade-off* is a central issue in search-based techniques [30], where *exploration* refers to the search in different areas of the search space, while *exploitation* refers to the local search in a specific area. Such a problem also applies to falsification of continuous signals. The quantitative robustness serving as the objective function for the classic falsification approach (see §5.2), can be very useful for guiding the exploitation phase; however, it does not provide informative guidance to explore the system state space.

The problem is further exacerbated when considering AI-enabled hybrid control systems (see §5.2.1) as the target of the falsification. Indeed, the complex decision logic of the DNN itself is already quite difficult to explore, and it is not clear if enough input signals have been used to explore all the possible behaviours.

Therefore, to improve the exploration phase of falsification and thus increase the diversity of internal behaviour coverage of DNN controllers, we propose FalsifAI, a coverage-guided falsification framework, whose workflow is shown in Fig. 5.1. Compared to the classic falsification

approach (see §5.2.3), FalsifAI performs a more thorough exploration of systems behaviours, under the guidance provided by the coverage criteria we proposed in §5.3.

We first give a high-level overview of FalsifAI in §5.4.1, and then elaborate on its hill climbing-based mutation sub-process in §5.4.2. Note that besides the aforementioned motivation, one important goal of this paper is to demonstrate the potential usefulness of our proposed criteria to empower the falsification process.

5.4.1 OVERVIEW OF THE COVERAGE-GUIDED FALSIFICATION

The proposed falsification algorithm consists of two embedded loops, to jointly search the falsification space with a balance between *exploration* and *exploitation*. The outer loop performs exploration guided by coverage criteria to find signals that cover particular behaviours of the neural network controller, while the inner loop performs exploitation for searching in the neighborhood of such signals with the aim of minimizing robustness. Formally, we present the proposed coverage-guided falsification approach in Alg. 3.

Input. The algorithm requires an AI-enabled system model $\mathcal{M}^C$, equipped with a neural network controller C , an STL formula φ specifying the expected behaviour, a constant K , a *global budget* B_G , and a *local budget* parameter B_L . We will give their more detailed description later in the section.

Note that, differently from the black-box setting of a classic falsification method, we treat the system as a gray-box: we not only collect the system output $\mathcal{M}^C(\mathbf{u})$ under a given input $\mathbf{u}$, but we also gather the output of the intermediate neurons of the neural network controller C , to compute coverage as a guidance for our search.

Initialization. The algorithm starts with the initialization of a candidate signal queue Q_{seed} of initial *seeds* (Line 2), by uniformly random sampling in the space of input signals; so, a seed is a temporal input signal $\mathbf{u}$ that can be fed to the system model. The purpose of the seed queue is to

Algorithm 3 FalsifAI: Coverage-guided falsification

Require: An AI-enabled system $\mathcal{M}^C$ with a neural network controller C , an STL specification φ , the initial size $K \in \mathbb{N}_{>0}$ of the seed queue, a global budget $B_G \in \mathbb{N}_{>0}$, and a local budget $B_L \in \mathbb{N}_{>0}$.

```

1: function FALSIFAI( $\mathcal{M}^C, \varphi, K, B_G, B_L$ )
2:   initialize  $Q_{\text{seed}}$  by random sampling  $K$  input signals
3:    $\mathcal{X} \leftarrow \emptyset$  ▷ collection of all sampled signals
4:    $\text{cov} \leftarrow 0$  ▷ coverage achieved by  $\mathcal{X}$ 
5:    $\text{rb} \leftarrow \infty$  ▷ minimum robustness so far
6:   while within the global budget  $B_G$  do
7:     for  $i \in \{0, \dots, B_L - 1\}$  do
8:        $\mathbf{u}_i \leftarrow \begin{cases} \text{DEQUEUE}(Q_{\text{seed}}) & \text{if } i = 0 \\ \text{HILL-CLIMB}(\langle \mathbf{u}_l, \text{rb}_l \rangle_{l=0, \dots, i-1}) & \text{otherwise} \end{cases}$ 
9:        $\text{rb}_i \leftarrow \llbracket \mathcal{M}^C(\mathbf{u}_i), \varphi \rrbracket$  ▷ robustness
10:       $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{u}_i\}$ 
11:       $\text{cov}_i \leftarrow \text{CALCCOV}_{\text{Cr}}(\mathcal{X}, C)$  ▷ coverage update
12:      if  $\text{rb}_i < 0$  then
13:        return  $\mathbf{u}_i$  ▷ falsifying input
14:      else if  $\text{cov}_i > \text{cov}$  then
15:         $Q_{\text{seed}} \leftarrow \text{ENQUEUE}(Q_{\text{seed}}, \mathbf{u}_i)$ 
16:         $\text{cov} \leftarrow \text{cov}_i$ 

```

improve the diversity of the generated inputs, and this is achieved in two ways: 1) it collects the input signals that increase the cumulative coverage of the neural network controller, so possibly showing different types of behaviour; 2) the seed signals are arranged in a FIFO manner, so that priority is given to the seeds that have not been visited for a longer period. Besides the seed queue, the algorithm also initializes a set $\mathcal{X}$ for collecting the inputs generated during the search (Line 3), and two variables for keeping track of the current coverage level cov (Line 4) and the minimum observed robustness rb (Line 5).

Falsification Loop. As long as there is enough budget B_G (Line 6), the algorithm performs different local searches that aim at finding a falsifying input, starting from a different initial seed popped from the seed queue Q_{seed} . Each local search is performed for B_L steps (Line 7); for each step i , it performs the following operations:

- it samples an input signal. This consists in two cases, as shown in Line 8:

- at the beginning of the search (i.e., $i = 0$), it dequeues the head element of Q_{seed} , and assigns it to $\mathbf{u}_0$;
- in the next iterations, it samples a point by considering the history of previously sampled points; more details on this step are given in §5.4.2.
- it performs a simulation, by feeding the sampled input signal $\mathbf{u}_i$ to the system, and obtains the outputs, including the output signal $\mathcal{M}^C(\mathbf{u}_i)$, and the outputs $\varphi^n(\mathbf{x}_i)$ ($n \in N$) of the neurons in the intermediate layers of the neural network controller C . These outputs are used for two purposes:
 - to compute the robustness $\llbracket \mathcal{M}^C(\mathbf{u}_i), \varphi \rrbracket$ of the output signal $\mathcal{M}^C(\mathbf{u}_i)$ to the given specification φ (Line 9), by applying the STL robust semantics in Def. 4;
 - to compute the coverage achieved by the sampled input signals collected in $\mathcal{X}$, for the neural network controller C , using one coverage criterion Cr defined in §5.3 (Line 11)⁵;
- if the robustness is negative (Line 12), it means that $\mathbf{u}_i$ is a falsifying input, and so it is returned as final result (Line 13), witnessing that the specification φ does not hold;
- otherwise, if the coverage is increased (Line 14), the input $\mathbf{u}_i$ is added to the end of Q_{seed} (Line 15), because it is an input that covers a new behaviour of the neural network controller, and so it is worthy of exploitation in subsequent falsification trials; moreover, the coverage information is updated (Line 16). If no new behaviour is covered by $\mathbf{u}_i$, the algorithm simply drops it, and enters the next loop.

Remark 1. In our algorithm, we combine exploration (Line 6 Alg. 3) and exploitation (Line 7 in Alg. 3), the former embodied by the use of the seed queue, while the latter achieved via the mutation. Specifically, the elements are arranged in the queue in a way that a seed that has not been visited longer, will be visited sooner. Moreover, to avoid exploiting signals triggering similar DNN behaviour, we tend to explore those that increase the coverage, i.e., the diversity of system

⁵To improve the code efficiency, we maintain a set of covered neurons, and simply check if there are new neurons covered by the new input.

behaviours.

5.4.2 HILL CLIMBING-BASED MUTATION

In a general search-based testing framework, a *mutation* operation is performed for a local search that aims at finding a falsifying input, starting from an initial seed. In our context, we employ the *hill-climbing optimization* technique as the engine for mutation (i.e., Line 8 in Alg. 3).

Hill-climbing optimization has proved to be efficient in numerical search, and so it is also adopted in the classic falsification, as introduced in §5.2.3. Typically, it first randomly samples some points in the input space and computes their robustness; then, it looks at the history of previous observations $\langle \mathbf{u}_l, \text{rb}_l \rangle_{l=0, \dots, i-1}$ of input signals $\mathbf{u}_l$ with their corresponding robustness values rb_l , and suggests a new input signal that is close to an observed input signal, with the aim of decreasing the robustness. In comparison with the pure random sampling strategy, hill-climbing optimization has guidance given by the robustness, and so it is more intelligent and more efficient, as shown by existing works [177].

As explained in §5.2.3, inputs to the falsification problem are piecewise constant signals, i.e., sequences of $c \cdot M$ constant values (c values for each signal dimension). Therefore, in our context, the mutation (i.e., HILL-CLIMB function in Line 8 of Alg. 3) of an input signal consists in modifying one or more of these $c \cdot M$ values.

5.5 EVALUATION DESIGN AND SETTING

We here describe the design of the experiments we conducted to evaluate the effectiveness of our proposed coverage-guided falsification method FalsifAI. All the source code and benchmarks are available online [175].⁶

⁶<https://github.com/choshina/FalsifAI>

5.5.1 RESEARCH QUESTIONS

In order to assess the effectiveness of the proposed approach, we identify the following research questions:

- **RQ1:** *Falsification Success Rate (Effectiveness).* Is FalsifAI more effective, in terms of falsification results, than the state-of-the-art falsification methods?
- **RQ2:** *Effectiveness of Coverage in Guiding Falsification.* Which is the best time-aware criterion in terms of guidance for the falsification?
- **RQ3:** *Coverage Achieved by Successful and Unsuccessful Falsification Trials.* What is the difference between the obtained coverage level of successful (i.e., the specification is falsified) and unsuccessful falsification trials?
- **RQ4:** *Parameter Impact on Coverage Effectiveness.* What is the influence of the parameters of the coverage criteria on the effectiveness of FalsifAI?
- **RQ5:** *Scalability w.r.t DNN Size.* How does the effectiveness of the approach change as the size of the neural network controller increases?

5.5.2 EVALUATION METRICS

Experiments are assessed using the following metrics: (i) *SR* (Success Rate): the number of runs, out of 30, in which the falsification approach is successful, i.e., it found a falsifying input for the specification of a given benchmark. (ii) *time*: the average time of the successful runs, i.e., those in which a falsifying input is found. (iii) *#sim*: the average number of simulations (sampled inputs) of the successful runs.

RQ1. In order to answer RQ1, we compare the SR of FalsifAI and of the baseline approaches (see §5.5.4) across all the benchmarks. In order to draw more definitive conclusions, we conduct pairwise comparison of the results of the baseline approaches with the results of each version of FalsifAI, using the Wilcoxon signed rank test for the statistical significance (at significance level

$\alpha = 0.05$), and the Vargha-Delaney’s $\hat{A}_{12}$ as effect size, by following the guideline in [12]. When comparing SR values (across all the benchmarks) of two falsification approaches, if the p-value of the Wilcoxon signed rank test is less than α , it means that there is a significant difference. In this case, if the $\hat{A}_{12}$ value is higher, it is more likely that the first approach is better than the second, and the higher $\hat{A}_{12}$ the higher the likelihood. $\hat{A}_{12}$ values below 0.5 mean that the second approach is more likely to be better.

RQ2. To answer RQ2, we compare the SR results of the different instantiations of FalsifAI with the different coverage criteria, using the Wilcoxon signed rank test and the Vargha-Delaney’s $\hat{A}_{12}$ effect size as done for RQ1. Moreover, we compare the approaches in a more detailed way, by considering each benchmark individually; given two approaches and one benchmark, we build a contingency table containing how many times (out of 30) each approach has been successful (i.e., it falsified) and how many times it was not. The contingency table is given as input to the Fisher’s exact test (with $\alpha = 0.05$); in case of significantly different results, we use odds ratio (OR) as the effect size measure: an OR value greater than 1 means that the first approach is highly likely to be better than the second, and lower than 1 the opposite. We then count how many times (across all the benchmarks) one approach is better than the other.

RQ3. This RQ aims to reveal the effectiveness of the proposed coverage criteria, by comparing the coverage levels achieved by falsified and unfalsified trials. To answer this RQ, we select the benchmarks that have comparable numbers of falsified trials and unfalsified trials, and we compare their coverage levels. In this way, we can check whether successful falsification is usually correlated to higher coverage.

RQ4. To understand the impact of the coverage parameters on the effectiveness of the approach, we select two benchmarks, and vary their coverage parameters: for each criterion, we select 8 different parameter configurations. We run FalsifAI with the differently configured coverage criteria, and check how the SR varies by changing the parameter configurations.

RQ5. In this RQ, we aim to understand the scalability of the proposed approach to neural network controllers of different sizes. The scalability concern arises from the time consumption of coverage computation for neural network controllers of larger size: indeed, if that is costly, out of the total time budget, less time will be spent in actual search, and so the performance of `FalsifAI` could be diminished. To assess this cost, we proceed as follows. First, since the neural network controllers used in our experiments have different sizes, we can observe the SR results of the largest neural network controllers to see if their sizes bring negative impact on their falsification performance; second, for these neural network controllers that are more likely to suffer from the scalability issue, we record the time consumption for computing coverage and that for system simulation respectively, to see the ratio of the time spent on these two components.

5.5.3 BENCHMARKS

As benchmarks, we consider Simulink system models, that are largely used for CPS modelling in industry. We selected three models that have been widely used in the existing literature, covering diverse application scenarios: *Adaptive Cruise Control (ACC)* [74, 75, 143, 144], *Abstract Fuel Control (AFC)* [46, 48, 73], and *DC-to-DC Power Converter (DPC)* [120].

Adaptive Cruise Control (ACC). As introduced in Ex. 1, this system model intends to maintain the safety distance of the ego car from a lead car, by controlling the ego car; it achieves this by tracking a set of signals, including the position P_{lead} and velocity v_{lead} of the lead car.⁷ The plant in this model is the ego car, while the lead car plays the role of the external environment. The input signal F_{lead} is correlated to the acceleration of the lead car. The range of F_{lead} is $[-1, 1]$ in our experiment, and we simulate the system for 50 seconds.

- The specification S_{ACC}^1 aims to maintain a safe distance d_{rel} between two cars, where the expected distance between two cars is defined by the sum of a constant d_{safe} and the braking

⁷Note that our model is slightly adapted from the original one in [74, 75], in that the motion of the leading car is decided by external inputs. Consequently, this adaptation significantly enlarges the input space of the entire system.

distance $1.4 * v_{ego}$ of the ego car. In our experiment, we set d_{safe} to be 4.

$$S_{ACC}^1 = \square_{[0,50]}(d_{rel} \geq d_{safe} + 1.4 * v_{ego})$$

- The specification S_{ACC}^2 aims to evaluate the system's ability to recover from a dangerous status. Here, the situation where the relative distance d_{rel} between two cars is smaller than $1.4*v_{ego}+4.5$ is considered as a dangerous status, and S_{ACC}^2 requires that the system should recover from the status in 5 seconds.

$$S_{ACC}^2 = \square_{[0,50]} \left(\begin{array}{l} d_{rel} < 1.4 * v_{ego} + 4.5 \rightarrow \\ \diamond_{[0,5]}(d_{rel} > 1.4 * v_{ego} + 4.5) \end{array} \right)$$

Abstract Fuel Control (AFC). This system model was originally a powertrain control model, released by *Toyota* [73], and now has been widely used in the falsification community [46, 48]. The plant of the system consists of a throttle and intake manifold, and the air-path dynamics; the controller is used to control the *air-to-fuel* ratio, which is an important attribute related to the safety and performance of the system. The system has two external signals, *engine speed* and *pedal angle*. In our experiments, we set the ranges of engine speed and pedal angle as [900, 1100] and [8.8, 61.1], respectively, as required by [73] to guarantee the system working in the *normal* mode; and we simulate for 30 seconds.

- The specification S_{AFC}^1 requires that, during the simulation, the system output, namely the *air-to-fuel* ratio AF, never deviates from the reference value AFref by more than α percent. Here, AFref is 14.7; in our experiment, we set α to be 20. The specification is written in STL as follows:

$$S_{AFC}^1 = \square_{[0,30]} \left(\left| \frac{AF - AFref}{AFref} \right| < \alpha \right)$$

- The specification S_{AFC}^2 requires that, once AF ratio deviates from AFref for a value of 10%, it should promptly respond by adjusting the ratio to the acceptable range.

$$S_{AFC}^2 = \square_{[10,30]} \left(\begin{array}{l} \left| \frac{AF - AFref}{AFref} \right| > 0.1 \rightarrow \\ \diamond_{[0,1.5]} \left(\left| \frac{AF - AFref}{AFref} \right| < 0.1 \right) \end{array} \right)$$

- The specification S_{AFC}^3 requires that a large deviation of AF from AFref never lasts for more than 3 seconds. Once such a dangerous status continues for more than 3 seconds, the specification is considered to be violated.

$$S_{AFC}^3 = \square_{[10,30]} \left(\diamond_{[0,3]} \left(\left| \frac{AF - AFref}{AFref} \right| < 0.1 \right) \right)$$

DC-to-DC Power Converter (DPC). This system model has been studied heavily as a traditional control model [68], and was proposed as an AI-enabled benchmark in [120]. It models a DC-to-DC power converter, that converts a source voltage level to a load voltage level. The system has one input signal, that is, a reference voltage V_{ref} in the range $[10, 15]$, and one output signal, that is, an output voltage V_{out} . The specification of the model regulates the accuracy of the conversion, i.e., the output V_{out} should not deviate from the reference V_{ref} for ϵ voltage, after the initial stage, which is formalized as follows:

$$S_{DPC}^1 = \square_{[1,6]} (|V_{out} - V_{ref}| < \epsilon)$$

SUBJECT NEURAL NETWORK CONTROLLERS

To obtain diverse neural network controllers with competitive performance for the three previous models, we adopt the training process reported in [143], with the following steps:

- we first obtain a Simulink model with a sound traditional ODE-based controller w.r.t. a specification φ , by running the model with the traditional controller using different input signals,

Table 5.1: The structures of the neural network controllers used in our experiments

Model ID	ACC#1	ACC#2	ACC#3	ACC#4	ACC#5	ACC#6
#layers	3	4	4	5	5	3
#neurons per layer	50	30	50	30	50	130
Model ID	AFC#1	AFC#2	AFC#3	AFC#4	AFC#5	AFC#6
#layers	3	3	4	4	6	6
#neurons per layer	15	20	10	15	20	25
Model ID	DPC#1	DPC#2	DPC#3	DPC#4	DPC#5	DPC#6
#layers	3	4	4	6	8	3
#neurons per layer	30	10	20	7	7	50

and fixing the parameters in φ , such that the model satisfies φ in most of the cases;

- we then generate training data by running the model with the traditional controller, and outputting the inputs and outputs for the controller as the training set;
- finally, we train a neural network using the *DeepLearning* toolbox of MATLAB; after training and validating the neural network, we replace, in the Simulink model, the traditional controller with the neural network controller.

Note that, in this work, our system inputs are constrained by input ranges. The training data are sampled from those ranges, and the falsifying inputs are also searched in the same ranges. Therefore, the possible falsifying inputs that are out of the distribution of the input ranges are not considered in this work.

To test neural networks with diverse structures, for each of the three models, we trained six different neural network controllers, having different numbers of layers and of neurons in a layer⁸; note that the neural networks for the different models are different from each other. So, in total, we have 18 ($= 3 \times 6$) subject neural networks in our experiments. We will denote these 18 benchmark instances as $B\#i$, where B is the name of the system model (i.e., ACC, AFC, or DPC), and $\#i$ identifies the corresponding neural network controller ($i = 1, \dots, 6$). The structures of the neural network controllers are presented in Table 5.1. Overall, the benchmark models span from

⁸In preliminary experiments, we tried different architectures, and we selected those performing better.

Table 5.2: Parameter selection for coverage criteria. τ : secs

Model	Params	NC	TK	TNC	TTK	PD	ND	MI	MD
ACC	τ	-	-	5	5	5	5	5	5
	h	10	-	10	-	10	10	-	-
	k	-	3	-	3	-	-	-	-
AFC	τ	-	-	3	3	3	3	2	2
	h	0.15	-	0.15	-	0.15	0.15	-	-
	k	-	2	-	3	-	-	-	-
DPC	τ	-	-	10	5	10	10	1	1
	h	5	-	5	-	10	10	-	-
	k	-	3	-	3	-	-	-	-

controllers having 40 neurons to controllers having 390 neurons; such sizes are representative of typical sizes of neural network controllers (e.g., the controller ACC used in [143] has 100 neurons).

5.5.4 EXPERIMENT SETUP

FalsifAI is experimented with all the 8 time-aware coverage criteria proposed in §5.3; we will identify as Fal_{Cr} the approach initialized with criterion Cr. Depending on the values set for their parameters, some coverage criteria could be very difficult or very easy to achieve. For our purposes (i.e., guidance during falsification), we would like a criterion whose coverage does increase during the search, but that it is still not too easy to achieve. To decide the values of the parameters, we proceeded as follows. For each parameter, we fixed a set of possible values and, for each value, we run the model with 10 input signals and computed the achieved coverage; note that running 10 simulations means to evaluate the DNN roughly 6% of the times that it is evaluated during the falsification experiments, where we set an execution budget of 150 simulations. Then, we sort the parameter values by the obtained coverage and we select the first one that obtains at least 10% of the coverage, or, if no parameter value reaches 10% of the coverage, the one obtaining maximum coverage. The parameters values obtained using this procedure are used in the experiments and are reported in Table 5.2. In any case, we will investigate the effect

of using different parameter values in RQ4.

We compare FalsifAI with the state-of-the-art falsification method Breach [33] (that implements Alg. 2) that we take as first baseline (called Br). Moreover, we also compare it with S-TaLiRo [10], another state-of-the-art falsification method, that employs a different optimization algorithm than Breach; we take S-TaLiRo as second baseline (called ST). FalsifAI uses the three categories of time-aware coverage criteria in §5.3, that characterize the temporal behaviour of DNN controllers in different ways, to increase the exploration of the system behaviours and possibly obtain better falsification results. In order to check whether this type of coverage is really advantageous, we compare it with a more simple exploration guided by the coverage of the input space. Namely, we modify FalsifAI by replacing the ENQUEUE operation based on DNN temporal coverage (Lines 14-16 in Alg. 3) by a random sampling in the input space; we call this third baseline approach as $\text{Fal}_{\text{InpSp}}$.

In our context, an *experiment* is the execution of one approach over a specific benchmark model with the corresponding formal specification under falsification. For all the approaches, we set a budget (B_G in Alg. 2 and 3) of 150 simulations. To counteract the randomness of these algorithms, each configuration has been repeated for 30 trials, as suggested by guidelines for conducting experiments [12] with randomized algorithms.

Software Dependencies. The ACC model requires the *Model Predictive Control* and *Control System* MATLAB toolboxes; the DPC model requires the *Stateflow* MATLAB toolbox. The training for all the models requires the *Deep Learning* MATLAB toolbox.

Hardware Platform. For DNN training, we use a server with Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz Processor, 24 CPUs and 15G RAM, and a TITAN V GPU. To enable our large-scale evaluation with over 12,000 experiment runs, for the execution of the falsification approaches, we use 10 AWS cloud instances of type c4.2xlarge: 2.9 GHz Intel Xeon E5-2666 v3, 15G RAM.

5.6 EVALUATION RESULTS

Table 5.3 reports experimental results. For each approach and benchmark, it reports metrics *SR*, *time*, and *#sim* (see §5.5.1).

RQ1 *Falsification Success Rate (Effectiveness)*

For the results of SR, we observe that all the versions of FalsifAI are often better than the state-of-the-art falsification tools Breach and S-TaLiRo that are only guided by robustness. For 23 benchmarks, all FalsifAI versions are strictly better than Breach, and, for seven benchmarks, at least six versions of FalsifAI are better. Only in AFC#4 for S_{AFC}^1 , FalsifAI is consistently worse than Breach. For 20 benchmarks, all FalsifAI versions are strictly better than S-TaLiRo and, for four benchmarks, at least six versions of FalsifAI are better. Only in DPC#1 for S_{DPC}^1 , FalsifAI is consistently worse than S-TaLiRo.

The baseline approach Fal_{InpSp} usually performs better than Breach and S-TaLiRo, but it is almost never the best approach (only in AFC#3 and AFC#6 for S_{AFC}^3 , in AFC#2 for S_{AFC}^3 , and in DPC#4 for S_{DPC}^1).

We highlighted best performers in each benchmark giving first priority to SR and then to time cost.

In order to draw more definitive conclusions, we conducted pairwise comparison of the results of the baselines with all the versions of FalsifAI, using the Wilcoxon signed rank test and the Vargha-Delaney’s $\hat{A}_{12}$ effect size (see §5.5.2). We observed that any version of FalsifAI is significantly better than Breach and S-TaLiRo, but there is no significant difference with Fal_{InpSp} . Moreover, there is no significant difference between Breach and S-TaLiRo. Instead, Fal_{InpSp} is significantly better than both Breach and S-TaLiRo; this shows that more exploration provides an advantage and so we will not consider Breach and S-TaLiRo in the following RQs. In the next RQ, we will investigate which is the best way to achieve good exploration.

Table 5.3: RQ1 and RQ2 – Experimental results for ACC, AFC, and DPC with their specifications. (SR: number of successful runs finding a falsifying input (out of 30). *time*: average time of successful runs. *#sim*: average number of simulations of successful runs). Best results are highlighted in gray

Spec.	Alg.	ACC#1			ACC#2			ACC#3			ACC#4			ACC#5			ACC#6		
		SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ¹ _{ACC}	Br	6	49.7	16.5	1	62.9	21	11	84.5	28.2	8	74.0	24.8	27	80.0	26.7	8	80.4	18.9
	ST	5	331.2	82.8	3	272.3	68.0	3	248.3	49.7	7	155.4	38.7	20	303.6	56.6	3	709.0	99.3
	Fal _{InpSp}	24	190.6	56.3	15	304.9	84.2	29	238.9	56.2	25	248.3	57.6	30	174.3	33.8	26	480.3	73.6
	Fal _{NC}	25	202.0	59.4	19	248.4	68.9	30	208.3	49.2	25	245.9	57.4	30	217.5	43	27	404.7	63.5
	Fal _{TK}	25	241.9	68.9	21	217.0	59.2	30	232.6	54.4	27	305.1	71	30	154.6	30.2	30	329.1	51.6
	Fal _{TNC}	25	187.8	54.5	16	248.2	68.8	30	173.6	40.6	29	252.3	58.3	30	180.2	34.6	28	443.1	70.4
	Fal _{TTK}	23	224.8	64.6	22	279.2	75.3	30	191.6	43.6	27	218.1	50.4	29	140.3	27.1	28	306.8	48.3
	Fal _{PD}	29	204.3	58.8	22	306.8	84.8	30	264.9	60.5	28	262.7	60.6	30	215.6	42.1	30	441.8	67.9
	Fal _{ND}	29	367.2	64.2	28	468.9	91.8	30	497.2	59.7	21	468.2	71.1	29	431.7	37.2	18	566.2	94.2
	Fal _{MI}	21	194.6	57.3	14	337.3	93.2	27	245.2	57.1	21	289.1	66.4	26	191.0	36.9	28	463.6	69.7
Fal _{MD}	30	247.9	71.0	26	259.0	60.5	29	226.5	52.7	26	259.0	60.5	27	197.2	38.6	22	288.0	44.0	
Spec.	Alg.	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ² _{ACC}	Br	0	-	-	3	37.2	18.0	0	-	-	4	48.6	16	0	-	-	25	118.0	26.4
	ST	9	284.4	71.1	2	150.0	37.5	2	105.0	21.0	7	306.3	76.6	6	490.1	91.7	6	261.5	37.3
	Fal _{InpSp}	20	179.8	55.0	18	256.4	75.0	26	226.8	56.0	27	202.6	49.0	11	324.5	68.0	27	352.6	53.3
	Fal _{NC}	29	202.3	63.3	16	314.6	92.6	29	225.1	56.3	27	242.4	60.1	26	402.0	84.5	27	399.3	63.4
	Fal _{TK}	25	230.4	72.0	18	282.9	81.4	30	235.0	59.0	26	177.6	43.5	21	358.4	75.6	30	321.3	54.0
	Fal _{TNC}	24	148.4	46.4	16	219.5	64.9	30	205.2	51.4	27	221.6	54.9	21	438.8	92.6	29	329.8	54.6
	Fal _{TTK}	27	225.1	70.0	25	299.5	87.9	30	236.9	60.3	28	237.7	58.7	24	445.7	92.0	30	306.9	51.4
	Fal _{PD}	27	230.3	71.9	12	268.4	77.9	27	234.8	59.7	29	237.0	56.6	21	421.5	88.4	28	339.1	56.3
	Fal _{ND}	24	364.5	67.0	20	379.7	78.1	25	379.7	47.7	26	322.8	50.4	13	479.5	78.8	22	240.2	39.9
	Fal _{MI}	28	285.0	87.6	19	284.5	80.8	29	254.6	62.5	28	219.0	52.4	17	360.4	74.2	23	358.1	55.2
Fal _{MD}	21	273.4	84.2	15	239.9	68.7	25	273.1	67.3	26	228.8	56.0	15	343.5	70.5	24	294.9	45.3	
Spec.	Alg.	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ¹ _{AFC}	Br	6	131.9	66.3	17	190.5	94.9	3	195.1	98	13	175.1	78.8	0	-	-	0	-	-
	ST	6	223.5	74.5	15	273.5	88.3	4	185.3	61.8	1	22.0	7.0	1	425.0	76.0	22	311.5	62.5
	Fal _{InpSp}	18	129.4	64.7	11	108.2	48.1	14	155.3	62.1	5	198.4	78.4	0	-	-	24	227.7	55.9
	Fal _{NC}	17	143.7	69.5	14	188.6	78.6	9	186.8	77.6	2	204.7	79.5	0	-	-	23	212.4	53.6
	Fal _{TK}	14	172.6	82.9	15	179.0	71.9	9	155.5	61.3	4	144.7	55	2	166.8	49	21	273.4	70.1
	Fal _{TNC}	13	91.9	42.8	17	179.7	71.9	9	160.7	67.1	6	270.9	108.2	0	-	-	18	263.5	68.0
	Fal _{TTK}	14	134.8	65.6	11	159.5	66.6	10	147.0	59	5	147.5	58.4	2	321.0	94	24	236.0	60.7
	Fal _{PD}	15	145.4	71.7	14	178.3	70.6	8	217.5	87.6	2	213.8	85.5	0	-	-	21	226.2	58.6
	Fal _{ND}	16	152.5	67.9	16	177.9	70.6	9	125.5	50.2	5	264.1	99	0	-	-	21	262.2	67.1
	Fal _{MI}	17	110.8	54.6	18	167.2	69.6	10	202.5	82.1	4	229.1	91.3	1	361.6	105	23	231.3	57.7
Fal _{MD}	18	121.0	59.8	17	172.6	68.5	10	155.3	62.5	3	181.9	70	0	-	-	17	211.8	51.7	
Spec.	Alg.	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ² _{AFC}	Br	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	1	50.8	17.0
	ST	3	249.0	83.0	5	151.1	47.4	1	69.0	23.0	0	-	-	0	-	-	11	281.0	54.5
	Fal _{InpSp}	2	207.9	104.0	10	176.6	85.0	0	-	-	2	361.3	144.0	0	-	-	17	337.2	81.4
	Fal _{NC}	1	280.0	140.0	10	134.6	67.3	0	-	-	5	267.4	107.0	0	-	-	14	227.2	56.0
	Fal _{TK}	4	167.0	83.5	6	178.0	89.0	0	-	-	1	222.2	89.0	1	432.2	134.0	14	274.4	66.7
	Fal _{TNC}	2	70.9	35.5	14	125.2	62.6	0	-	-	6	240.9	96.3	0	-	-	18	308.3	75.3
	Fal _{TTK}	2	70.0	35.0	8	114.6	55.4	0	-	-	1	102.7	41.0	0	-	-	15	250.4	60.8
	Fal _{PD}	1	164.0	82.0	6	163.6	81.8	0	-	-	4	159.4	63.8	0	-	-	16	291.3	70.8
	Fal _{ND}	1	184.0	92.0	7	127.9	54.9	0	-	-	3	83.4	33.3	0	-	-	22	312.6	76.4
	Fal _{MI}	1	234.0	117.0	5	176.8	84.6	1	94.1	41.0	1	102.3	41.0	0	-	-	19	293.2	72.2
Fal _{MD}	1	272.0	136.0	7	119.1	56.0	0	-	-	1	252.7	101.0	0	-	-	17	353.7	86.0	
Spec.	Alg.	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ³ _{AFC}	Br	0	-	-	1	45.8	23.0	0	-	-	1	31.8	16.0	0	-	-	0	-	-
	ST	0	-	-	4	271.0	89.8	0	-	-	0	-	-	0	-	-	17	407.0	83.0
	Fal _{InpSp}	0	-	-	14	132.2	61.0	0	-	-	1	347.7	139.0	0	-	-	13	282.8	69.2
	Fal _{NC}	0	-	-	8	168.9	75.6	1	34.2	15.0	2	222.5	89.0	0	-	-	17	219.6	53.2
	Fal _{TK}	0	-	-	12	215.5	97.8	0	-	-	2	277.4	111.0	0	-	-	18	339.9	80.1
	Fal _{TNC}	0	-	-	14	178.2	83.6	0	-	-	1	192.3	77.0	1	12.6	4.0	11	220.3	52.6
	Fal _{TTK}	0	-	-	8	136.4	63.1	0	-	-	2	280.0	107.0	0	-	-	16	220.3	53.8
	Fal _{PD}	0	-	-	10	185.3	84.9	0	-	-	0	-	-	0	-	-	20	245.0	59.4
	Fal _{ND}	0	-	-	10	193.9	77.6	0	-	-	1	180.0	72.0	0	-	-	16	259.6	62.2
	Fal _{MI}	2	242.5	119.5	12	152.1	65.2	0	-	-	0	-	-	0	-	-	18	309.9	75.7
Fal _{MD}	0	-	-	11	182.5	75.6	0	-	-	0	-	-	0	-	-	15	239.9	58.1	
Spec.	Alg.	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim	SR	time	#sim
S ¹ _{DPC}	Br	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-	2	74.6	2.0
	ST	7	31061.3	117.0	0	-	-	11	17305.8	65.6	1	29904.2	115.0	0	-	-	0	-	-
	Fal _{InpSp}	1	3309.6	127	2	506.9	26.5	14	2764.4	103.9	3	1999.6	88	2	1471.4	50.5	28	2470.4	57.8
	Fal _{NC}	4	2006.5	76.8	2	2101.4	108.5	19	1243.1	47.2	2	1687.9	76	1	1574.2	59	24	1968.6	46.0
	Fal _{TK}	5	1735.7	64.8	0	-	-	12	1617.3	63.1	0	-	-	3	2373.5	90.7	27	2275.0	52.6
	Fal _{TNC}	2	2392.7	90.5	6	1309.3	68.5	11	2103.5	80.4	1	2062.7	86	4	2027.3	75.8	27	2309.5	53.3
	Fal _{TTK}	6	2602.4	101.5	2	1357.2	69.5	15	2031.6	73.7	1	2240.6	98	1	3611.9	135	25	2467.7	58.6
	Fal _{PD}	6	2142.6	80.2	3	787.0	41	12	2183.5	82.3	0	-	-	4	2295.0	85	27	2283.1	51.8
	Fal _{ND}	3	2348.1	80.7	1	2634.3	138	18	1557.6	58.1	1	3357.0	143	2	2638.1	92.5	29	1792.8	41.7
	Fal _{MI}	4	2559.7	98.5	3	1264.1	65.7	14	2228.4	83.3	1	419.2	18	4	2742.1	98.3	28	2366.3	53.5
Fal _{MD}	5	2157.2	81.2	4	1580.3	77.5	19	1934.9	73.8	1	729.9	32	1	708.5	25	27	2611.1	61.9	

Table 5.4: RQ2 – Comparison of guidance of coverage criteria (number of benchmarks in which the approach on the row is statistically significantly better than the one on the column, using the Fisher’s exact test). The performance of a criterion is indicated by the degree of gray; a highlight in darker gray indicates a better result.

	Fal _{InpSp}	Fal _{NC}	Fal _{TK}	Fal _{TNC}	Fal _{TTK}	Fal _{PD}	Fal _{ND}	Fal _{MI}	Fal _{MD}	#benchs
Fal _{InpSp}	-	0	0	0	0	0	1	0	0	1
Fal _{NC}	2	-	0	0	0	0	2	1	2	3
Fal _{TK}	1	0	-	0	0	0	2	1	2	3
Fal _{TNC}	1	0	0	-	0	0	3	1	0	4
Fal _{TTK}	1	1	0	1	-	1	4	2	3	5
Fal _{PD}	1	0	0	0	0	-	2	2	1	4
Fal _{ND}	1	1	1	1	0	0	-	2	0	2
Fal _{MI}	1	0	0	0	0	0	1	-	1	2
Fal _{MD}	2	0	0	1	1	0	0	2	-	2

Answer to RQ1: FalsifAI significantly outperforms two state-of-the-art falsification methods that we take as baseline (Breach and S-TaLiRo), that do not include good exploration. However, FalsifAI is not better than Fal_{InpSp}, the other baseline approach which emphasizes exploration. The importance of exploration is confirmed by the fact that the Fal_{InpSp} is also better than Breach and S-TaLiRo.

RQ2 Effectiveness of Coverage in Guiding Falsification

In this RQ, we want to investigate which type of coverage provides better guidance for falsification. We perform a more fine-grained comparison between the different instantiations of FalsifAI with the different coverage criteria, and also with the baseline Fal_{InpSp} which is based on the exploration of the input space; namely, given two approaches and one benchmark, we apply the *Fisher’s exact test* and the *odds ratio* to their results as explained in §5.5.2. Table 5.4 reports, for each pair of approaches, for how many benchmarks the approach on the row is better than the approach on the column. The table also reports the number of individual benchmarks for which one approach performs better than at least another approach, by the column *#benchs*.

By considering *#benchs*, we observe that the baseline Fal_{InpSp} is not better than any of the

proposed coverage guided approaches, showing that, for DNN-controlled hybrid systems, the coverage of the DNN temporal behaviour is important to obtain diverse system behaviours, and thus allows to obtain a better falsification rate. Regarding the different coverage criteria, we observe that some are better than others:

- NC and TK are significantly better than at least another criterion for three benchmarks, showing that, for some benchmarks, the activation of some neurons at some point during the system execution is sufficient to guide the falsification search;
- we also notice that TNC and TTK are significantly better for four and five benchmarks, respectively (more benchmarks than NC and TK), showing that the continuous activation of the neurons (at some point of system execution) improves the search;
- also PD is better for four benchmarks; this could mean that, in some cases, the increment of a neuron activation over time can show a behaviour that is useful for falsification;
- some criteria provide worse overall results than the above mentioned criteria, but are still significantly better for some specific benchmarks. For example, ND is better than NC, TK, TNC, and MI on the same benchmark S_{ACC}^1 for ACC#2, meaning that the DNN behaviour required by ND makes it suitable to falsify this benchmark. Note that knowing in advance which criterion is better for which benchmark is in general not possible; we leave as future work further investigation on this point.

Answer to RQ2: For falsification of DNN-controlled hybrid systems, guidance provided by the proposed DNN coverage criteria is better than the one provided by the coverage of the input space. However, there is no best criterion in general, and different criteria are better for different benchmarks.

RQ3 *Coverage Achieved by Successful and Unsuccessful Falsification Trials*

In this RQ, we want to investigate whether there is a difference between the coverage obtained when an approach is able to falsify and when it is not. To this aim, we select the benchmarks

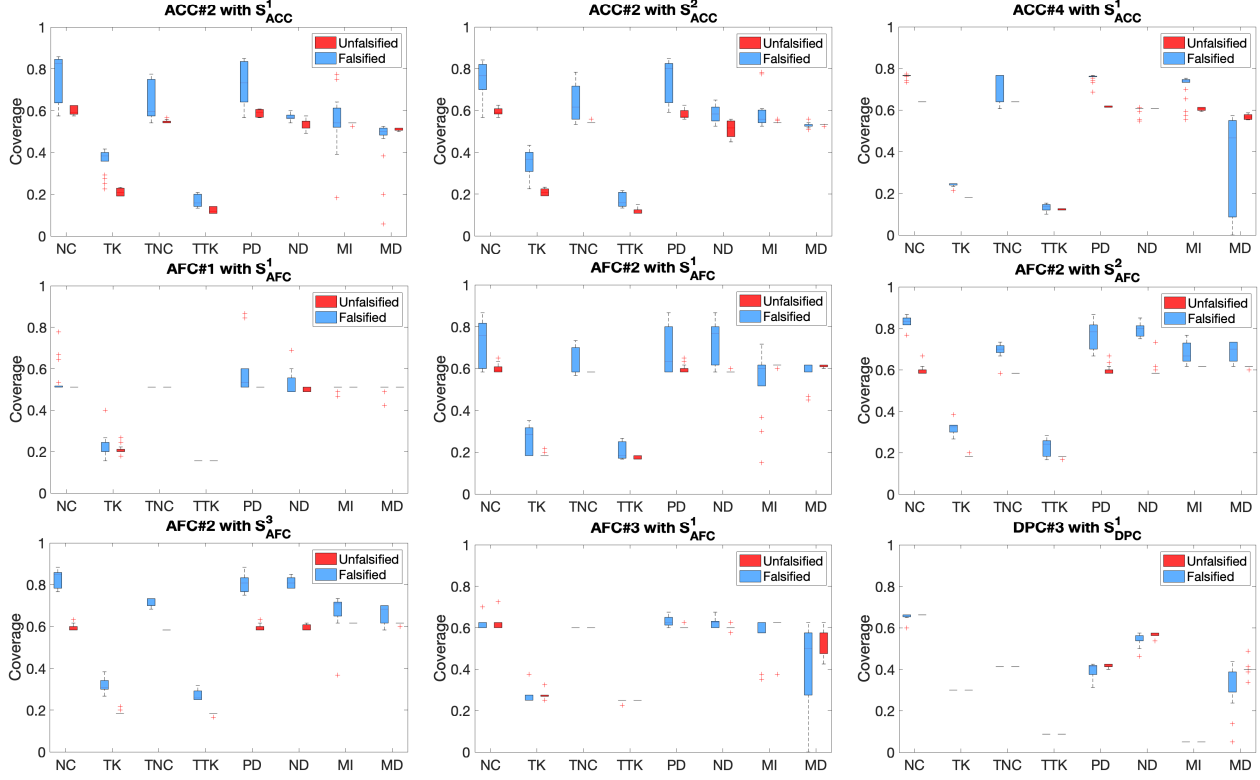


Figure 5.3: RQ3 – Coverage Achieved by Successful and Unsuccessful Falsification Trials

for which neither the number of falsified trials nor the number of unfalsified trials is too small, because they allow to observe the difference of coverage in the two cases.⁹ In our case, we select the benchmarks following this procedure: 1. we compute the difference between the number of successful trials and unsuccessful trials; 2. we compute the average number of difference over all criteria for each benchmark; 3. we select the ones that have an average difference less than 21. As a result, we select nine benchmarks, as shown in Fig. 5.3. The figure reports the coverage results of the selected benchmarks; namely, for each benchmark and each approach, it reports the distribution of coverage level of falsified and unfalsified trials (30 in total).

We observe that, overall, the coverage of falsified cases is higher than that of unfalsified cases, e.g., ACC#2 with S^1_{ACC} , ACC#2 with S^2_{ACC} , AFC#2 with S^2_{AFC} , AFC#2 with S^3_{AFC} , etc. This shows

⁹Benchmarks for which the number of falsified trials is high are too easy (and so any falsification approach is effective), while benchmarks for which the number falsified trials is very low or null are too difficult (and so any falsification approach struggles).

that increasing the coverage really allows to find different system behaviours that falsify the specification.

For some benchmarks and some approaches, instead, there is no big difference in coverage between falsified and unfalsified trials. Note that also these cases show the positive effect of coverage on the falsification search; indeed, falsified trials obtain the same coverage of unfalsified ones with less simulations (as they terminate earlier than the budget expiration), showing that the inputs that allow to improve the coverage are also those that allow to guide the search towards the falsifying input.

In some cases, such as MD in ACC#4 with S_{ACC}^1 , AFC#3 with S_{AFC}^1 , and DPC#3 with S_{DPC}^1 , and PD in DPC#3 with S_{DPC}^1 , the coverage of falsified attempts is even lower: this can be explained by the fact that when an approach finds a falsifying input, it stops, so it does not continue increasing the coverage as in the unfalsified cases that use the whole budget.

Answer to RQ3: Coverage level is usually higher in falsified trials than in unfalsified ones, and this seems to confirm that the coverage is helpful to guide the falsification. However, this does not always hold, because falsified trials may terminate early, while unfalsified trials always consume the whole budget.

RQ4 *Parameter Impact on Coverage Effectiveness*

Some of our proposed criteria are parametrized in terms of activation threshold h , and interval length I . Having different values for these parameters can make the criteria stronger/weaker, so also possibly affecting their ability to guide the falsification search. We here assess how falsification results change by changing the criteria parameters. Fig. 5.4 shows the SR results in ACC#2 with S_{ACC}^1 and AFC#1 with S_{AFC}^1 , for each of the eight criteria, initialized with different values for their parameters. We observe that there is indeed a quite strong effect of the parameters on the coverage criteria, that can lead to an SR difference of 15 (ND) and 13 (TK) between the best and the worst cases for ACC#2. The impact of parameters for AFC#1 is less evident—ND still

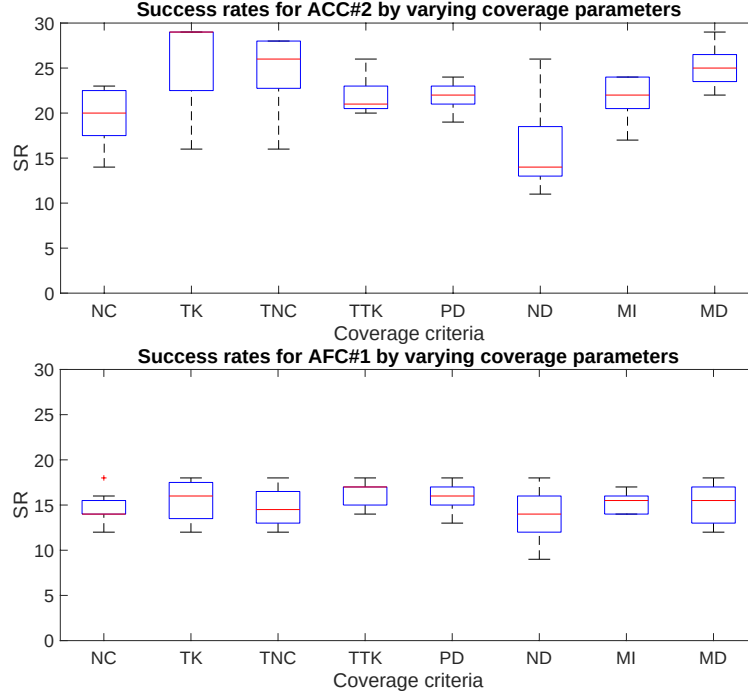


Figure 5.4: RQ4 – Parameter Impact on Coverage Effectiveness

observes the largest difference of 9 between the best and the worst case, while for other criteria, the difference is never larger than 6.

Answer to RQ4: There is a non-negligible effect of the criteria parameters on their effectiveness (in terms of SR) in guiding the falsification search, with a maximum variation of 15 between the best and the worst performance. A way to tackle this is to do parameter tuning, i.e., to make multiple parameter attempts on few samples, and select the best candidate on a given benchmark.

RQ5 Scalability w.r.t DNN Size

Since our proposed coverage criteria are computed based on the outputs of the neurons in the intermediate layers of neural network controllers, a potential concern is related to the scalability of the approach, i.e., whether the approach works effectively for systems that have neural network controllers of different sizes. In our benchmark sets, we have trained neural network

controllers of various sizes; specifically, ACC#6, AFC#6, and DPC#6 have the largest sizes over their respective systems. Note that, in general, the size of the neural networks cannot be unrestrictedly large, since that can lead to the overfitting problem in neural network training [87] (that would make them easily falsifiable). Our benchmarks ACC#6, AFC#6, and DPC#6 do not suffer from the overfitting problem, so they are not always easy to be falsified, as indicated by the results in Table 5.3. In this RQ, we investigate the influence of the benchmark size on the performance of the approach, from the following two perspectives.

First, we check the falsification results in Table 5.3 for the benchmarks ACC#6, AFC#6, DPC#6 that have the largest controller size among all the benchmarks. According to the results, we observe that, in 5 out of 6 benchmarks, versions of FalsifAI are the best performers; in the only one case where $\text{Fal}_{\text{InpSp}}$ is the best performer ($\text{SR} = 24$), Fal_{TTK} achieves the same success rate. This result shows that increasing the size of the three benchmarks does not diminish the performance of FalsifAI significantly; FalsifAI is still an effective approach for their testing.

Table 5.5: RQ5 – Comparison between time taken by simulation and coverage computation. Time in seconds

	ACC#1		ACC#2		ACC#3		ACC#4		ACC#5		ACC#6	
criterion	#sim	coverage	#sim	coverage	#sim	coverage	#sim	coverage	#sim	coverage	#sim	coverage
NC	1.88	0.006	1.44	0.007	2.11	0.008	1.57	0.009	2.38	0.01	3.56	0.006
TK	1.7	0.016	1.34	0.02	2.07	0.021	1.65	0.024	2.52	0.026	3.63	0.018
TNC	1.66	0.013	1.33	0.016	2.13	0.017	1.57	0.021	2.56	0.02	3.94	0.015
TTK	1.7	0.024	1.4	0.028	2.14	0.032	1.69	0.036	2.45	0.037	3.6	0.03
PD	1.72	0.019	1.46	0.016	2.17	0.024	1.59	0.02	2.43	0.031	3.55	0.043
ND	1.64	0.018	1.47	0.016	2.07	0.023	1.71	0.019	2.35	0.031	3.56	0.042
MI	1.56	0.011	1.4	0.012	2	0.016	1.69	0.016	2.69	0.02	3.84	0.019
MD	1.66	0.013	1.47	0.013	2.08	0.017	1.66	0.016	2.47	0.021	3.56	0.025
Aver.	1.69	0.015	1.41	0.016	2.10	0.020	1.64	0.020	2.48	0.025	3.65	0.025

Second, to further understand the overhead introduced by coverage computation, we record the time taken by system simulation and coverage computation, respectively, for the six benchmarks of ACC that have from 120 to 390 neurons in their DNNs; we selected ACC because it is the

system that has the largest DNN among all the benchmarks. To account for the variability of time of coverage computation and system simulation, we run each of the ACC benchmarks, with each coverage criterion, for 50 simulations, and then report the average time. The data are presented in Table 5.5.

Based on Table 5.5, we conclude that, 1. The time consumed by coverage computation is almost negligible compared to the time consumed by system simulation; 2. The time consumption for coverage computation slightly increases with the size of the DNN controllers; however, even for the largest DNN controller, it is still extremely small compared to the time consumption for simulation.

Answer to RQ5: FalsifAI performs effectively for benchmarks of different sizes. The time spent for coverage computation is negligible compared to that for system simulation.

5.7 THREATS TO VALIDITY

Some threats may affect the validity of the proposed approach. We discuss them, using the classical classification of *construct*, *conclusion*, *internal*, and *external validity* [151].

Construct validity It could be that the metrics we use to evaluate the approach do not reflect the object of the investigation, that is, in our case, the guidance provided by coverage criteria to the falsification search. The used metric SR measures the number of trials (out of 30) in which the proposed approach is able to find a falsifying input; moreover, the proposed approach has been compared against a state-of-the-art approach not guided by coverage, and an approach guided by a different type of coverage (of the input space instead of the DNN). In this way, the used metrics are able to assess the ability of the criteria to expose failures; a similar correlation is investigated when assessing classical coverage criteria [27]. Therefore, we believe that the used metrics are suitable.

Conclusion validity It could be that different factors could affect the ability to draw definitive

conclusions. For example, the random behaviour of the search algorithms must be considered during the evaluation. So, each experiment has been executed for 30 trials, as suggested by Arcuri and Briand [12] in the guideline on conducting experiments with randomised algorithms. Moreover, results have also been compared using suitable statistical tests that account both for statistical significance and effect size, still following [12].

Internal validity We could identify a causal relationship between the usage of a criterion and the improvement in falsification, but this is determined by other factors.

For example, it could be that the improvement in falsification is simply due to more exploration and not specifically to the proposed criteria. To mitigate this threat, in RQ1 and in RQ2, we have compared our coverage-based approach with an approach that achieves more exploration by random search. Results of RQ1 indeed show that having more exploration is in general better; however, results from RQ2 show that the coverage provided by our proposed coverage criteria allows to obtain better falsification results than random exploration.

External validity One such a threat is that the improvement demonstrated by FalsifAI on the 36 benchmarks, does not apply to other case studies. To mitigate such a threat, we have carefully selected benchmarks commonly used in the falsification community [46, 48] and in the community of AI-enabled control systems [74, 75], and we have trained different types of network, so to model different types of systems (of good and bad quality).

Moreover, the training methods may also be a threat that affects the effectiveness of the approach. For instance, regularization methods are usually used to solve the overfitting problem if the sizes of the DNN controllers are selected too large [87]. In that case, the neuron outputs may be so small that the computed coverage becomes less meaningful. We leave as future work the investigation of the influence of the training method on the effectiveness of the approach.

5.8 CONCLUSION AND FUTURE WORK

AI-Enabled Hybrid Control Systems are emerging as a new class of systems controlled by DNNs. Classical falsification usually applied to CPSs, does not perform effectively for this kind of systems, because it is guided by STL robustness only, which is not sufficiently informative to characterize the temporal behaviour of DNN controllers due to their consecutive executions in the hybrid control systems. So, in this paper, we have proposed a set of time-aware coverage criteria for DNNs, and we have used these for guiding falsification. Results show that the approach is more effective than the state-of-the-art falsification methods that do not consider DNN coverage, and of an approach that is guided by the coverage of the input space.

The proposed coverage criteria require the setting of some parameters that can affect their performance; in this paper, we have selected them as described in §5.5.4, with the aim of providing a good guidance during falsification, considering the overall allocated budget. However, the cost of such parameter tuning is non-negligible, and may not always be applicable by a user. Therefore, as future work, we plan to devise cheaper techniques to properly tune them, possibly adjusting them at runtime during falsification.

Experimental results (RQ2) show that different coverage criteria allow to obtain better guidance for different benchmarks. However, properly characterizing which criterion is better for which type of system is difficult to assess, as it would require a much larger benchmark set that is difficult to obtain. We leave this type of investigation as future work.

We also consider extending the framework to other types of neural networks, e.g., recurrent neural networks (RNN). Unlike the NN used in this work which is stateless, RNN has a mechanism of storing the internal states and use them during a sequence of inferences. Therefore, in the case of RNN, we can consider defining their time-aware coverage criteria w.r.t. the internal states and use those criteria for test case generation.

6 | SPECTACLE: FAULT LOCALISATION OF AI-ENABLED CPS BY EXPLOITING SEQUENCES OF DNN CONTROLLER INFERENCES

In this chapter, we propose SPECTACLE, which can localise the faulty parameters in DNN controllers. SPECTACLE considers the DNN inferences preceding the specification violation and uses forward impact to determine the DNN parameters that are more relevant to the DNN outputs. Then, it identifies which of these parameters are responsible for the specification violation, by adapting classic suspiciousness metrics. Moreover, we propose two versions of SPECTACLE, that consider differently the timestamps that precede the specification violation. We experimentally evaluate the effectiveness of SPECTACLE on 554 faulty benchmarks, spanning over different application domains. The results show that SPECTACLE can detect most of the faults.

6.1 INTRODUCTION

A *Cyber-physical system* (CPS) is given by the combination of a physical system (*plant*) and a computational component (*controller*), where the controller monitors and controls the plant

according to the physical states of the plant and external environments. Recently, *Deep Neural Networks* (DNNs) have been increasingly used as controllers to perform complex control tasks (e.g., *self-driving* [24] and *robotic arms control* [62]); indeed, they have shown to be able to handle intricate environments more efficiently than classic controllers. Such CPSs driven by AI-based controllers are called as *AI-enabled CPSs* [69, 74, 75, 108, 109, 143, 144, 176].

Despite the great potential in terms of control, the safety of AI-enabled CPSs becomes a major concern. Indeed, a DNN controller may produce unforeseen and inexplicable behaviours that lead the plant to malfunctions and catastrophic hazards, which can further bring irreversible social and economic losses. To avoid these dangerous situations, it is necessary to detect the potential unsafe behaviours of DNN controllers and identify the root causes that produce them.

There have been extensive studies on generating unsafe system executions for these systems, i.e., finding a *time-variant* external input signal from the environment that leads the system execution to violating a temporal-logic specification, mainly from the testing and falsification communities [37, 49, 80, 114, 134, 159, 176]. However, these works mostly demonstrate the presence of unsafe behaviours, without delving into a comprehensive investigation of these behaviours. To assist system engineers in enhancing system safety, it is necessary to identify the *root cause* of the violation: since the decision logic of the AI-enabled system is embedded in the DNN controller, this boils down to localising the faults in the DNN, specifically in the configuration of its parameters (e.g., weights).

However, precisely identifying the model parameters of DNN controllers that are responsible for the violations is a challenging problem. This is due to the unexplainable programming logic of DNNs, which makes it difficult to properly assess which parameters inside a DNN controller are responsible for a given (wrong) decision. Some *fault localisation* approaches have been proposed for DNN classifiers [26, 43, 44, 133, 149]; such approaches consider different inferences of a DNN and compare the outcome with the *ground truth* (e.g., the expected label in classification). However, such approaches are not applicable for fault localisation of DNN controllers, due to

two main issues:

- (1) There is no explicit ground truth for the behaviours of DNN controllers, and the correctness of their behaviours can only be assessed by considering the system level specification. Specifically, this requires executing the whole system controlled by the DNN controller for a time period, and then checking whether the system outputs satisfy the temporal-logic specification.
- (2) In an AI-enabled CPS, a system violation is due to a *sequence of control decisions* of the DNN controller, i.e., a sequence of inferences. Hence, fault localisation must analyze sequences of DNN inferences, rather than single ones as done by state-of-the-art fault localisation approaches in the context of other domains such as image classification.

Contributions

In this paper, we propose SPECTACLE (*SPECTrum-based fault localisation for Ai ControllERs*), a fault localisation approach for AI-enabled CPSs. Given a *test suite* (i.e., a set of input signals) and a *system specification*, SPECTACLE executes each test in the test suite and obtains: (i) in case of a failing test, the sequence of DNN controller inferences preceding the start of the *violation episode* (i.e., the timestamp when the specification is shown to be violated), (ii) the whole sequence of inferences for a passing test. For each inference in each test, SPECTACLE assesses the *relevance* of DNN weights on the control decision by calculating their *forward impacts*. It then constructs the *execution spectrum* for each DNN weight, based on the status of test executions (i.e., passed or failed) and relevance of the DNN weight to the control decisions. Finally, it adapts existing suspiciousness metrics from *spectrum-based fault localisation (SBFL)* [155] to identify the DNN weights that are more likely responsible for the specification violation.

We propose two versions of SPECTACLE, distinguished by the significance assigned to the DNN inferences that may account for the violation, based on their distances to the beginning of

specification violation: (i) *unweighted* SPECTACLE ($\text{SPECTACLE}_{\text{UW}}$) considers equally all the DNN inferences before the violation; (ii) *weighted* SPECTACLE ($\text{SPECTACLE}_{\text{W}}$) considers more relevant the inferences closer to the violation, and so weights them more in the construction of the execution spectrum.

We experimentally evaluate the effectiveness of SPECTACLE on 6067 faulty benchmarks, obtained by injecting artificial faults into the DNN weights of 13 correct AI-enabled CPSs taken from existing literature [114, 134, 143, 176]. The results show that SPECTACLE is able to report most of the artificial faults, which demonstrates its effectiveness.

In summary, the main contributions of this paper are:

- We propose SPECTACLE, a fault localisation framework for AI-enabled CPS that aims to identify the faulty weights of DNN controllers, by exploiting the sequences of DNN inferences. Specifically, we provide two versions of SPECTACLE: $\text{SPECTACLE}_{\text{UW}}$ gives equal importance to the inferences of each timestamp before the violation episode, while $\text{SPECTACLE}_{\text{W}}$ weights more the inferences that are closer to the violation episode.
- We implement both versions of SPECTACLE, which are available online [100].
- We perform experimental evaluation on 6067 faulty versions of commonly-used benchmarks of AI-enabled CPSs, controlled by DNN controllers with various structures and configurations. The results demonstrate the effectiveness of SPECTACLE in identifying the faulty weights of the DNN controllers.

6.2 PROBLEM DEFINITION

6.2.1 AI-ENABLED CPS

Fig. 6.1 illustrates the architecture of a typical AI-enabled CPS. The whole system $\mathcal{M}^C$ is

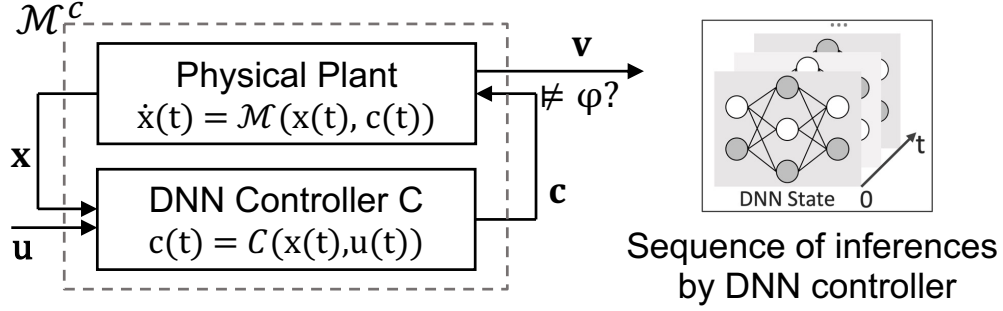


Figure 6.1: The architecture of a typical AI-enabled CPS

composed of a physical plant $\mathcal{M}$ and a DNN controller $\mathcal{C}$. At each timestamp t , the DNN controller $\mathcal{C}$ takes as input the system state $\mathbf{x}(t)$ and the external input $\mathbf{u}(t)$, and produces a control decision $\mathbf{c}(t)$ to trigger the system state evolution to the next timestamp t' . In this way, the system state $\mathbf{x}$ and the control decision $\mathbf{c}$ are both *time-variant signals*, mapping each timestamp to a vector. In particular, some of the system variables are observable from the outside, and the evolution of these variables produces another time-variant signal $\mathbf{v}$, as the *system output*.

In this loop, the DNN controller is the primary component that decides the evolution of the system. In this paper, we consider fully-connected DNNs, defined as follows.

Definition 3 (DNN controller $\mathcal{C}$). A DNN $\mathcal{C}$ includes an *input* layer, an *output* layer, and multiple *hidden* layers. At each timestamp, $\mathcal{C}$ takes as input a vector (jointly constituted by system state and external input) at the input layer, and computes an output scalar as the control decision at the output layer. Each hidden layer consists of a number of neurons; specifically, at hidden layer L_i ($i \in \{1, \dots, L\}$), there are s_i neurons (where s_i is a positive integer). Each neuron has a set of parameters, including a vector of weights and a bias. We elaborate on the numerical transformations in hidden layers. Formally, the hidden layer L_i takes an s_{i-1} -dimensional vector $\vec{O}_{i-1}$ as the input of the layer, and gives an s_i -dimensional vector $\vec{O}_i$ as the output of the layer.¹

¹Specifically, s_0 is the dimension of the input layer, and $\vec{O}_0$ is the input of the DNN from the input layer.

The transformation from $\vec{O}_{i-1}$ to $\vec{O}_i$ is given as follows:

$$\vec{O}_i = \sigma \left(W \vec{O}_{i-1} + \vec{b}_i \right) \quad (6.1)$$

where $W \in \mathbb{R}^{s_{i-1} \times s_i}$ is a matrix of *neuron weights* and $\vec{b}_i \in \mathbb{R}^{s_i}$ is a vector of *neuron bias* at layer L_i , and σ is an *activation function* that performs non-linear transformations. Common choices of activation functions include *ReLU*, *Sigmoid*, *tanh*, etc.

The output layer computes the DNN output O_c , by applying a linear transformation to $\vec{O}_l$, the output vector of the last hidden layer, which is then used as the control decision. $\triangleleft$

In Eq. 6.1, the computation from $\vec{O}_{i-1}$ to $\vec{O}_i$ consists of a linear transformation empowered by *weights* and *biases*, and a non-linear transformation based on the activation function. In this paper, since the numerical transformation of each layer is primarily decided by the neuron weights, we follow the existing literature [25, 88, 133, 142] in DNN fault localisation and repair, and treat these neuron weights as the main target of our proposed fault localisation approach.

To identify a neuron weight in a DNN, we introduce the following notation. We use $W_{(i-1,j)(i,p)}$ ($i \in \{1, \dots, L\}$, $j \in \{1, \dots, s_{i-1}\}$ and $p \in \{1, \dots, s_i\}$) to identify the neuron weight that has the index (p, j) in the weight matrix at the layer L_i (remember that s_i is the number of neurons at layer L_i). Intuitively, $W_{(i-1,j)(i,p)}$ is located at the p -th neuron at layer L_i , used to multiply the output of the j -th neuron at layer L_{i-1} , i.e., $(i-1, j)$ and (i, p) respectively indicate the locations of the two neurons connected by the weight $W_{(i-1,j)(i,p)}$.

6.3 MOTIVATION

In this work, since the DNN controller is the primary component that decides the evolution of a system, we assume that the system executions that violate the specification should be attributed to erroneous control decisions given by the DNN controller. Consequently, we take DNN con-

trollers as the target of our fault localisation.

For single-inference DNNs, fault localisation approaches have been successfully applied to identify the causes of wrong DNN inferences. Eniser et al. [44] proposed DeepFault, which leverages SBFL to construct execution spectra of neurons and utilizes existing suspiciousness metrics to identify the suspicious neurons responsible for the misclassification of DNN. Sohn et al. [133], in their DNN repair tool Arachne, proposed a bidirectional localisation approach to identify the faulty weights, which uses both *gradient losses* and *forward impacts* of each weight to characterize the relevance of the weight to the final DNN output. However, these approaches cannot be applied in the context of AI-enabled CPS. This is due to two main issues:

- Both DeepFault and Arachne require a notion of *correctness* of a DNN inference, which relies on the ground truth labels of DNN inputs. However, in an AI-enabled CPS, it is not possible to determine whether a single inference of the DNN controller is correct or not, i.e., there is no such an explicit ground truth. To assess the correctness of DNN inferences, we can only rely on the *effect* they have on the satisfaction of the system specification (see §2.2), i.e., whether they provide control decisions that guarantee the specification satisfaction of system executions. In this sense, the satisfaction of the system specification provides an *indirect* ground truth regarding the correctness of the inference of the DNN controller.
- Moreover, while DeepFault and Arachne only consider individual DNN inferences, the DNN controller of an AI-enabled CPS performs a “sequence of inferences” (as shown by Fig. 6.1) that drive the evolution of the plant. These inferences must be considered together in fault localisation.

Example 1. Consider the AI-enabled *Adaptive Cruise Control* (ACC) system introduced in §6.5.2 where the DNN controller C produces the acceleration a_{ego} that determines the motion of the ego car and therefore affects the relative distance d_{rel} between the ego car and the lead car. The system specification φ requires that d_{rel} should always be greater than the safe distance $d_{safe} = 10$,

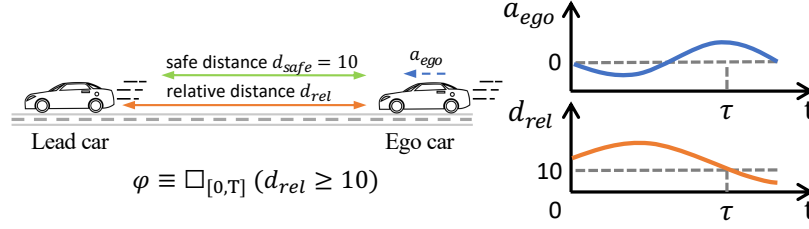


Figure 6.2: Example – Adaptive Cruise Control system that violates the specification $\varphi \equiv \square_{[0,T]} (d_{rel} \geq 10)$

during the time interval $[0, T]$; formally, $\varphi \equiv \square_{[0,T]} (d_{rel} \geq 10)$. While we cannot tell whether a given acceleration command is correct or not, we can check how it affects the relative distance d_{rel} : if this violates the system specification, the acceleration command can be deemed not correct. Fig. 6.2 shows examples of execution traces of a_{ego} and d_{rel} . τ is the timestamp that identifies the beginning of the violation; therefore, the control decisions that lead to the violation must precede τ . Given the inference process of DNN (as introduced in Def. 3), these erroneous control decisions originate from a number of suboptimal DNN parameters, and therefore, our primary focus in fault localisation is to identify these parameters that account for the specification violation. While the exact timestamps at which the controller makes incorrect decisions are not known, it is feasible to assess the impacts of each parameter to a control decision and thereby select the parameters that exert the greatest impact on the control decisions. These parameters can be deemed to be more relevant to the erroneous control decisions, hence they can be further considered as more responsible for the specification violation of system executions. $\triangleleft$

Problem statement. In this paper, we consider an AI-enabled CPS that violates a specification, and the specification violation is due to erroneous control decisions made by the DNN controller. These erroneous decisions result from suboptimal settings of some DNN *parameters* that significantly influence the DNN inferences, leading to the generation of erroneous decisions. Specifically, we consider *neuron weights* (see Def. 3) as our target parameters, following existing literature [133] on DNN classifiers, as these neuron weights can be directly modified by using DNN repair approaches [25, 101, 133] that take as input fault localisation results for DNN repair.

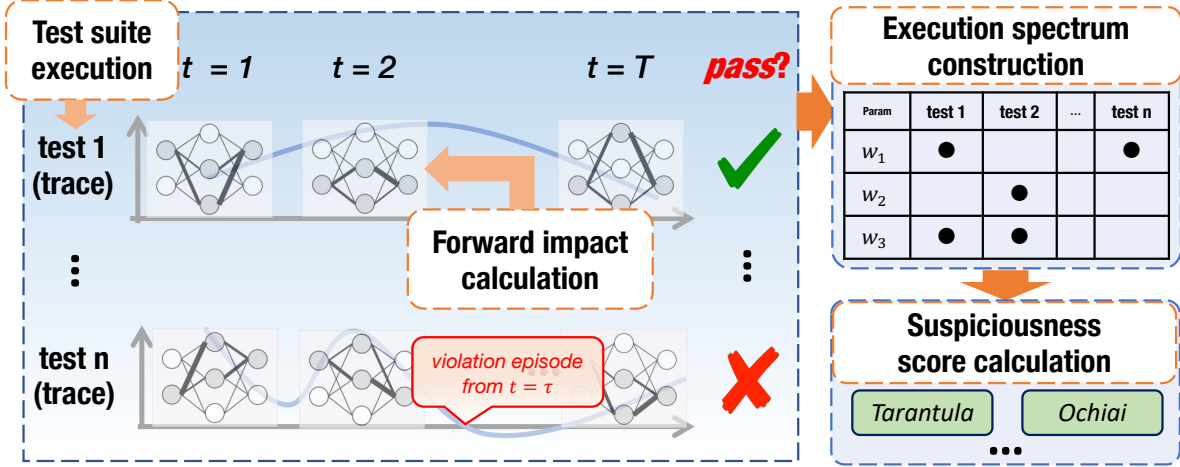


Figure 6.3: SPECTACLE – Workflow of the approach

6.4 SPECTACLE

We assume to have an AI-enabled CPS $\mathcal{M}^C$ that violates a system specification φ ; the violation has been determined by a test suite TS of input signals $\mathbf{u}$, each of which is executed for an execution time T (in practice, the time horizon T of signals can be decided based on the specification φ , such that the satisfaction of φ can be assessed (see Def. 1 and Def. 2)).

We propose that fault localisation approach SPECTACLE, whose aim is to detect the *cause* of the specification violation in the DNN controllers. Fig. 6.3 provides an overview of SPECTACLE. The approach consists of four phases:

- (i) *Test suite execution:* it monitors the system executions of the tests in TS and obtains the sequences of DNN controller inferences before the specification violation for the failing tests (as test n in Fig. 6.3), or the whole execution if no violation occurs (as test 1 in Fig. 6.3).
- (ii) *Forward impact calculation:* based on the system executions for the tests, it calculates the *forward impact* of each DNN weight, which indicates the relevance of a DNN weight to the final DNN output (i.e., a control decision).
- (iii) *Execution spectrum construction:* by using the values of the forward impact in passing and

failing tests, it builds a *spectrum* for each weight w , which tells how often w was *relevant* in passing and failing tests.

- (iv) *Suspiciousness score calculation*: starting from the spectra of the different weights, it uses suspiciousness metrics borrowed from *spectrum-based fault localisation (SBFL)* to identify the weights that are more likely responsible for the specification violations.

Alg. 4 presents the four phases of SPECTACLE.

The algorithm takes as input an AI-enabled CPS $\mathcal{M}^C$, a system specification φ , an execution time T , a test suite TS of input signals $\mathbf{u}$ for $\mathcal{M}^C$, the number of top k weights to consider in each hidden layer, the number s of suspicious weights to return in the final results, and a suspiciousness metric SusMetr .

The following sections describe in detail the four phases of SPECTACLE.

6.4.1 TEST SUITE EXECUTION

SPECTACLE first initialises an empty execution spectrum ES_w with four attributes $\langle ep, np, ef, nf \rangle$ for each weight w of the DNN controller C (Lines 1-3). The meaning of these attributes and how these attributes are updated will be described in *Execution spectrum construction* in §6.4.3.

Then, it takes each test case $\mathbf{u}$ in the test suite TS (Line 4), and feeds it to the system $\mathcal{M}^C$ for system execution (Line 5). The execution returns the system output $\mathbf{v}$ and the sequence InfSeq of inferences of the DNN controller C . Each infer_t in InfSeq contains the information regarding a single DNN inference at time t : the DNN input, the output $O_{i,j}$ of each neuron $n_{i,j}$, and the final output O_c of the DNN that is used as the control action $\mathbf{c}(t)$ for the plant $\mathcal{M}$. The neuron outputs and the final DNN output are used for the forward impact calculations (see §6.4.2).

Given the output signal $\mathbf{v}$, it assesses whether the specification φ is satisfied or violated (Line 6). Moreover, in case of violation, it analyses the execution trace and obtains the timestamp τ when the system output $\mathbf{v}$ starts to violate the specification; the DNN controller must

Algorithm 4 SPECTACLE – The proposed approach

Require: An AI-enabled CPS $\mathcal{M}^C$ with a DNN controller C ; a set $\mathcal{W}$ of neuron weights (subset at hidden layers L_i denoted as W_i); a system specification φ ; maximum execution time T ; a test suite TS ; number k of top weights to consider in each hidden layer; number s of weights to return as suspicious; a suspiciousness metric SusMetr .

Ensure: A set of suspicious weights Θ_s .

```

1: for all  $W_i \in \{W_1, \dots, W_L\}$  do
2:   for all  $w \in W_i$  do
3:      $ES_w \leftarrow \langle 0, 0, 0, 0 \rangle$  ▷ Initialize spectrum for each weight
4:   for all  $u \in TS$  do
5:      $\langle \mathbf{v}, \text{InfSeq} \rangle \leftarrow \mathcal{M}^C(u)$  ▷ System execution
6:     Identify  $\tau$  as:
       
$$\tau = \begin{cases} \text{Start of violation episode, if } \mathbf{v} \not\models \varphi \\ \text{Execution time } T, \text{ otherwise.} \end{cases}$$

7:      $\beta \leftarrow \frac{T}{\int_0^\tau \Omega(t) dt}$  ▷ Compute scaling factor
8:     for all  $t \in [0, \tau]$  do
9:        $\text{infer}_t \leftarrow \text{InfSeq}(t)$  ▷ Inference at timestamp  $t$ 
10:      for all  $W_i \in \{W_1, \dots, W_L\}$  do
11:        for all  $w \in W_i$  do
12:           $I_w \leftarrow \text{CALCULATEFORWARDIMPACT}(w, \text{infer}_t)$ 
13:           $\text{top}_k^i \leftarrow \text{Sort} \{I_w \mid w \in W_i\}$  ▷ Select top  $k$  weights
14:          for all  $w \in W_i$  do
15:            Update  $ES_w$  based on  $\mathbf{v}$ ,  $\varphi$ ,  $w \in \text{top}_k^i$ , and  $\beta$ :
              
$$\begin{aligned}
ES_w(ep) &\leftarrow ES_w(ep) + 1 && \text{if } \mathbf{v} \models \varphi \wedge w \in \text{top}_k^i, \\
ES_w(np) &\leftarrow ES_w(np) + 1 && \text{if } \mathbf{v} \models \varphi \wedge w \notin \text{top}_k^i, \\
ES_w(ef) &\leftarrow ES_w(ef) + \beta \cdot \Omega(t) && \text{if } \mathbf{v} \not\models \varphi \wedge w \in \text{top}_k^i, \\
ES_w(nf) &\leftarrow ES_w(nf) + \beta \cdot \Omega(t) && \text{if } \mathbf{v} \not\models \varphi \wedge w \notin \text{top}_k^i.
\end{aligned}$$

16:   for all  $w \in \mathcal{W}$  do
17:      $\text{score}_w \leftarrow \text{CALCULATESUSPICIOUSNESS}_{\text{SusMetr}}(ES_w)$ 
18:    $\Theta_s \leftarrow \text{Sort} \{\text{score}_w \mid w \in \mathcal{W}\}$  ▷ Select top  $s$  suspicious weights

```

have done some erroneous decisions before the timestamp τ , i.e., in $[0, \tau]$. For example, in Example 1, since d_{rel} starts to be below the safe distance 10 at τ , τ is then the first timestamp when we start to observe the violation of the specification. If the system execution satisfies the spec-

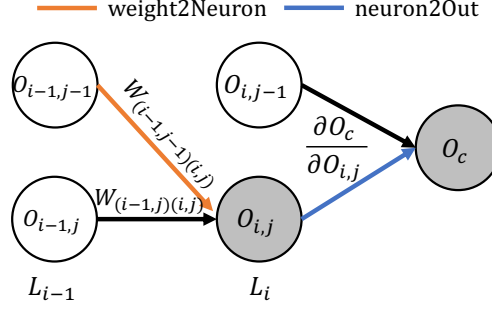


Figure 6.4: Forward impact of $W_{(i-1,j-1)(i,j)}$ to the final DNN output O_c

ification, τ is set as the simulation time T . In order to identify τ , we use the technique of *trace diagnostics* proposed by Bartocci et al. [18].

6.4.2 FORWARD IMPACT CALCULATION

In the interval $[0, \tau]$, the DNN controller performs a sequence of inferences to determine the control decision at each timestamp, which leads to the satisfaction or violation of the specification. SPECTACLE, for each timestamp $t \in [0, \tau]$ (Line 8), computes the *forward impact* of each DNN weight in this interval to determine its contribution to the DNN output, namely, the control decision at t (Lines 9-12). The calculation follows the approach proposed by Sohn et al. [133]; see an illustration of the calculation in Fig. 6.4. In order for a weight $W_{(i-1,j-1)(i,j)}$ to have an impact on the control decision of a single inference infer_t , the following conditions must occur: (i) the weight must affect the linear output of its associated neuron $n_{i,j}$; (ii) the neuron output $O_{i,j}$ of its associated neuron $n_{i,j}$ should contribute to the final DNN output of the current inference. Hence, the forward impact of a weight on an inference is given by two key components:

- *weight2Neuron* of a weight $W_{(i-1,j-1)(i,j)}$ is the impact on the output of its neuron $n_{i,j}$, formally:

$$\text{weight2Neuron} = \frac{W_{(i-1,j-1)(i,j)} \times O_{i-1,j-1}}{\sum_{m=1}^{s_{i-1}} |W_{(i-1,m)(i,j)} \times O_{i-1,m}|}$$

i.e., the ratio of the product of the weight and the output value of the neuron in the previous

layer it is connected to, over the sum of the absolute value of the element-wise product between the output values of the neurons in the previous layer and the corresponding weights of the current neuron. Intuitively, it determines the relative contribution of weight $W_{(i-1,j-1)(i,j)}$ to the neuron output $O_{i,j}$ of neuron $n_{i,j}$.

- *neuron2Out* is the impact of the output $O_{i,j}$ of neuron $n_{i,j}$ on the final DNN output O_c , i.e., the control decision of the DNN controller, formally:

$$neuron2Out = \frac{\partial O_c}{\partial O_{i,j}}$$

i.e., the gradient of O_c with respect to the neuron output $O_{i,j}$.

The forward impact $I_{W_{(i-1,j-1)(i,j)}}$ of a weight $W_{(i-1,j-1)(i,j)}$ on the control decision can be represented as the absolute value of the multiplication of the two elements, namely:

$$I_{W_{(i-1,j-1)(i,j)}} = |weight2Neuron \times neuron2Out|$$

6.4.3 EXECUTION SPECTRUM CONSTRUCTION

After the computation of the forward impact, SPECTACLE constructs an *execution spectrum* $ES_w = \langle ep, np, ef, nf \rangle$ for each weight w , that represents the contributions of w to the satisfaction and violation of system executions. The execution spectrum considers both the specification satisfaction/violation, and the contributions of w to the system execution identified by its forward impact (see §6.4.2); these elements are reflected in the naming of the spectrum components:

- p (for *pass*) and f (for *fail*) denote whether a system execution satisfies the system specification or not;
- e (for *executed*) and n (for *not executed*) denote whether the weight is *executed* or not, i.e., if it contributes to the satisfaction/violation of the system execution. It decides this as

follows: it sorts the weights of each hidden layer L_i in a descending order according to their forward impact (Line 13); if a weight w is ranked in the top k greatest weights (labeled as top_k^i) in layer L_i , then w will be deemed as *executed*.

For example, attribute ep records the number of times a weight is ranked in the top k (thus *executed*) and its corresponding inference is situated in a “safe” system execution (thus *pass*).

So, in Alg. 4, for each weight w , the corresponding attribute in $\langle ep, np, ef, nf \rangle$ is incremented (Line 15) after one system execution:

- if $v \models \varphi \wedge w \in top_k^i$, i.e., the specification is satisfied and w is ranked in the top k , ep is incremented;
- if $v \models \varphi \wedge w \notin top_k^i$, i.e., the specification is satisfied and w is not ranked in the top k , np is incremented;
- if $v \not\models \varphi \wedge w \in top_k^i$, i.e., the specification is violated and w is ranked in the top k , ef is incremented;
- if $v \not\models \varphi \wedge w \notin top_k^i$, i.e., the specification is violated and w is not ranked in the top k , nf is incremented.

In the update of the spectrum components (Line 15), a spectrum component is incremented by a quantity $\Omega(t)$ called *window*, that depends on the time when the inference occurs, and it is scaled by a factor β that depends on the length of the test (namely, the time horizon T of the signal). In the following, §6.4.3.1 provides two versions of Ω , and §6.4.3.2 explains the scaling factor computation.

6.4.3.1 TWO APPROACHES FOR SPECTRUM CONSTRUCTION

Recall from §6.4.1 that the timestamps when the DNN controller makes erroneous decisions are before the timestamp τ that identifies the beginning of the specification violation, and, there-

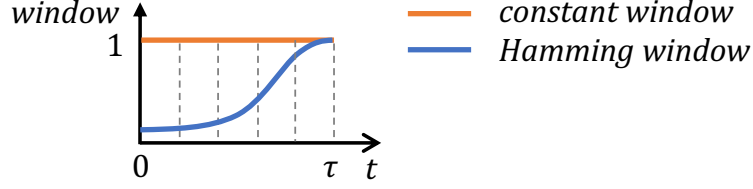


Figure 6.5: Window functions

fore, fault localisation should focus on the weights that are relevant in the time interval $[0, \tau]$. However, it is difficult to exactly determine *when* the wrong decisions are made, and so decide which timestamps should be considered in fault localisation. To this end, we propose two versions of SPECTACLE ($\text{SPECTACLE}_{\text{UW}}$ and $\text{SPECTACLE}_{\text{W}}$) that make different assumptions on which timestamps should be considered. This is reflected in the definition of a window function Ω used in spectrum construction (see Line 15), that increments the spectrum depending on the specific timestamp:

$$\Omega(t) = \begin{cases} 1 & \text{in } \text{SPECTACLE}_{\text{UW}} \\ \Omega_{\text{ham}}(t) & \text{in } \text{SPECTACLE}_{\text{W}} \end{cases}$$

$\text{SPECTACLE}_{\text{UW}}$ considers in the same way all the relevant weights of all the timestamps (it uses a *constant window*), while $\text{SPECTACLE}_{\text{W}}$ weights more the neuron weights that were relevant closer to the system violation (it uses a *Hamming window*), as explained in the following. These two types of windows are visualized in Fig. 6.5.

$\text{SPECTACLE}_{\text{UW}}$ (*Unweighted SPECTACLE*) $\text{SPECTACLE}_{\text{UW}}$ considers every infer_t in the DNN inference sequence InfSeq as having equal relevance to the violation or satisfaction. So, the value added to a spectrum component is always 1, regardless of the distance of the timestamp t of the inference to the beginning of specification violation τ . In this sense, SPECTACLE is *unweighted*.

$\text{SPECTACLE}_{\text{W}}$ (*Weighted SPECTACLE*) Differently from the unweighted SPECTACLE, $\text{SPECTACLE}_{\text{W}}$ emphasizes the weights that are relevant at the timestamps that are closer to the beginning of the specification violation τ . In this sense, SPECTACLE is *weighted*. Hence, $\text{SPECTACLE}_{\text{W}}$ requires

a time-related window function; to that end, we select the *Hamming window*, which is widely applied in signal processing and spectral analysis to smooth data and reduce noise [5]. The Hamming window Ω_{ham} is defined as:

$$\Omega_{\text{ham}}(t) = 0.54 - 0.46 \cdot \cos\left(\frac{\pi \cdot t}{\tau}\right) \quad \text{with } t \in [0, \tau]$$

When $t = \tau$, the Hamming window has the maximum value 1; instead, when $t = 0$, it has the minimum value 0.08.

6.4.3.2 SCALING FACTOR

Note that different tests $\mathbf{u}$ contribute with different numbers of inferences to the construction of the spectrum. For passing tests, all the inferences till T are considered, while for failing tests, all the inferences till τ ($\tau \leq T$) are considered. To ensure that each test contributes equally to the construction of the spectrum, the window value in failing tests is *scaled* depending on the length of the test. Specifically, when increasing the spectrum of ef and nf at Line 15 of Alg. 4, this is scaled by the *scaling factor* β computed at Line 7. Intuitively, the spectrum additions for inferences of the shorter tests (i.e., the failing ones) are weighted more than those of longer tests (i.e., the passing ones).

The computation of the scaling factor depends on the window function used. Let us consider the constant window used in $\text{SPECTACLE}_{\text{UW}}$. In this case, the scaling factor becomes $\frac{T}{\int_0^\tau 1 dt} = \frac{T}{\tau}$. For example, the scale factor of a test failing at $\tau = T/2$ is $\beta = T/(T/2) = 2$; in this way, a failing test with half of the length of a passing test provides the same contribution to the spectrum. In the case of the Hamming window, the scaling factor can be generalized to $\frac{T}{\int_0^\tau \Omega_{\text{ham}}(t) dt}$.

Table 6.1: Suspiciousness metrics SusMetr used in this paper

Tarantula [76]	Ochiai [2]	D* [154]	Jaccard [3]	Kulczynski2 [117]
$\frac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf} + \frac{ep}{ep+np}}$	$\frac{ef}{\sqrt{(ef+nf)(ef+ep)}}$	$\frac{ef^*}{ep+nf}$	$\frac{ef}{ef+nf+ep}$	$\frac{1}{2} \left(\frac{ef}{ef+nf} + \frac{ef}{ef+ep} \right)$

* > 0. Here we set * as 3, according to previous practices [154, 156].

6.4.4 SUSPICIOUSNESS SCORE CALCULATION

Based on the execution spectra of all weights, SPECTACLE uses *suspiciousness metrics* to calculate the *suspiciousness score* of each weight, to identify the weights that are highly relevant to the violations. Table 6.1 reports five suspiciousness metrics that have been widely used in software fault localisation [155].² The basic premise behind these metrics is that the more frequently a weight is ranked in top k when the violation occurs, and the less frequently it is ranked in top k when the satisfaction happens, the more suspicious the weight is. A higher suspiciousness score means that the weight is more likely to be a faulty one. Lines 16-18 describe the process of *Suspiciousness score calculation*. Based on the execution spectrum ES_w obtained by Alg. 4, SPECTACLE calculates the suspiciousness score for each w based on a given suspiciousness metric SusMetr (Lines 16-17) and selects the top s weights that have the highest suspiciousness scores as the output of fault localisation (Line 18), because these are the weights that are more likely responsible for the violations.

Example 2. For explanation purposes, we show how SPECTACLE_{uw} (that uses the constant window) works on a toy example; the application of SPECTACLE_w would be similar. We consider a DNN controller having only one hidden layer that includes two neurons, each having only one weight. Table 6.2 shows the computation of fault localisation for it. The system executes in time interval $[t_0, t_3]$. The test suite TS consists of two input signals $\mathbf{u}_1$ and $\mathbf{u}_2$. $\mathbf{u}_1$ triggers the inference sequence InfSeq_1 and gives the system output $\mathbf{v}_1$ that satisfies specification φ ; $\mathbf{u}_2$ triggers

²Note that we did not use the metric Kulczynski1 [117], as it has been shown that it is equivalent to Jaccard (see Proposition 7.2 in [117]).

Table 6.2: An illustrative example of SPECTACLE_{UW} (gray cells indicate top- k)

		Forward impact				Exec. spectrum from an exec.	Exec. spectrum $\langle ep, np, ef, nf \rangle$	Suspiciousness score
		t_0	t_1	t_2	t_3			
w_1	u_1	0.7	0.9	0.6	1.1	$\langle 3, 1, 0, 0 \rangle$	$\langle 3, 1, 0, 4 \rangle$	0
	u_2	0	0.1	0.3	$\times$	$\frac{4}{3} \cdot \langle 0, 0, 0, 3 \rangle$		
w_2	u_1	0.3	0.5	0.8	0.2	$\langle 1, 3, 0, 0 \rangle$	$\langle 1, 3, 4, 0 \rangle$	0.8
	u_2	0.4	0.9	2.3	$\times$	$\frac{4}{3} \cdot \langle 0, 0, 3, 0 \rangle$		

the inference sequence InfSeq_2 and gives the system output v_2 that violates φ at t_2 . As shown in Line 6 of Alg. 4, we need to identify the timestamp τ that is either when the system output starts to violate the specification, or the simulation time T ; as a result, we identify τ_1 and τ_2 respectively for u_1 and u_2 , namely, they are $\tau_1 = t_3$ and $\tau_2 = t_2$. Table 6.2 also reports the *forward impact* of weights w_1 and w_2 for each DNN inference at each timestamp.

We set the hyperparameter k in Line 13 as 1, i.e., the weight whose *forward impact* is ranked in top-1 at the hidden layer will be regarded as *executed* (in gray in the table). For instance, for InfSeq_1 , since the *forward impact* of w_1 at t_1 is 0.9, higher than 0.5 of w_2 , w_1 is in the top-1 at t_1 ; hence, ep of w_1 and np of w_2 are incremented in the spectrum. Considering all the four timestamps, the execution spectra of w_1 and w_2 for InfSeq_1 are $\langle 3, 1, 0, 0 \rangle$ and $\langle 1, 3, 0, 0 \rangle$.

In this example, SPECTACLE_{UW} uses a constant window (see §6.4.3.1) to assign equivalent significance to DNN inferences at different timestamps. For the sequence InfSeq_2 (that leads to the violation of φ at $\tau_2 = t_2$), the constant window gives the same contribution of 1 at t_0 , t_1 , and t_2 . Since the *forward impact* of w_2 is consistently greater than that of w_1 , we consider w_2 as always executed; so, the execution spectrum of w_2 for InfSeq_2 is $\langle 0, 0, 3, 0 \rangle$. The scaling factor β is used to keep the same contribution to the spectrum between passing and failing sequences. For InfSeq_2 that leads to specification violation at $\tau_2 = t_2$, β is $\frac{|\{t_0, \dots, t_3\}|}{|\{t_0, \dots, t_2\}|} = \frac{4}{3}$.

In this example, we use Jaccard as suspiciousness metric and we set $s = 1$. The score of w_1 is 0 and the score of w_2 is 0.8, so SPECTACLE_{UW} returns w_2 as the suspicious weight. $\triangleleft$

6.5 EXPERIMENT SETUP

In this section, we elaborate on the detailed experimental setup to investigate the effectiveness of SPECTACLE. All AI-enabled CPSs used as benchmarks and the source code of SPECTACLE for reproducing our experiments are publicly available on the GitHub repository [100]. Additional plots not reported in the paper are available at [103].

6.5.1 RESEARCH QUESTIONS

We identified six research questions to assess the effectiveness of SPECTACLE.

RQ1: *Does SPECTACLE effectively localise the faulty DNN weights in AI-enabled CPSs?*

In this RQ, we evaluate whether SPECTACLE can actually identify faulty weights that exist in a DNN controller C . To do this, we compare it with two baseline approaches.

RQ2: *How does the selection of hyperparameter k influence the effectiveness of SPECTACLE?*

In this RQ, we investigate which value of k (used to consider a weight “executed” or not in spectrum construction. See §6.4.3 and Line 13 in Alg. 4) allows to obtain the best fault localisation results.

RQ3: *How do $SPECTACLE_W$ and $SPECTACLE_{UW}$ compare with each other?*

In this RQ, we investigate the influence of the window function (see §6.4.3.1) to the effectiveness of SPECTACLE. Namely, we compare the performances of $SPECTACLE_W$ with that of $SPECTACLE_{UW}$.

RQ4: *How does the selection of the suspiciousness metric $SusMetr$ influence the effectiveness of SPECTACLE?*

In this RQ, we investigate which suspiciousness metric, among the five reported in Table 6.1, allows to obtain the best fault localisation results.

Table 6.3: Benchmark AI-enabled CPSs – CPSs and their DNN controllers ($|\mathcal{W}|$ is the number of weights of C ; $|\mathcal{M}|$ is the number of blocks of $\mathcal{M}$)

CPS	Description	C's inputs		C's output	$\mathcal{M}^C$	C's structure	$ \mathcal{W} $	$ \mathcal{M} $
		$\mathbf{u}$ (range [lower, upper])	$\mathbf{x}$					
ACC	Keep relative distance and ego car speed	lead car's acceleration [-1, 1]	ego car's speed	ego car's acceleration	ACC#1	[10, 10, 10]	200	49
					ACC#2	[15, 15, 15]	450	49
AFC	Maintain an optimal	throttle angle [8.8, 61.1]; engine speed	inlet air mass flow rate;	commanded fuel	AFC#1	[15, 15, 15]	450	153
					AFC#2	[15, 15, 15, 15]	675	153
SC	air-to-fuel ratio	[900, 1100]	air-to-fuel ratio	cooling	SC#1	[10, 10, 10, 10]	300	55
	Stabilize the pressure	steam flowrate [3.99, 4.01]	actual pressure	water flow	SC#2	[15, 15, 15, 15]	675	55
				rate	SC_R	[3]	18	55
WT	Monitor the water level	water level reference [10, 20]	actual water level	waterflow	WT#1	[5, 5, 5]	50	11
					WT#2	[15, 15, 15]	450	11

RQ5: *Are the weights identified by SPECTACLE useful for improving the DNN controller performance?*

In this RQ, we investigate whether the performance of the DNN controller can be improved by repairing the weights identified by SPECTACLE.

RQ6: *What is the computational cost of SPECTACLE?*

In this RQ, we investigate if the computational cost of executing SPECTACLE is acceptable and if it differs across different AI-enabled CPSs.

6.5.2 BENCHMARKS

To evaluate the effectiveness of SPECTACLE, we selected nine AI-enabled CPSs. Table 6.3 reports their functionalities, controller details, and plant details.

The AI-enabled CPSs have been built as follows. We first selected, as plants $\mathcal{M}$, four CPSs largely used in the literature [73, 109, 114, 134, 143, 159, 176] that span over different domains, such as automotive and chemical industry: Adaptive Cruise Control (ACC), Abstract Fuel Control (AFC), Steam Condenser (SC), and Water Tank (WT). All the subject CPSs are implemented in Simulink [112], the de facto standard to design and model CPSs. All these CPSs are embedded

with a classic controller, that we want to replace with a DNN controller. In the literature, for each of these CPSs, some STL specifications describing their expected behaviour have been reported. We selected some specifications for each CPS and tuned their parameters (e.g., constants used as thresholds), so that the classic controller always behaves correctly over the input signals $\mathbf{u}$ that we use to train the DNN controller (as the classic controller must act as a supervisor during training. See §6.5.2).

Adaptive Cruise Control (ACC). ACC [109] aims to maintain a safe distance between the ego car and the lead car by adjusting the acceleration of the ego car. The ACC system takes the lead car’s acceleration as input and provides outputs in terms of the velocities and distances covered by both the lead and ego cars. The specification φ_{ACC}^1 requires that the relative distance d_{rel} between two cars must always be higher than the sum of the safe distance d_{safe} and the *reaction distance* reactDist (i.e., $1.4 \times v_{\text{ego}}$, where 1.4 is the *time-gap* [93]) of the ego car during the time interval $[0, 50]$, and that the speed of the ego car v_{ego} should always be lower than a given speed v_{set} . Here, d_{safe} and v_{set} are set to 10 and 30, respectively.

$$\varphi_{\text{ACC}}^1 \equiv \Box_{[0,50]}((d_{\text{rel}} \geq d_{\text{safe}} + \text{reactDist}) \wedge v_{\text{ego}} \leq v_{\text{set}}) \quad \text{with } \text{reactDist} = 1.4 \times v_{\text{ego}}$$

The second specification φ_{ACC}^2 states that during the time interval $[0, 45]$, if the relative distance d_{rel} is lower than the sum of the reaction distance reactDist of the ego car and a distance slightly above the safe distance d_{safe} (we use 12 in the specification), the system is on the edge of a hazardous state. The system is required to recover from this state and return to a safe state within 5 seconds.

$$\varphi_{\text{ACC}}^2 \equiv \Box_{[0,45]}(d_{\text{rel}} < 12 + \text{reactDist} \rightarrow \Diamond_{[0,5]}(d_{\text{rel}} \geq 12 + \text{reactDist})) \quad \text{with } \text{reactDist} = 1.4 \times v_{\text{ego}}$$

Abstract Fuel Control (AFC). AFC, developed by Toyota [73], aims to regulate the mixture ratio

of air and fuel in the engine intake for optimal combustion efficiency. The system takes the pedal angle and engine speed as exogenous inputs, and minimizes the deviation $\mu = |AF - AF_{ref}| / AF_{ref}$, which is expressed as the absolute difference between the actual air fuel ratio AF and its reference value AF_{ref} divided by AF_{ref} . With the exogenous signals, the controller dynamically adjusts intake of air and fuel, to maintain the optimal air-fuel ratio. The specification φ_{AFC}^1 requires that the deviation μ should never exceed a predefined value μ_{set} during the time interval $[0, 30]$. Here, μ_{set} is set to 0.2.

$$\varphi_{AFC}^1 \equiv \Box_{[0,30]} (\mu \leq \mu_{set})$$

The second specification φ_{AFC}^2 requires that, during time interval $[10, 28.5]$, when μ exceeds 0.1, the controller should bring μ back to 0.1 within 1.5 seconds.

$$\varphi_{AFC}^2 \equiv \Box_{[10,28.5]} (\mu > 0.1 \rightarrow \Diamond_{[0,1.5]} (\mu \leq 0.1))$$

Steam Condenser (SC). SC [159] is a component of an electric power system. During the condensation process of steam, it aims to keep the pressure in the condenser around a reference pressure. The external input of the system is the steam mass flow rate, while the system's output is the internal pressure of the condenser. The specification φ_{SC} of SC requires that the pressure should be maintained between 87 and 87.5 in the time interval $[30, 35]$.

$$\varphi_{SC} \equiv \Box_{[30,35]} (pressure \geq 87 \wedge pressure \leq 87.5)$$

Water Tank (WT). WT [134] is used to control the inflow and outflow of water. In this system, water can enter or flow out the tank until the water level reaches a predefined reference value. Over time, the water level should stabilise and match the reference level. The system takes the reference water level h_{ref} as input and continuously outputs the actual water level h_{out} . The specification φ_{WT} requires that the absolute deviation error between h_{out} and h_{ref} remains below

a predefined threshold value err_{set} during specific time intervals, namely $[4, 5]$, $[9, 10]$, and $[14, 15]$. In our experiments, we set err_{set} to 0.86.

$$\varphi_{WT} \equiv \square_{[4,5] \cup [9,10] \cup [14,15]} (|h_{out} - h_{ref}| \leq err_{set})$$

CONSTRUCTION OF AN AI-ENABLED CPS

To build AI-enabled CPSs to use as a target of SPECTACLE, we proceeded as follows. In order to obtain AI-enabled CPSs $\mathcal{M}^C$, for each plant $\mathcal{M}$ described above, we considered two feed-forward DNNs as controller C ; moreover, for SC, we also considered an RNN as controller C .

Some of these controllers are taken from existing repositories. Specifically, for AFC, we took the feed-forward DNNs trained in FalsifAI [176]. For SC, instead, we took the RNN controller proposed in [159]. The other controllers, instead, have been trained for this work. Specifically, for ACC, SC and WT, we followed the approach explained in [176] and trained two feed-forward neural networks for each of them. The specific training steps are detailed as follows:

- (i) we randomly generate a set of 10,000 input signals $\mathbf{u}$; these input signals $\mathbf{u}$ are constrained within specific input ranges (reported in Table 6.3) obtained by following official documentations and technical manuals [109, 134, 159];
- (ii) we execute the model embedded with the classic controller over the generated input signals $\mathbf{u}$, and collect the controller output $\mathbf{c}$ and system state $\mathbf{x}$; note that, as explained before, the classic controllers never violate the system specification;
- (iii) the inputs $\mathbf{u}$, and the observed outputs $\mathbf{c}$ and $\mathbf{x}$ constitute the training data for the DNN controller C ;
- (iv) we employ the *Deep Learning Toolbox* of MATLAB to train a neural network C with the aim to minimise the prediction error of $\mathbf{c}$ given input $\langle \mathbf{x}, \mathbf{u} \rangle$ for C .³ After training and validating

³We decided to train the neural networks in MATLAB to ease the integration with the CPS plants that are modeled

Table 6.4: Training details of the DNN controllers C

DNN controller C	Training algorithm	Max epoch	Min gradient	Final training loss (MSE)
ACC#1	LMBP [*]	1000	1×10^{-10}	3.18×10^{-3}
ACC#2	SCG ^{**}	1000	1×10^{-10}	7.49×10^{-3}
AFC#1	LMBP	1000	1×10^{-10}	1.27×10^{-5}
AFC#2	LMBP	1000	1×10^{-10}	1.79×10^{-1}
SC#1	LMBP	1000	1×10^{-10}	3.62×10^{-3}
SC#2	LMBP	1000	1×10^{-10}	3.78×10^{-3}
SC_R	–	–	–	–
WT#1	BFG ^{***}	1000	1×10^{-10}	1.84×10^0
WT#2	BFG	1000	1×10^{-10}	1.83×10^0

^{*} LMBP: *Levenberg–Marquardt Backpropagation (LMBP) Algorithm* [99]

^{**} SCG: *Scaled Conjugate Gradient backpropagation (SCG) Algorithm* [115]

^{***} BFG: *BFGS quasi-Newton (BFG) Algorithm* [58]

the neural network, we replace the classic controller in the Simulink model with the newly trained neural network controller C , so obtaining an AI-enabled CPS $\mathcal{M}^C$.

Table 6.4 reports details about the training of the DNN controllers (either performed by us or in the original papers) in terms of used training algorithm, maximum number of training epochs, the threshold of gradient descent for which the training stops, and the final loss in terms of Mean Squared Error (MSE).⁴ Table 6.3 reports the complexities of the nine AI-enabled CPSs in terms of number of blocks of each $\mathcal{M}$ and number of weights of the controller C . From Table 6.4, we observe that, for most of the benchmarks, the final training loss is similar between the two DNN controllers; the only difference is AFC, for which the most complex controller (i.e., AFC#2) obtains a larger loss; this is probably due to the fact that AFC#2 is a much more complex network than AFC#1.

In the following, a *correct benchmark* is given by an AI-enabled CPS $\mathcal{M}^C$ and a specification φ , identified as $\mathcal{M}^C_\varphi$. Therefore, in total, there are 13 correct benchmarks (there are four correct benchmarks for both ACC and AFC, three correct benchmarks for SC, and two correct benchmarks for WT).

in Simulink.

⁴Note that, for SC_R, we only have access to the RNN controller reported in [109, 159], but we do not have information on how it was trained and what was the final training loss.

FAULTY BENCHMARKS

To evaluate the effectiveness of SPECTACLE, we introduce artificial faults in C and we check the ability of SPECTACLE and of the compared baseline approaches in identifying them. To do this, we mutate neuron weights to produce faulty DNN controllers, similarly to what was done by Sohn et al. [133] to mimic improper training of the weights. In order to measure “how much” a faulty controller affects the correctness of the system, we define the *safety rate* that counts how many tests of the test suite satisfy the specification. Formally, given a test suite TS , the safety rate SR of an AI-enabled CPS $\mathcal{M}^C$ w.r.t. a specification φ is defined as:

$$SR(\mathcal{M}^C, \varphi, TS) = \frac{|\{\mathbf{u} \in TS \mid \mathcal{M}^C(\mathbf{u}) \models \varphi\}|}{|TS|} \quad (6.2)$$

i.e., the ratio of tests satisfying the specification.

Given a correct benchmark $\mathcal{M}^C_{-\varphi}$ (see §6.5.2), the generation of *faulty benchmarks* containing different number of faults is as follows.

First, we generate a test suite TS of 100 tests (i.e., input signals $\mathbf{u}$) for the original AI-enabled CPS. Since the time domain is continuous, it is not possible to synthesise an input signal by providing its value at each moment; therefore, we adopt a commonly used parametrised representation, namely *piecewise constant*, in which the time domain is split in H intervals, and a signal is given by H constants, each identifying the value of the signal in a time interval. We generate the tests by uniformly sampling the values for the H intervals. We set H by following related literature [134, 176]: following [176], H has been set to 10 for ACC, and to 5 for AFC; following [134], H has been set to 7 for SC, and to 3 for WT. $\mathcal{M}^C$ passes all the tests, i.e., the safety rate $SR(\mathcal{M}^C, \varphi, TS)$ is 100%.

Then, we produce a set of *faulty benchmarks* containing single faults as follows:

- we identify the range of weights values in the original controller C , i.e., $[W_{min}, W_{max}]$;

- for each weight w of C :
 - we sample a value v' in the original range of weights enlarged of 50%, i.e., $v' \in [W_{min} - \frac{\text{sgn}(W_{min}) \times W_{min}}{2}, W_{max} + \frac{\text{sgn}(W_{max}) \times W_{max}}{2}]$, where $\text{sgn}(\cdot) \in \{-1, 1\}$ identifies the sign of a value. We set w to v' , so obtaining a mutated model $\mathcal{M}_f^C$. The intuition of sampling around the weights values of the original controller is that we want to produce realistic DNNs that are obtainable by a good training;
 - we run $\mathcal{M}_f^C$ over each test in TS and assess φ 's satisfaction; if the safety rate $SR(\mathcal{M}_f^C, \varphi, TS)$ is between 10% and 90%, we keep $\mathcal{M}_f^C$ as a faulty model, otherwise we discard it and sample another value; the intuition behind using this range of SR is to guarantee that all faulty benchmarks have both passing and failing tests (which is required for SBFL to be effective), and to have different types of faults. At most, we perform 20 attempts of modification of each weight w ;
- we denote by $\text{SetMuts}_{\mathcal{M}^C-\varphi}^1$ the set of all the faulty benchmarks $\mathcal{M}_f^C$ (obtained from the correct benchmark $\mathcal{M}^C-\varphi$) containing a single fault. At most, $\text{SetMuts}_{\mathcal{M}^C-\varphi}^1$ contains as many mutants as the number of weights of $\mathcal{M}^C$.⁵

Then, we build faulty benchmarks containing higher number of faults as follows. We build faulty benchmarks $\text{SetMuts}_{\mathcal{M}^C-\varphi}^2$ having two faults by combining two single-fault benchmarks from $\text{SetMuts}_{\mathcal{M}^C-\varphi}^1$. We only keep the faulty benchmarks $\mathcal{M}_f^C$ whose safety rate $SR(\mathcal{M}_f^C, \varphi, TS)$ is between 10% and 90%; moreover, to guarantee that the two weight mutations are both contributing to the decrease in safety (and not cancelling each other), we require that the SR of the combined benchmark is lower than that of the constituent single-fault faulty benchmarks. At most, we generate 200 benchmarks in $\text{SetMuts}_{\mathcal{M}^C-\varphi}^2$; however, we usually generate less than 200 benchmarks, as we do not find enough suitable combinations of single-fault benchmarks.

⁵Note that, for some weights, in 20 attempts we were not able to change the weight in a way that ensures that the safety rate $SR(\mathcal{M}_f^C, \varphi, TS)$ falls between 10% and 90%.

Table 6.5: Number of faulty benchmarks (for each $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^i$ ($i \in \{1, \dots, 5\}$) and for $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$) generated from the 13 correct benchmarks

	$\text{ACC}\#1_{-\varphi_{\text{ACC}}}^1$	$\text{ACC}\#1_{-\varphi_{\text{ACC}}}^2$	$\text{ACC}\#2_{-\varphi_{\text{ACC}}}^1$	$\text{ACC}\#2_{-\varphi_{\text{ACC}}}^2$	$\text{AFC}\#1_{-\varphi_{\text{AFC}}}^1$	$\text{AFC}\#1_{-\varphi_{\text{AFC}}}^2$	$\text{AFC}\#2_{-\varphi_{\text{AFC}}}^1$
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$	36	39	39	59	66	49	79
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^2$	179	200	195	190	164	178	147
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^3$	122	186	143	182	110	147	99
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^4$	77	164	93	170	68	123	65
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^5$	53	123	64	151	39	87	37
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$	467	712	534	752	447	584	427
	$\text{AFC}\#2_{-\varphi_{\text{AFC}}}^2$	$\text{SC}\#1_{-\varphi_{\text{SC}}}$	$\text{SC}\#2_{-\varphi_{\text{SC}}}$	$\text{SC_R}_{-\varphi_{\text{SC}}}$	$\text{WT}\#1_{-\varphi_{\text{WT}}}$	$\text{WT}\#2_{-\varphi_{\text{WT}}}$	Total
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$	70	12	11	20	23	71	574
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^2$	169	37	44	7	177	200	1887
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^3$	124	10	13	0	136	199	1471
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^4$	91	65	3	0	85	191	1195
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^5$	61	35	57	0	56	177	940
$\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$	515	159	128	27	477	838	6067

In a similar way, we generate benchmark set $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^3$ (by merging benchmarks from $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$ and $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^2$), benchmark set $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^4$ (by merging benchmarks from $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$ and $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^3$), and benchmark set $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^5$ (by merging benchmarks from $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$ and $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^4$).⁶

So, for each of the 13 correct benchmarks $\mathcal{M}^C_{-\varphi}$, we produce the set $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}} = \cup_{i \in \{1, \dots, 5\}} \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^i$ of its faulty versions. Table 6.5 reports the number of generated faulty benchmarks. Moreover, Table 6.6 shows, for different intervals of safety rate (first row), the percentage of faulty models having safety rate in that interval (second row). We observe that, overall (i.e., for $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$), we obtain a quite uniform distribution that covers different situations. Regarding the different subsets $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^i$ (with $i \in \{1, \dots, 5\}$), we observe that, as expected, those containing more faulty weights tend to have more benchmarks with lower SR.

⁶Note that, by following this protocol, we were not able to build faulty benchmarks with more than one faulty weight for $\text{SC}\#1_{-\varphi_{\text{SC}}}$ and $\text{SC}\#2_{-\varphi_{\text{SC}}}$, as we do not have enough benchmarks in $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$ to combine. Therefore, only for these two correct benchmarks, we produce faulty benchmarks by directly changing weights values. Moreover, for $\text{SC_R}_{-\varphi_{\text{SC}}}$, we were not able to produce faulty benchmarks with more than two modifications for which SR is greater than 10%; so, we only have faulty benchmarks in $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^1$ and $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}^2$ for $\text{SC_R}_{-\varphi_{\text{SC}}}$.

Table 6.6: Distribution of safety rate SR across the faulty benchmarks

	$SR(\mathcal{M}_f^C, \varphi, TS)$							
	[10%, 20%)	[20%, 30%)	[30%, 40%)	[40%, 50%)	[50%, 60%)	[60%, 70%)	[70%, 80%)	[80%, 90%]
SetMuts ¹ _{$\mathcal{M}_f^C - \varphi$}	12.3%	8.5%	11.0%	9.9%	10.1%	9.0%	14.1%	25.1%
SetMuts ² _{$\mathcal{M}_f^C - \varphi$}	4.4%	6.8%	8.5%	7.1%	11.5%	18.7%	24.8%	18.2%
SetMuts ³ _{$\mathcal{M}_f^C - \varphi$}	14.8%	13.5%	13.3%	10.7%	16.6%	13.3%	9.9%	8.0%
SetMuts ⁴ _{$\mathcal{M}_f^C - \varphi$}	20.7%	18.5%	14.1%	12.1%	14.5%	8.0%	6.2%	5.9%
SetMuts ⁵ _{$\mathcal{M}_f^C - \varphi$}	23.0%	16.1%	14.5%	15.3%	13.6%	4.9%	6.1%	6.6%
SetMuts _{$\mathcal{M}_f^C - \varphi$}	13.7%	12.3%	11.9%	10.5%	13.6%	12.2%	13.7%	12.2%

6.5.3 COMPARED APPROACHES

In the experiments, we assess both versions of SPECTACLE, i.e., SPECTACLE_{UW} and SPECTACLE_W. Moreover, we experiment with five different values of k (see Line 13 in Alg. 4), that we identify as *Low*, *Mid-low*, *Mid*, *Mid-high*, and *High*. The values of k depend on the number of the weights whose forward impact is greater than 0, during the execution of the entire test suite by the system. Specifically, given a faulty AI-enabled CPS $\mathcal{M}_f^C$, a specification φ , and a test suite TS , we execute $\mathcal{M}_f^C$ over TS and record the number of the weights with a positive forward impact before τ , denoted as N_p . Then, for $\mathcal{M}_f^C$, we set k as $i \times N_p$, with $i \in \{10\%, 20\%, 30\%, 40\%, 50\%\}$, to instantiate *Low*, *Mid-low*, *Mid*, *Mid-high*, and *High*. The rationale for using N_p is to filter out the weights that have never contributed to the system execution; the justification for selecting at most 50% of these weights, is that higher values may introduce excessive noise into fault localisation (i.e., select too many unnecessary weights).

We compare SPECTACLE with two baseline approaches. First, we compare SPECTACLE with RANDOM selection which randomly selects suspicious weights in Θ_s . In order to account for the non-determinism of RANDOM, by following the guideline [12] of running experiments with randomized algorithms, we executed the RANDOM approach 100 times for each faulty model, and we reported average results.

Then, to assess the importance of considering only the top- k weights, we compared SPECTACLE

with a simpler version of SPECTACLE (called ACT-BASED) that considers all the weights that are *activated*, similarly to what is done in approaches like DeepFault [44] that, as SPECTACLE, is based on SBFL; we consider a weight activated in a given inference if its forward impact is greater than 0. We created two versions of ACT-BASED: ACT-BASED_{uw} and ACT-BASED_w that use the two types of windows.

6.5.4 EVALUATION METRICS

To assess the effectiveness of SPECTACLE and of the baseline approaches (see §6.5.3), for each set of faulty benchmarks $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$ (see §6.5.2), we run each compared approach APP over each $\mathcal{M}_f^C \in \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$, and assess its *detection rate* defined as follows. First, we define the *recall* of an approach APP over a faulty benchmark $\mathcal{M}_f^C$ as the percentage of modified weights that have been correctly identified by APP in its final results Θ_s (see Line 18 in Alg. 4), i.e.,

$$\text{recall}(\text{APP}, \mathcal{M}_f^C) = \frac{\# \text{ of modified weights of } \mathcal{M}_f^C \text{ correctly identified in } \Theta_s \text{ by APP}}{\# \text{ of modified weights in } \mathcal{M}_f^C}$$

The *detection rate* (DR) is then defined as

$$\text{DR}(\text{APP}, \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}) = \frac{\sum_{\mathcal{M}_f^C \in \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}} \text{recall}(\text{APP}, \mathcal{M}_f^C)}{|\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}|} \quad (6.3)$$

We calculate the DR for each possible value of s suspicious weights in the final results Θ_s . Subsequently, we compute the *area under the curve* (AUC) described by DR at the increase of s (we call it AUC_{DR}). The performance of the different approaches will be evaluated based on their respective AUC_{DR} values. Specifically, in order to compare the performance of the different approaches, we perform statistical analysis of their AUC_{DR} results. Given two approaches APP1 and APP2, we do pairwise comparison between the values of AUC_{DR} across all the faulty benchmarks and suspiciousness metrics, using the non-parametric test Wilcoxon signed-rank test. We use

$\alpha = 0.05$ as significance level. If the p-value is less than α , we reject the null hypothesis that there is no significant difference. If there is a significant difference, we use Cohen's d effect size [29] to assess the strength of the significance. If $d > 0$, then APP1 is better; otherwise APP2 is better. We interpret the effect size using the following categories [83]: *small* if $d \in (0, 0.2)$, *medium* if $d \in [0.2, 0.8)$, and *large* if $d \geq 0.8$.

SPECTACLE, as any other spectrum-based fault localisation approach, returns false positive results that depend on the value of the hyperparameter s (i.e., number of weights returned in Θ_s): the higher the s , the higher the number of false positive results. In order to assess this, given an approach APP and a faulty benchmark $\mathcal{M}_f^C \in \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$, we first compute the *false detection* as the percentage of wrongly identified weights, i.e.,

$$\text{falseDetection}(\text{APP}, \mathcal{M}_f^C) = \frac{\# \text{ of modified weights of } \mathcal{M}_f^C \text{ wrongly identified in } \Theta_s \text{ by APP}}{|\Theta_s|}$$

Then, we compute *false discovery rate* (FDR) as

$$FDR(\text{APP}, \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}) = \frac{\sum_{\mathcal{M}_f^C \in \text{SetMuts}_{\mathcal{M}^C_{-\varphi}}} \text{falseDetection}(\text{APP}, \mathcal{M}_f^C)}{|\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}|} \quad (6.4)$$

Note that a given level of FDR is unavoidable, and primarily depends on the value of s . A higher value of FDR will usually correspond to a higher value of DR ; so, a user must decide s depending on the desired trade-off between accuracy and precision of the results. Similarly to what done for DR , we calculate the FDR for each possible value of s , and compute the *area under the curve* (AUC) described by FDR at the increase of s (we call it AUC_{FDR}). Then, we do statistical analysis among approaches as described before. In this case, the interpretation of Cohen's d effect size is different, as a lower value of AUC_{FDR} is better. So, given two approaches APP1 and APP2, if $d < 0$, then APP1 is better; otherwise APP2 is better. We interpret the effect size using the following categories [83]: *small* if $d \in (-0.2, 0)$, *medium* if $d \in (-0.8, -0.2]$, and *large* if $d \leq -0.8$.

ASSESSMENT OF RQs

In order to answer RQ1, we check the statistical comparison in terms of AUC_{DR} and AUC_{FDR} , between the different versions of SPECTACLE and, respectively, RANDOM and ACT-BASED.

In order to answer RQ2, we check the statistical comparison between the different versions of SPECTACLE using the different values of k , i.e., *Low*, *Mid-low*, *Mid*, *Mid-high*, and *High* (see §6.5.3).

In order to answer RQ3, we check the statistical comparison between $SPECTACLE_{UW}$ and $SPECTACLE_W$. Moreover, we also report, for each benchmark $\mathcal{M}^C_{-\varphi}$, the percentage of times (across different faulty benchmarks $\mathcal{M}^C_f$, values of k , suspiciousness metrics $SusMetr$, and values of s) that $SPECTACLE_{UW}$ outperforms $SPECTACLE_W$, that they obtain the same results, and that $SPECTACLE_W$ outperforms $SPECTACLE_{UW}$.

In order to answer RQ4, we perform statistical analysis using Wilcoxon signed-rank test and Cohen’s d effect size as described before. However, in this case, given two suspiciousness metrics $SusMetr_1$ and $SusMetr_2$, we compare the values of AUC_{DR} and AUC_{FDR} across different faulty benchmarks and settings of SPECTACLE.

In order to answer RQ5, we need to select some representative faulty benchmarks to repair, as repairing all of them would take too much time. We proceed as follows. We first select four correct benchmarks: for each of the four CPSs, we select a specification and its most complex DNN controller, namely $ACC\#2_ \varphi^1_{ACC}$, $AFC\#2_ \varphi^1_{AFC}$, $SC\#2_ \varphi_{SC}$, and $WT\#2_ \varphi_{WT}$. Then, for each correct benchmark $\mathcal{M}^C_{-\varphi}$, we select one faulty benchmark in $SetMuts^1_{\mathcal{M}^C_{-\varphi}}$ (called $\mathcal{M}^C_{f1}$), one in $SetMuts^3_{\mathcal{M}^C_{-\varphi}}$ (called $\mathcal{M}^C_{f3}$), and one in $SetMuts^5_{\mathcal{M}^C_{-\varphi}}$ (called $\mathcal{M}^C_{f5}$). Therefore, we have selected 12 faulty benchmarks to repair. Then, for each $\mathcal{M}^C_f$, among all fault localisation results obtained with the different versions of SPECTACLE, we select those that allow identifying all the modified weights with the minimum value of top s . Then, we run a search-based DNN repair approach that has been recently proposed for AI-enabled CPSs [101]; the approach searches for alternative

values of some weights of the DNN controller, with the goal of maximising the number of passing tests in a test suite TS . As search variables, we use the weights Θ_s that have been identified by SPECTACLE; as test suite TS , we use the same used by SPECTACLE (see §6.5.2). We use a population size of 50 individuals, and run the approach for 50 generations. Finally, we check the safety rate SR obtained by the final repaired model.

In order to answer RQ6, we select, for each of the 13 correct benchmarks $\mathcal{M}^C_{-\varphi}$, five faulty benchmarks, one for each set of mutants $\text{SetMuts}^i_{\mathcal{M}^C_{-\varphi}}$ ($i \in \{1, \dots, 5\}$). Then, we record the time for executing the test suite (i.e., the simulation of the tests), and the time spent for doing fault localisation. We report the time for doing fault localisation in terms of time spent for calculating the forward impact and for calculating the execution spectrum and the suspiciousness score.

6.5.5 SOFTWARE AND HARDWARE SPECIFICATIONS

All subject CPSs are implemented in Simulink with dependencies on MATLAB *Deep Learning Toolbox*. The experiments were conducted on a server with a 3.3GHz Intel i9-10940X CPU, 64G RAM, and two NVIDIA RTX A6000 GPUs, 10 threads.

6.6 EXPERIMENTAL RESULTS

In this section, we present the experimental results. Before answering the RQs, we first provide a visual representation of the results. Fig. 6.6 illustrates the overall effectiveness of SPECTACLE and of the compared approaches (see §6.5.3) on the 13 groups of faulty benchmarks (see §6.5.2), using Kulczynski2 as the suspiciousness metric.⁷

Each figure represents, for a given benchmark $\mathcal{M}^C_{-\varphi}$, the *detection rate* DR (see Eq. 6.3) of the approaches, i.e., their ability in identifying the faulty weights of the faulty versions $\text{SetMuts}_{\mathcal{M}^C_{-\varphi}}$. The x -axis represents the percentage of weights examined in the final ranking of suspicious

⁷We selected the Kulczynski2 metric, as it is the best metric according to the results of RQ4 in §6.6.4.

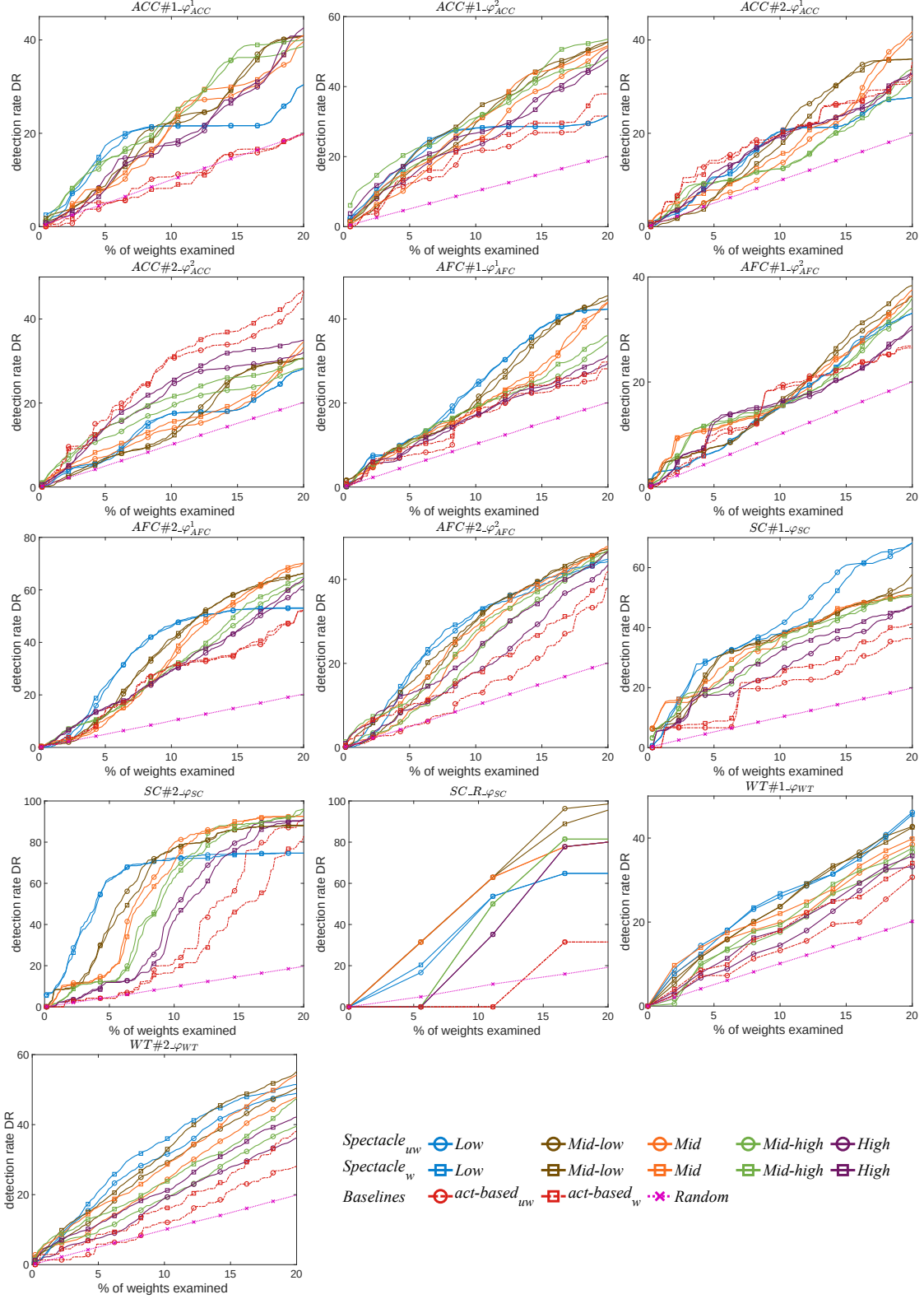


Figure 6.6: Effectiveness of the compared approaches (in terms of detection rate DR) on the 13 groups of faulty benchmarks (using Kulczynski2 as suspiciousness metric $SusMet_r$)

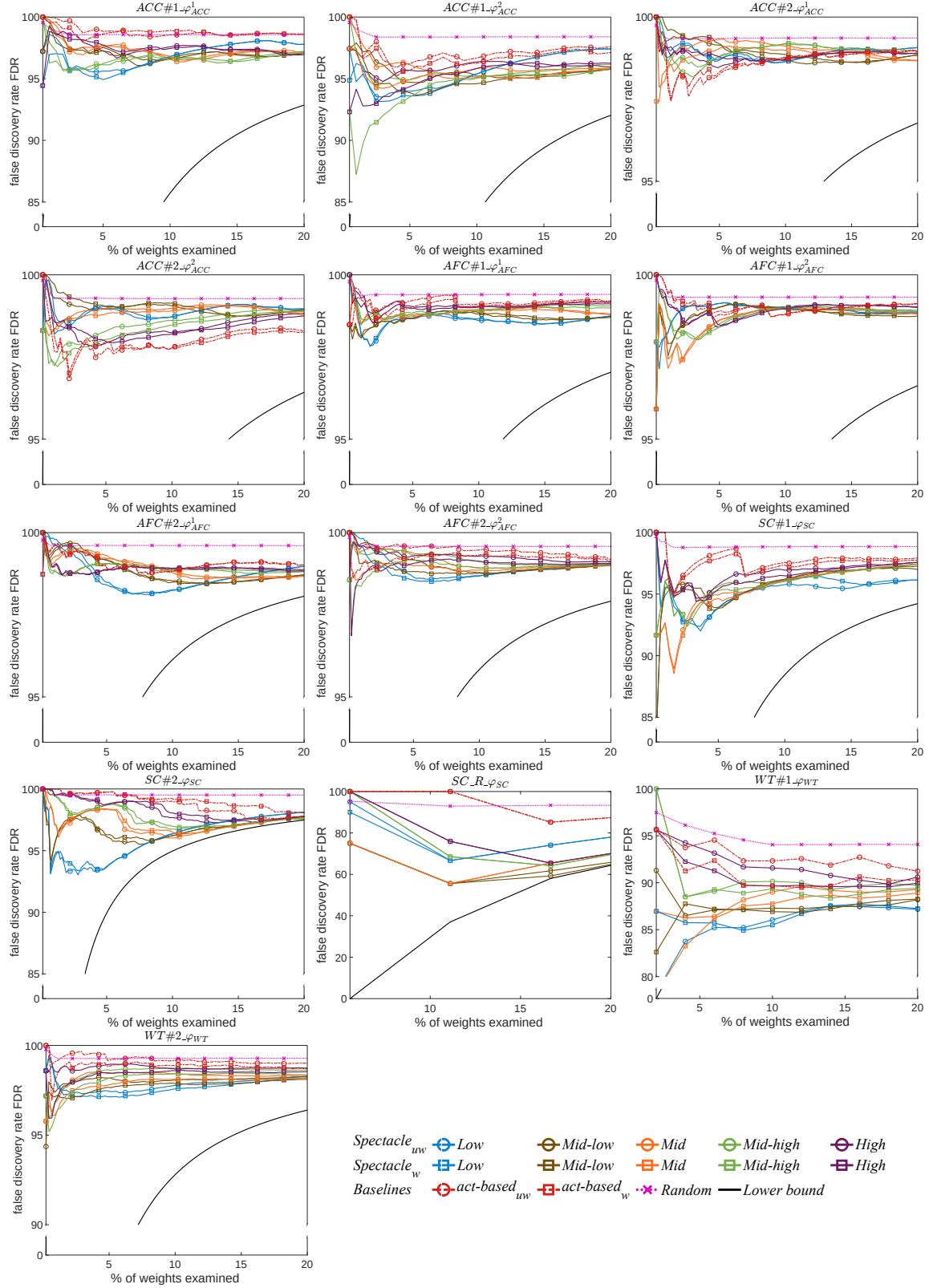


Figure 6.7: Effectiveness of the compared approaches (in terms of false discovery rate *FDR*) on the 13 groups of faulty benchmarks (using Kulczynski2 as suspiciousness metric *SusMetr*)

Table 6.7: Effectiveness of the compared approaches in terms of AUC_{DR} with $s \leq 20\%$, using Kulczynski2 as suspiciousness metric SusMetr. The higher the value, the better (the best approach and second best approach are highlighted in gray and light gray, respectively)

	SPECTACLE _{uw}					ACT-BASED _{uw}	SPECTACLE _w					ACT-BASED _w	RANDOM
	Low	Mid-low	Mid	Mid-high	High		Low	Mid-low	Mid	Mid-high	High		
ACC#1_ φ^1_{ACC}	0.1806	0.2105	0.1930	0.2297	0.1882	0.0994	0.1847	0.2090	0.2028	0.2377	0.1855	0.0966	0.0998
ACC#1_ φ^2_{ACC}	0.2301	0.2843	0.2568	0.2813	0.2463	0.1849	0.2335	0.3074	0.2924	0.3179	0.2662	0.2102	0.1008
ACC#2_ φ^1_{ACC}	0.1660	0.1939	0.1650	0.1528	0.1752	0.1937	0.1648	0.1837	0.1736	0.1467	0.1752	0.1891	0.0993
ACC#2_ φ^2_{ACC}	0.1448	0.1536	0.1424	0.1744	0.2052	0.2575	0.1429	0.1494	0.1523	0.1947	0.2226	0.2667	0.1008
AFC#1_ φ^1_{AFC}	0.2400	0.2274	0.1972	0.1800	0.1598	0.1504	0.2383	0.2237	0.1996	0.1874	0.1674	0.1660	0.1017
AFC#1_ φ^2_{AFC}	0.1612	0.1736	0.1786	0.1690	0.1567	0.1578	0.1620	0.1793	0.1819	0.1730	0.1577	0.1547	0.1004
AFC#2_ φ^1_{AFC}	0.3653	0.3619	0.3326	0.3056	0.2933	0.2573	0.3696	0.3665	0.3433	0.3175	0.3022	0.2612	0.1010
AFC#2_ φ^2_{AFC}	0.2692	0.2596	0.2408	0.2199	0.1953	0.1433	0.2800	0.2751	0.2658	0.2532	0.2372	0.1885	0.1005
SC#1_ φ_{SC}	0.4117	0.3451	0.3372	0.3133	0.2620	0.1901	0.3985	0.3464	0.3454	0.3300	0.2915	0.2211	0.1008
SC#2_ φ_{SC}	0.6210	0.6201	0.5957	0.5415	0.4571	0.3436	0.6153	0.6089	0.5785	0.5230	0.4273	0.2702	0.0973
SC_R_ φ_{SC}	0.4424	0.6250	0.5478	0.4784	0.4270	0.1312	0.4475	0.6044	0.5478	0.4784	0.4270	0.1312	0.1114
WT#1_ φ_{WT}	0.2457	0.2338	0.2058	0.1846	0.1609	0.1361	0.2429	0.2331	0.2208	0.1995	0.1821	0.1745	0.1001
WT#2_ φ_{WT}	0.2962	0.2755	0.2435	0.1992	0.1802	0.1268	0.3244	0.3137	0.2857	0.2408	0.2206	0.1700	0.0995

weights; this is related to the size s of the set of returned suspicious weights Θ_s in Line 18 of Alg. 4. The y -axis represents the detection rate $DR(\text{APP}, \text{SetMuts}_{\mathcal{M}^C_{-\varphi}})$. We report results with percentage of examined weights ranging from 0 to 20%, as it is not reasonable to have a fault localisation approach that considers too many weights. Table 6.7 reports the corresponding values of AUC_{DR} .⁸

In Fig. 6.6, the higher the percentage of weights considered (i.e., the value of s), the higher the *recall*, i.e., the probability that an approach returns in Θ_s the faulty weights (see Line 18 in Alg. 4). However, a higher s also means a higher *false detection*, i.e., a higher number of non-faulty weights; in this case, the effort required to identify the real fault among all weights in Θ_s increases (e.g., a repair approach taking in input fault localisation results may need to consider too many weights for repair). In order to assess this effect, Fig. 6.7 reports, for each benchmark $\mathcal{M}^C_{-\varphi}$ and approach APP, the *false discovery rate* $FDR(\text{APP}, \text{SetMuts}_{\mathcal{M}^C_{-\varphi}})$ (see Eq. 6.4) for increasing values

⁸In the supplementary website [103], we report DR plots and AUC_{DR} tables for all the suspiciousness metrics, for $s \leq 20\%$ and $s \leq 100\%$.

Table 6.8: Effectiveness of the compared approaches in terms of AUC_{FDR} with $s \leq 20\%$, using Kulczynski2 as suspiciousness metric SusMetr. The lower the value, the better (the best approach and second best approach are highlighted in gray and light gray, respectively)

	SPECTACLE _{uw}					ACT-BASED _{uw}	SPECTACLE _w					ACT-BASED _w	RANDOM	Theoretical Lower Bound
	Low	Mid-low	Mid	Mid-high	High		Low	Mid-low	Mid	Mid-high	High			
ACC#1_ φ_{ACC}^1	0.9214	0.9219	0.9243	0.9171	0.9262	0.9383	0.9199	0.9215	0.9233	0.9172	0.9266	0.9380	0.9366	0.7479
ACC#1_ φ_{ACC}^2	0.9082	0.9071	0.9112	0.9067	0.9116	0.9220	0.9073	0.9015	0.9043	0.8964	0.9055	0.9185	0.9349	0.7276
ACC#2_ φ_{ACC}^1	0.9668	0.9667	0.9679	0.9672	0.9661	0.9643	0.9672	0.9676	0.9668	0.9678	0.9665	0.9648	0.9715	0.8604
ACC#2_ φ_{ACC}^2	0.9667	0.9677	0.9673	0.9632	0.9622	0.9580	0.9672	0.9684	0.9666	0.9618	0.9616	0.9582	0.9708	0.8497
AFC#1_ φ_{AFC}^1	0.9640	0.9648	0.9666	0.9675	0.9689	0.9699	0.9641	0.9651	0.9665	0.9670	0.9682	0.9688	0.9719	0.8697
AFC#1_ φ_{AFC}^2	0.9672	0.9665	0.9651	0.9661	0.9673	0.9681	0.9673	0.9663	0.9651	0.9659	0.9672	0.9684	0.9712	0.8560
AFC#2_ φ_{AFC}^1	0.9724	0.9736	0.9748	0.9744	0.9744	0.9760	0.9720	0.9733	0.9743	0.9742	0.9743	0.9759	0.9813	0.9069
AFC#2_ φ_{AFC}^2	0.9747	0.9757	0.9765	0.9773	0.9781	0.9799	0.9737	0.9742	0.9746	0.9749	0.9756	0.9775	0.9811	0.9012
SC#1_ φ_{SC}	0.9195	0.9287	0.9243	0.9283	0.9347	0.9442	0.9207	0.9279	0.9232	0.9269	0.9321	0.9417	0.9555	0.7863
SC#2_ φ_{SC}	0.9462	0.9529	0.9569	0.9616	0.9673	0.9732	0.9469	0.9538	0.9582	0.9628	0.9690	0.9764	0.9804	0.8874
SC_R_ φ_{SC}	0.4102	0.3369	0.3605	0.3751	0.3888	0.4990	0.4102	0.3438	0.3605	0.3751	0.3888	0.4990	0.5179	0.3078
WT#1_ φ_{WT}	0.6912	0.6987	0.7082	0.7173	0.7311	0.7407	0.6921	0.6992	0.7016	0.7117	0.7202	0.7226	0.7553	0.3738
WT#2_ φ_{WT}	0.9562	0.9590	0.9609	0.9631	0.9656	0.9700	0.9545	0.9558	0.9574	0.9596	0.9618	0.9665	0.9708	0.8490

of s (up to 20% of weights).⁹

As explained in §6.5.4, a given level of FDR is unavoidable; the dashed black line in the figures represents the minimum value of FDR that is achievable. We notice that, in general, the FDR decreases at the beginning of the plot, until it reaches the value of s that allows to obtain the best trade-off between the correctly identified and the wrongly identified weights; after that point, FDR inevitably increases, as any additional weight returned in Θ_s is most likely not faulty. Table 6.8 reports the corresponding values of AUC_{FDR} . In the following, we answer the research questions reported in §6.5.1.

6.6.1 RQ1: EFFECTIVENESS OF SPECTACLE

In this RQ, we are interested in assessing whether the proposed approach is better than the baseline approaches in detecting the faulty weights. To do this, for each pair of approaches, we conducted pairwise comparisons between the baseline approaches and all versions of SPECTACLE

⁹In the supplementary website [103], we report FDR plots and AUC_{FDR} tables for all the suspiciousness metrics, for $s \leq 20\%$ and $s \leq 100\%$.

Table 6.9: RQ1, RQ2, RQ3 – Statistical comparison between each pair of approaches in terms of AUC_{DR} . (Legend. $\equiv$: there is no difference between the two approaches. $\checkmark, \checkmark\checkmark, \checkmark\checkmark\checkmark$: the approach on the rows is significantly better than the approach on the column with strength *small, medium, large*. $\times, \times\checkmark, \times\checkmark\checkmark$: the approach on the row is significantly worse than the approach on the column with strength *small, medium, large*)

		SPECTACLE _{uw}					ACT-BASED _{uw}	SPECTACLE _w					ACT-BASED _w	RANDOM
		Low	Mid-low	Mid	Mid-high	High		Low	Mid-low	Mid	Mid-high	High		
SPECTACLE _{uw}	Low	$\equiv$	$\times$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-low	$\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times$	$\times\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-high	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	High	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
ACT-BASED _{uw}		$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark\checkmark$
SPECTACLE _w	Low	$\checkmark\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\equiv$	$\times\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-low	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid	$\times$	$\times\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-high	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	High	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
ACT-BASED _w		$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark\checkmark$
RANDOM		$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\equiv$

Table 6.10: RQ1, RQ2, RQ3 – Statistical comparison between each pair of approaches in terms of AUC_{FDR} . (Legend. $\equiv$: there is no difference between the two approaches. $\checkmark, \checkmark\checkmark, \checkmark\checkmark\checkmark$: the approach on the rows is significantly better than the approach on the column with strength *small, medium, large*. $\times, \times\checkmark, \times\checkmark\checkmark$: the approach on the row is significantly worse than the approach on the column with strength *small, medium, large*)

		SPECTACLE _{uw}					ACT-BASED _{uw}	SPECTACLE _w					ACT-BASED _w	RANDOM
		Low	Mid-low	Mid	Mid-high	High		Low	Mid-low	Mid	Mid-high	High		
SPECTACLE _{uw}	Low	$\equiv$	$\times\checkmark$	$\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-low	$\checkmark\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark$	$\times\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	Mid	$\times$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	Mid-high	$\times$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark\checkmark$	$\checkmark\checkmark$	$\times$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	High	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark\checkmark$	$\equiv$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
ACT-BASED _{uw}		$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark\checkmark\checkmark$
SPECTACLE _w	Low	$\checkmark\checkmark$	$\times$	$\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\equiv$	$\times\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark\checkmark$
	Mid-low	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	Mid	$\checkmark$	$\times\checkmark$	$\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\times$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	Mid-high	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark\checkmark$	$\checkmark\checkmark$
	High	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\equiv$	$\checkmark$	$\checkmark\checkmark$
ACT-BASED _w		$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times$	$\equiv$	$\checkmark\checkmark\checkmark$
RANDOM		$\times\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark$	$\times\checkmark\checkmark$	$\equiv$

across all the benchmarks and different suspiciousness metrics in terms of AUC_{DR} and AUC_{FDR} , using the Wilcoxon signed-rank test and Cohen’s d effect size (see §6.5.4). Table 6.9 reports the results for AUC_{DR} . Table 6.10, instead, reports the results for AUC_{FDR} . First of all, by comparing the results in Tables 6.9 and 6.10, we observe that the results are almost always consistent: whenever an approach is better in terms of AUC_{DR} , it is also better in terms of AUC_{FDR} , with similar strength

of the effect size.¹⁰ So, for simplicity, in this and in the following RQs, we will focus our analysis on the results of AUC_{DR} , as also considering AUC_{FDR} does not provide any additional insight.

By looking at Table 6.9, we observe that any version of SPECTACLE is always better than RANDOM, with *large* strength. This demonstrates the effectiveness of the proposed fault localisation in finding faults.

Moreover, we observe that any version of SPECTACLE is better than the two versions of the baseline approach ACT-BASED ($ACT-BASED_{UW}$ and $ACT-BASED_W$), always with *medium* strength. This shows that the top- k criterion adopted by SPECTACLE is more effective than the *activation* criterion adopted by ACT-BASED for the construction of the spectrum.

Answer to RQ1: SPECTACLE can effectively localise the faulty DNN weights in AI-enabled CPSs and performs better than the two baseline approaches RANDOM and ACT-BASED.

6.6.2 RQ2: HYPERPARAMETER IMPACT ON THE EFFECTIVENESS OF SPECTACLE

In this RQ, we are interested in assessing which value of hyperparameter k provides the best results. As explained in §6.5.3, we experimented with five different values of different magnitudes, i.e., *Low*, *Mid-low*, *Mid*, *Mid-high*, and *High*.

In Table 6.9, we check the comparisons of the different versions of $SPECTACLE_{UW}$ among themselves, and those of the versions of $SPECTACLE_W$ among themselves. We notice that, for both $SPECTACLE_{UW}$ and $SPECTACLE_W$, *Low* and *Mid-low* outperform other top- k values, which indicates that a too large top- k value is not suitable for an effective fault localisation. *Mid-low* is also always better than *Low*, once with *small* effect size (for $SPECTACLE_{UW}$) and once with *medium* effect size (for $SPECTACLE_W$); still, we think that also *Mid-low* could be effective in some cases.

These results show that SPECTACLE can obtain the best performance without selecting too

¹⁰The only exceptions are the comparison between $SPECTACLE_{UW}$, *Low* and $SPECTACLE_W$, *Mid-low*, and between $SPECTACLE_{UW}$, *Mid-high* and $SPECTACLE_W$, *Mid-high*, in which the result changes between AUC_{DR} and AUC_{FDR} . However, in both cases, the strength of the effect size is *small*, so not too significant.

Table 6.11: RQ3 – Comparison between $\text{SPECTACLE}_{\text{UW}}$ and $\text{SPECTACLE}_{\text{W}}$ (better approach in gray)

	UW	=	W		UW	=	W
$\text{ACC\#1}_{-\varphi_{\text{ACC}}^1}$	27.2%	10.1%	62.7%	$\text{ACC\#1}_{-\varphi_{\text{ACC}}^2}$	11.0%	8.6%	80.4%
$\text{ACC\#2}_{-\varphi_{\text{ACC}}^1}$	50.4%	10.5%	39.1%	$\text{ACC\#2}_{-\varphi_{\text{ACC}}^2}$	34.0%	10.8%	55.2%
$\text{AFC\#1}_{-\varphi_{\text{AFC}}^1}$	29.2%	4.2%	66.6%	$\text{AFC\#1}_{-\varphi_{\text{AFC}}^2}$	22.4%	0.5%	77.0%
$\text{AFC\#2}_{-\varphi_{\text{AFC}}^1}$	30.5%	1.5%	68.0%	$\text{AFC\#2}_{-\varphi_{\text{AFC}}^2}$	10.3%	0.6%	89.1%
$\text{SC\#1}_{-\varphi_{\text{SC}}}$	51.3%	1.6%	47.1%	$\text{SC\#2}_{-\varphi_{\text{SC}}}$	63.1%	10.7%	26.2%
$\text{SC}_R_{-\varphi_{\text{SC}}}$	27.0%	48.0%	25.0%	$\text{WT\#1}_{-\varphi_{\text{WT}}}$	23.2%	0.8%	76.0%
$\text{WT\#2}_{-\varphi_{\text{WT}}}$	9.9%	2.1%	88.0%	All	31.8%	5.3%	62.9%

many unnecessary weights.

Answer to RQ2: Among the five top- k value categories, *Low* and *Mid-low* can localise faulty weights more effectively.

6.6.3 RQ3: $\text{SPECTACLE}_{\text{UW}}$ vs. $\text{SPECTACLE}_{\text{W}}$

In this RQ, we are interested in assessing the effectiveness of $\text{SPECTACLE}_{\text{UW}}$ and $\text{SPECTACLE}_{\text{W}}$, i.e., the contribution of using a constant window or the Hamming window (see §6.4.3.1). From Table 6.9, we observe that $\text{SPECTACLE}_{\text{W}}$ with *Low* and *Mid-low* is always better than any version of $\text{SPECTACLE}_{\text{UW}}$, almost always with strength *medium* (except for one case in which the strength is *small*).

To better analyse the results, in Table 6.11, we report, for each benchmark $\mathcal{M}^C_{-\varphi}$, the percentage of times (across different values of k , suspiciousness metrics, and values of s) that $\text{SPECTACLE}_{\text{UW}}$ is better than $\text{SPECTACLE}_{\text{W}}$ (column UW), that $\text{SPECTACLE}_{\text{W}}$ is better than $\text{SPECTACLE}_{\text{UW}}$ (column W), and that they are the same (column =). We observe that $\text{SPECTACLE}_{\text{W}}$ outperforms $\text{SPECTACLE}_{\text{UW}}$ on 9/13 of the benchmarks, to varying degrees, ranging from a 21.2% difference ($\text{ACC\#2}_{-\varphi_{\text{ACC}}^2}$) to a 78.8% ($\text{AFC\#2}_{-\varphi_{\text{AFC}}^2}$). By considering all the benchmarks together (line “All”), $\text{SPECTACLE}_{\text{W}}$ is better in 62.9% of the cases (i.e., same benchmark, same k , same suspiciousness

Table 6.12: RQ4 – Statistical comparison between each pair of suspiciousness metrics in terms of AUC_{DR} . (Legend. $\equiv$: there is no difference between the two metrics. $\checkmark$, $\checkmark\checkmark$, $\checkmark\checkmark\checkmark$: the metric on the rows is significantly better than the metric on the column with strength *small*, *medium*, *large*. $\times$, $\times\times$, $\times\times\times$: the metric on the row is significantly worse than the metric on the column with strength *small*, *medium*, *large*)

	Tarantula	Ochiai	D*	Jaccard	Kulczynski2
Tarantula	$\equiv$	$\times\times$	$\times\times$	$\times\times$	$\times\times$
Ochiai	$\checkmark\checkmark$	$\equiv$	$\checkmark\checkmark$	$\checkmark$	$\times$
D*	$\checkmark\checkmark$	$\times\times$	$\equiv$	$\times$	$\times\times$
Jaccard	$\checkmark\checkmark$	$\times$	$\checkmark$	$\equiv$	$\times$
Kulczynski2	$\checkmark\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark$	$\equiv$

metric, and same s), while $SPECTACLE_{UW}$ is better in only 31.8% of the cases.

These results confirm our intuition that weighting more the weights that are relevant close to the violation episode (as done by the Hamming window used in $SPECTACLE_W$) is better than considering all the inferences equally (as done by the constant window used in $SPECTACLE_{UW}$).

Answer to RQ3: $SPECTACLE_W$ outperforms $SPECTACLE_{UW}$ in most cases, which confirms that the Hamming window can enhance the effectiveness of $SPECTACLE$.

6.6.4 RQ4: SUSPICIOUSNESS METRIC IMPACT ON THE EFFECTIVENESS OF

$SPECTACLE$

In this RQ, we are interested in assessing the influence of the different suspiciousness metrics on the effectiveness of $SPECTACLE$. Similarly to the previous RQs, we performed pairwise comparisons between the five suspiciousness metrics across all the benchmarks and different hyper-parameters in terms of AUC_{DR} and AUC_{FDR} , using the Wilcoxon signed-rank test and the Cohen’s d effect size (see §6.5.4). Results are reported in Tables 6.12 and 6.13, respectively.

First of all, by comparing the two tables, we observe that the results are almost always consistent (except for the comparison of D^* and Jaccard), i.e., whenever an approach is better of another in terms of AUC_{DR} , it is also better in terms of AUC_{FDR} , with almost the same strength of effect size.

Table 6.13: RQ4 – Statistical comparison between each pair of suspiciousness metrics in terms of AUC_{FDR} . (Legend. $\equiv$: there is no difference between the two metrics. $\checkmark$, $\checkmark\checkmark$, $\checkmark\checkmark\checkmark$: the metric on the rows is significantly better than the metric on the column with strength *small*, *medium*, *large*. $\times$, $\times\times$, $\times\times\times$: the metric on the row is significantly worse than the metric on the column with strength *small*, *medium*, *large*)

	Tarantula	Ochiai	D*	Jaccard	Kulczynski2
Tarantula	$\equiv$	$\times\times$	$\times\times$	$\times\times$	$\times\times$
Ochiai	$\checkmark\checkmark$	$\equiv$	$\checkmark$	$\checkmark$	$\times$
D*	$\checkmark\checkmark$	$\times$	$\equiv$	$\checkmark$	$\times\times$
Jaccard	$\checkmark\checkmark$	$\times$	$\times$	$\equiv$	$\times$
Kulczynski2	$\checkmark\checkmark$	$\checkmark$	$\checkmark\checkmark$	$\checkmark$	$\equiv$

So, for simplicity, in the following we will focus on the description of the results of AUC_{DR} .

From Table 6.12, we observe that Tarantula is the worst metric, never better than any other metric. This result seems to be in agreement with reports about the low effectiveness of Tarantula in SBFL for classic code [85, 125]. Kulczynski2, instead, is the best metric, always better than the other metrics (half of the times with strength *medium*). A possible explanation is that, unlike Tarantula, Kulczynski2 gives more importance to *ef* (i.e., the weights that are relevant in failure test cases) which has been shown to be important in SBFL. On the other hand, Ochiai and D*, that also focus on *ef*, do not obtain such good results. The reason is that they may focus too much on *ef* [170]; while this is very important in fault localisation for code, it can be excessive for DNNs. These results combined show that Kulczynski2 provides the right balance between *ef* and *ep*.

Answer to RQ4: Among the five suspiciousness metrics, Kulczynski2 is the most effective, while Tarantula is always the worst.

6.6.5 RQ5: USEFULNESS OF IDENTIFIED WEIGHTS BY SPECTACLE

In this RQ, we are interested in assessing whether the weights identified by SPECTACLE can be used to improve the performance of the AI-enabled CPS. As explained in §6.5.4, we select three faulty benchmarks ($\mathcal{M}_{f1}^C$, $\mathcal{M}_{f3}^C$, and $\mathcal{M}_{f5}^C$) for each of the four CPSs (for each CPS, we select the one with the most complex DNN controller), and we repair it using the search-based approach

Table 6.14: RQ5 – Results of repair of AI-enabled CPSs using the weights identified by SPECTACLE

	ACC#2- φ_{ACC}^1			AFC#2- φ_{AFC}^1			SC#2- φ_{SC}			WT#2- φ_{WT}		
	$\mathcal{M}_{f1}^C$	$\mathcal{M}_{f3}^C$	$\mathcal{M}_{f5}^C$	$\mathcal{M}_{f1}^C$	$\mathcal{M}_{f3}^C$	$\mathcal{M}_{f5}^C$	$\mathcal{M}_{f1}^C$	$\mathcal{M}_{f3}^C$	$\mathcal{M}_{f5}^C$	$\mathcal{M}_{f1}^C$	$\mathcal{M}_{f3}^C$	$\mathcal{M}_{f5}^C$
SR before repair	73	11	55	47	21	53	16	74	82	68	64	44
SR after repair	100	100	100	100	100	82	100	100	100	100	100	100

proposed in [101], using the suspicious weights returned by SPECTACLE as search variables. Table 6.14 reports the experimental results. The table reports the values of the safety rate SR on the original faulty AI-enabled CPS *before repair*, and on the AI-enabled CPS repaired using the weights identified by SPECTACLE. We observe that the repair process is almost always able to completely repair the system (i.e., $SR = 100\%$); the only exception is $\mathcal{M}_{f5}^C$ of AFC#2- φ_{AFC}^1 for which the final SR is 82%. This shows that SPECTACLE can indeed identify weights that are responsible for misbehaviour.

Answer to RQ5: SPECTACLE identify weights that are indeed responsible for the misbehaviour of the AI-enabled CPS and, therefore, can be targeted when improving the system performance by, e.g., search-based repair.

6.6.6 RQ6: EFFICIENCY OF SPECTACLE

In this RQ, we are interested in assessing the computational cost of SPECTACLE. Table 6.15 reports, for each correct benchmark $\mathcal{M}^C_{-\varphi}$, the average time taken by SPECTACLE across five faulty benchmarks of $\mathcal{M}^C_{-\varphi}$. Specifically, the table reports the time spent in executing the test suite TS , and the time spent in fault localisation. This latter is further divided into the time spent in calculating the forward impact, and the time spent in calculating the execution spectrum and the suspiciousness score. We observe that most of time is taken by the execution of the test suite TS , taking on average 311.6 secs.; this time will increase proportionally with the size TS . Regarding the actual time taken by fault localisation, we observe that the forward impact calculation is the most costly one (189.5 secs. on average). On the other hand, the computation of the spectrum and

Table 6.15: RQ6 – Computational cost of SPECTACLE (secs.)

	TS execution	Fault localisation	
		forward impact	execution spectrum and suspiciousness score
ACC#1- φ_{ACC}^1	303.1	216.4	28.5
ACC#1- φ_{ACC}^2	306.7	246.2	41.8
ACC#2- φ_{ACC}^1	330.9	281.5	44.7
ACC#2- φ_{ACC}^2	340.7	259.9	61.3
AFC#1- φ_{AFC}^1	339.6	135.7	37.8
AFC#1- φ_{AFC}^2	343.2	161.8	46.8
AFC#2- φ_{AFC}^1	389.7	226.2	45.1
AFC#2- φ_{AFC}^2	382.4	242.1	55.9
SC#1- φ_{SC}	324.1	257.2	47.5
SC#2- φ_{SC}	389.0	256.6	83.0
SC_R- φ_{SC}	29.8	20.2	14.5
WT#1- φ_{WT}	234.5	76.7	9.3
WT#2- φ_{WT}	337.3	82.7	27.2
Avg.	311.6	189.5	41.8

of the suspiciousness score is faster (41.8 secs. on average). Regarding the different benchmarks, we observe that those with smaller DNN controllers (such as SC_R, WT#1, and WT#2) tend to take less time in fault localisation; this is reasonable, as they contain less weights that need to be checked. On the other hand, the execution time of fault localisation also depends on the number of failing tests; indeed, on failing tests, fewer inferences are checked (i.e., those till the specification violation) in comparison to the passing tests, so reducing time.

Answer to RQ6: The most computationally expensive part of SPECTACLE is the execution of the test suite, followed by computation of the forward impact.

6.7 THREATS TO VALIDITY

Here we discuss some threats that can affect the validity of the approach, following the typical classification of *construct validity*, *conclusion validity*, *internal validity*, and *external validity* [151].

Construct Validity. The metrics used in the assessment of the approach could not reflect the object of the investigation. In our case, since we are interested in measuring the effectiveness of SPECTACLE, we mutate some weights of the target DNN and examine whether those mutated weights can be localised, i.e., ranking at the top s . In this sense, the results largely rely on the value s and the mutations we made. To mitigate this threat, we examine the performance on different settings of top s , as done in other DNN fault localisation works [44]. Moreover, SPECTACLE requires the setting of a threshold k to decide whether a weight can be counted as executed or not. A too small k could result in considering too few weights as executed. To mitigate this threat, we tried different top- k values and checked the performance of SPECTACLE.

Conclusion Validity. The random behaviour of the RANDOM approach can affect the final result. Following the guideline of Arcuri and Briand [12], we executed the RANDOM approach 100 times for each faulty model, and we reported average results. SPECTACLE and ACT-BASED, instead, are deterministic and there is no need to repeat them multiple times.

Internal Validity. A threat of this type could be that the experimental results are obtained by chance. For example, an *instrumentation* threat (e.g., a faulty implementation) may affect the execution of the experiments. To mitigate this threat, we have extensively tested the implementation of all the compared approaches.

External Validity. One such a threat is related to the generalizability of SPECTACLE. To mitigate this issue, we conduct our work with a general model of AI-enabled CPS that contains a black-box physical plant and a DNN controller. Moreover, we experimented with different types of controllers, i.e., feed-forward neural networks and recurrent neural networks.

6.8 CONCLUSION AND FUTURE WORK

In this work, we propose the framework SPECTACLE, that localises the DNN controller weights responsible for the system specification violation, by exploiting the sequence of DNN controller inferences. Our technical novelty consists in the construction of the execution spectrum, in which we take the system-level specification as the correctness criterion for the DNN controller, and use forward impact to calculate the relevance of each neuron weight to the DNN controller inference. Our experimental results show the effectiveness of SPECTACLE, based on the comparison with two baseline approaches.

As a future work, we plan to devise other methods for computing forward impact, because this is the core factor that affects the effectiveness of the approach. For example, in this work, we consider the activation of a neuron as a possible cause of a wrong DNN control action. However, it could be that the non-activation of a neuron is responsible for a wrong action. As future work, we plan to devise a fault localisation approach in which also non-activation is considered as a possible cause of misbehaviour.

In SPECTACLE, we consider all the inferences preceding the violation to do fault localisation. However, since it is more likely that a wrong control decision happens close to the violation, in SPECTACLE_w we use the Hamming window to weight more the inferences close to the violation. Since the tests considered in this work are not too long (i.e., they are just long enough to check whether they satisfy the specification), such a weighted approach is already very effective, as shown by our experiments. Still, the approach could benefit from a more detailed identification of the moment “when” the control decision starts to be responsible for the misbehaviour. The theory of *actual causality* by Halpern and Pearl [64] requires three conditions to hold to claim that an event A is the cause of an event B : **(C1)** A must occur before B ; **(C2)** if A does not happen, but all the other conditions of the state remain the same, then B does not happen; **(C3)** A is minimal, i.e., there is no subset of A that satisfies **C1** and **C2**. In our context, we consider A as

the control action and B as the violation of the specification. It is easy to see that **C1** holds, as we check the time-points preceding the violation. However, guaranteeing condition **C2** is very difficult, as it requires to find, for each observed control action A in a test, another test in which only the control action A does not happen, but the remaining part of the state is the same. For an AI-enabled CPS, this would require to override the behaviour of the DNN controller and force a different control action to see if the violation does not occur. We leave as future work to design an approach that can do this in an effective manner.

7 | SEARCH-BASED REPAIR OF DNN CONTROLLERS OF AI-ENABLED CYBER-PHYSICAL SYSTEMS GUIDED BY SYSTEM-LEVEL SPECIFICATIONS

In this chapter, we propose a novel DNN repair approach that is guided by system-level specifications. The approach takes in input a system-level specification, some tests violating the specification, and some faulty DNN weights. The approach searches for alternative weight values with the goal of fixing the behaviour on the failing tests without breaking the passing tests. We also propose a heuristic that allows us to accelerate the search by avoiding the execution of some tests. Experiments on real-world AI-enabled CPSs show that the approach effectively repairs their controllers.

7.1 INTRODUCTION

Cyber-physical systems (CPS) are systems in which physical plants are controlled by computer systems, and they have been successfully applied in different domains, even safety-critical, like transportation, healthcare, etc. In order to improve their capabilities, CPSs are increasingly

adopting *deep neural networks* (DNNs) as controllers, leading to the emergence of a new class of systems called *AI-enabled CPS* [97, 122, 134, 176]. For example, in autonomous driving systems, DNN controllers permit to process complex environmental data collected by sensors and produce intelligent control decisions.

Despite their numerous benefits, DNN controllers can be faulty and can lead to failures of their AI-enabled CPSs; in safety-critical domains, such as autonomous driving systems, such failures can constitute serious safety issues. Debugging and improving DNNs is particularly difficult due to their data-driven computation paradigm and to the fact that the decision logic is not explicit but encoded in the setting of the weights. In order to tackle this problem, *DNN repair* is an emerging technique that allows performing localized modifications to a DNN with the goal to improve its performance [50, 89–91, 133, 137, 141, 164, 166].

However, the application of existing DNN repair approaches to DNN controllers of AI-enabled CPSs is not straightforward. Indeed, state-of-the-art repair techniques are usually applied for classification problems and require to have ground-truth labels for the DNN inputs used during repair. In an AI-enabled CPS, such ground-truth information regarding the DNN controller is not available; indeed, given a control decision of the DNN controller, it is not possible to decide whether it is correct or not. On the other hand, requirements are available regarding the expected behaviour of the controlled physical plant. Therefore, it is possible to specify *system-level specifications* predicting the behaviour of the controlled physical plant: if these specifications are satisfied, we can (indirectly) conclude that the DNN controller is behaving correctly; instead, a violation of the system specification is an indication that the controller provided a wrong control decision.

In this paper, based on this intuition, we propose CONTRREP, an approach to repair DNN controllers of AI-enabled CPSs that is guided by the evaluation of a system-level specification. Specifically, CONTRREP takes as input an AI-enabled CPS $\mathcal{M}^{C_{orig}}$ composed of a physical plant $\mathcal{M}$ and a DNN controller C_{orig} , a system specification φ over $\mathcal{M}^{C_{orig}}$ written in Signal Temporal Logic

(STL), and a test suite TS , in which some tests in TS satisfy the specification (called TS_{orig}^+) and some do not (called TS_{orig}^-). CONTRREP also takes in input a set of *suspicious weights* SW of the DNN controller C_{orig} , that have been identified by some fault localization technique [43, 45, 133]; these weights are those responsible for the failures in TS_{orig}^- . CONTRREP adopts a search-based approach to find an alternative setting for weights SW , with the goal of making the system pass the tests in TS_{orig}^- , and not break those in TS_{orig}^+ .

During the search, CONTRREP requires, for each fitness function evaluation, to execute the candidate AI-enabled CPS $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$ over all the tests in TS . This can be computationally expensive, in particular, if the test execution relies on some expensive simulation. To tackle this problem, we propose a version of the repair approach (called CONTRREP_{FAST}), that adopts a heuristic to compute the fitness function faster. The heuristic method exploits the STL *quantitative robustness* that, for a given system execution, not only tells *whether* the specification is satisfied or not, but also provides a real value that tells *how robustly* it does so (i.e., “how far” it is from violation/satisfaction): negative robustness means that the specification is violated, with lower values indicating more severe violations; positive robustness, instead, means that the specification is satisfied, with higher values indicating stronger satisfaction. The heuristic method sorts the failed tests TS_{orig}^- by decreasing robustness and evaluates them in order: as soon as a test is not repaired (i.e., it is not satisfied by the candidate AI-enabled CPS $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$), the remaining tests with lower robustness are not evaluated and considered as failing, as it is unlikely that they could be repaired (as they are more difficult to repair). Similarly, the passing tests TS_{orig}^+ are sorted by increasing robustness and evaluated in order: as soon as a test passes (i.e., it is also satisfied by the candidate AI-enabled CPS $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$), the remaining tests with higher robustness are not evaluated and it is assumed that they will also pass (as they are more difficult to break).

Paper structure. §7.2 formalizes the problem, and §7.3 presents the proposed approach. §7.4 introduces the design of the experiments, and §7.5 presents the experimental results. Finally, §7.6 discusses threats that may affect the validity of the approach, and §7.7 concludes the paper.

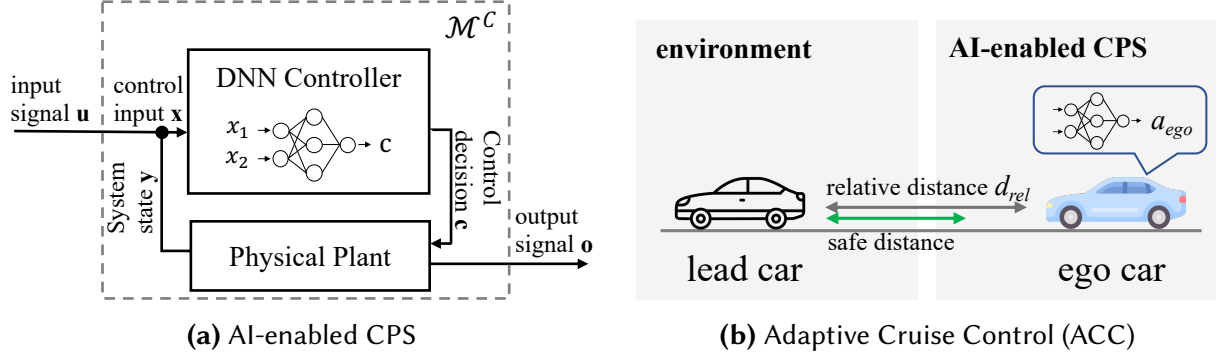


Figure 7.1: AI-enabled CPS model and an example ACC

7.2 PROBLEM DEFINITION

7.2.1 AI-ENABLED CYBER-PHYSICAL SYSTEMS

Definition 4 (AI-enabled CPS). As shown in Fig. 7.1, an AI-enabled CPS consists primarily of a DNN controller C and a physical plant $\mathcal{M}$. At a time instant t , the DNN controller C gives a control decision $\mathbf{c}(t)$ that decides the evolution of the state of the physical plant, based on external input signal $\mathbf{u}(t)$ and the state of the plant $\mathbf{y}(t)$. The dynamics of the plant is formulated as follows: $\dot{\mathbf{y}}(t) = \mathcal{M}(\mathbf{y}(t), \mathbf{c}(t))$, where $\mathcal{M}$ is a black-box function such that it can accommodate complex dynamics and third-party confidential components. Overall, the whole system $\mathcal{M}^C$ can be viewed as a function that maps an input signal $\mathbf{u}$ to an output signal $\mathbf{o}$. $\triangleleft$

The following example uses the ACC system (later introduced in §7.4.2) to illustrate the operation process of a CPS system.

Example 3. *Adaptive Cruise Control (ACC)*, as shown in Fig. 7.1(b), is an advanced control system that aims to maintain a safe distance of a vehicle (*ego car*) from the preceding vehicle (*lead car*). In Fig. 7.1(b), the ego car is considered as the physical plant, under the control of a DNN controller. The lead car is considered as the surrounding environment, and its speed and position detected by the sensors of the ego car are treated as the external input signal $\mathbf{u}(t)$ to the 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 a vector $\mathbf{x}(t)$ and sent to the controller C as its input for decision-making. Based on the input, the DNN controller C computes a control signal, i.e., the acceleration of the ego car, to control the motion of the plant. $\triangleleft$

As shown in Def. 4, the state evolution of the whole AI-enabled CPS is primarily decided by the DNN controller, and therefore, we take it as our main target for repairing unsafe system behaviour. While our technique is applicable to various types of neural networks, we elaborate on the widely-adopted fully-connected DNNs [133, 176].

Definition 5 (DNN controller). A DNN controller C is a function that, at each timestamp, maps an input vector $\vec{x}$ to an output scalar c (i.e., a control decision). It consists of L hidden layers and an output layer. The i -th hidden layer computes an output vector $\vec{x}_i$ by $\vec{x}_i = \sigma(\mathcal{W}_i \vec{x}_{i-1} + \vec{b}_i)$, where $\vec{x}_{i-1}$ is the output of the $(i-1)$ -th hidden layer, $\mathcal{W}_i$ is a matrix of *weights* and $\vec{b}_i$ is a vector of *bias* at the i -th layer, and σ is a non-linear *activation function*, such as *ReLU*, *sigmoid* and *tanh*. The output layer computes the output c of the DNN controller, by taking the weighted sum over different components of the output $\vec{x}(i)$ of the last hidden layer. $\triangleleft$

In this paper, we follow the literature [90, 133, 137, 141] and consider the *weights* as the target of repair. Specifically, we identify by $\mathcal{W}$ the set of weights across all the hidden layers in a DNN controller.

7.3 PROPOSED REPAIR APPROACH

In this section, we describe CONTRREP, our proposed approach for repairing DNN controllers of AI-enabled CPSs. Note that state-of-the-art DNN repair approaches are not applicable to DNN controllers, as they require the ground truth of the DNN behaviours that are not available for DNN controllers. We share with existing DNN repair approaches the basic assumption that failures are

due to a few components of the DNN that are not properly tuned, and only these should be changed to fix the failures.

CONTRREP takes in input an AI-enabled CPS $\mathcal{M}^{C_{orig}}$, a system specification φ , and a test suite TS of input sequences for $\mathcal{M}^{C_{orig}}$. TS should be generated with some generation technique [130, 169] that guarantees enough coverage of the input space. Some tests in the test suite satisfy φ and some do not; for convenience, we partition the test suite as $TS_{orig}^+ \cup TS_{orig}^-$, where:

$$TS_{orig}^+ = \{t \in TS \mid \mathcal{M}^{C_{orig}}(t) \models \varphi\}$$

$$TS_{orig}^- = \{t \in TS \mid \mathcal{M}^{C_{orig}}(t) \not\models \varphi\}$$

The correctness of the DNN controller C_{orig} depends on how many tests satisfy the specification. We introduce a *correctness measure* defined as:

$$CM(\mathcal{M}^{C_{orig}}, \varphi, TS) = \frac{|TS_{orig}^+|}{|TS|} \quad (7.1)$$

In the following, in §7.3.1, we explain how the weights to repair are selected; in §7.3.2, we introduce our proposed repair approach, and in §7.3.3, we present a heuristic to speed up the repair process.

7.3.1 IDENTIFICATION OF THE WEIGHTS TO REPAIR

The components to modify during repair are usually identified by some fault localization (FL) technique, inspired by fault localization for classic code [155]. Different FL approaches have been proposed for DNNs [26, 45, 133], mainly targeting the weights of the network; in our work, we follow this trend, and we also consider the weights as the target of the repair. The main intuition behind these approaches is that the weights that are activated during failures are more likely to be faulty; FL approaches identify these weights and return them as *suspicious weights* SW .

Our approach assumes that some FL approach has been applied to the weights of C_{orig} , considering the results of the execution of the test suite TS (i.e., TS_{orig}^+ and TS_{orig}^-), and has identified a set of *suspicious weights*. We want to point out that, similarly to FL for classic code, DNN FL is usually able to correctly identify the faulty weights in SW , but also identifies other additional weights that are not responsible for the failures. However, it is not known which of the returned weights are the faulty ones.

In our experiments, we are interested in assessing the capabilities of the repair approach and not of the FL approach. Therefore, in order to perform a proper assessment, we need to know the quality of the FL results. To do so, we proceed as usually done in the controlled experiments for automatic program repair [56]. Namely, we inject faults in the weights of the controller C_{orig} ; then, we build the set of suspicious weights SW , by including these faulty weights plus others that are actually correct. In this way, we reproduced in a controlled way the noise that can be introduced by an FL approach in the suspicious weights SW . We provide more details in §7.4.

7.3.2 CONTRREP – SEARCH-BASED REPAIR OF THE DNN CONTROLLER

In this section, we present our proposed repair approach. We cast the repair problem as a search problem, whose goal is to find better values for the suspicious weights SW . Specifically, we define a single-objective search problem as follows.

The *search variables* $\bar{x}$ of the approach are the possible alternative values for SW , i.e., :

$$\bar{x} = [x_1, \dots, x_{|SW|}]$$

A *search individual* $\bar{v}$ is an assignment to the search variables that can be generated during the search. A search individual identifies a new DNN controller whose weights SW are set to the values $\bar{v}$ and the other weights keep the same value as in C_{orig} . We will identify it as $C_{SW \leftarrow \bar{v}}$; we will also identify as $\mathcal{M}^{C_{SW \leftarrow \bar{v}}}$ the AI-enabled system equipped with the modified controller.

In order to define the *search space* for the search variables $\bar{x}$, we proceed as follows. The search space of a variable x depends on the original value v_{orig} of the corresponding weight w in C_{orig} . Namely, we allow to modify the weight in a way that its contribution to the activation function is at most δ times the original one or at least a $1/\delta$ fraction of it, where $\delta \in \mathbb{R}_{>0}$ is a parameter of the search. This is obtained by defining the search space of x as $\left[v_{orig} \cdot \delta^{-\text{sign}(v_{orig})}, v_{orig} \cdot \delta^{\text{sign}(v_{orig})} \right]$, where $\text{sign}(v_{orig}) \in \{-1, 1\}$ identifies the sign of v_{orig} . In the experiments, we set $\delta = 2$.

The goal of the search is to find an alternative controller $C_{SW \leftarrow \bar{v}}$ that can fix (i.e., make them pass) the tests failing with the original controller (i.e., TS_{orig}^-) and not break those that were passing (i.e., TS_{orig}^+). So, given an individual $\bar{v}$, the *fitness function* that must be maximized is:

$$fit(\bar{v}) = CM(\mathcal{M}^{C_{SW \leftarrow \bar{v}}}, \varphi, TS) \quad (7.2)$$

We adopt Differential Evolution (DE) [135] as the underlying search algorithm as it has been successfully used in other DNN repair approaches for image classifiers like Arachne [133]. However, any other population-based algorithm could be adopted.

At the end of the search, the approach returns the individual that obtained the best performance. We will identify the corresponding controller and AI-enabled CPS as C_{best} and $\mathcal{M}^{C_{best}}$.

7.3.3 CONTRREP_{FAST} – SPEEDING UP THE FITNESS COMPUTATION

The calculation of the fitness as described in §7.3.2 can be quite expensive, in particular, if the test suite TS contains several tests and the cost of executing one test is high (e.g. when relying on simulators). Therefore, in this section, we propose a modified version of the approach (called CONTRREP_{FAST}) that adopts a fitness function that still tries to compute the correctness measure, but in a faster way:

$$fit_{FAST}(\bar{v}) = APPROXFIT(\mathcal{M}^{C_{SW \leftarrow \bar{v}}}, \varphi, TS, TS_{orig}^+, TS_{orig}^-) \quad (7.3)$$

The fitness computation relies on the heuristic implemented in the method APPROXFIT that tries to avoid running all the tests by *estimating* which is the possible assessment (i.e., pass or fail) of some of them. The heuristics makes use of the quantitative robustness value associated with each test execution (see Def. 2) and it is based on the following intuitions:

- let us consider a failing test t in TS_{orig}^- (i.e., failing with the original system $\mathcal{M}^{C_{orig}}$) with negative robustness; if t is not repaired with $\mathcal{M}^{C_{SW \leftarrow \bar{v}}}$ (i.e., the robustness $\llbracket \mathcal{M}^{C_{SW \leftarrow \bar{v}}}(t), \varphi \rrbracket$ of the test t is still negative when executed with $\mathcal{M}^{C_{SW \leftarrow \bar{v}}}$), all the tests in TS_{orig}^- having lower robustness will likely be not repaired as well. The intuition is that the robustness value identifies the “difficulty to repair” and if it is not possible to repair “easy tests” (i.e., tests close to positive robustness), it is probably also not possible to repair more difficult tests;
- let us consider a passing test t in TS_{orig}^+ (i.e., passing with the original system $\mathcal{M}^{C_{orig}}$) with positive robustness; if t is still satisfied (i.e., the robustness $\llbracket \mathcal{M}^{C_{SW \leftarrow \bar{v}}}(t), \varphi \rrbracket$ of the test is still positive with $\mathcal{M}^{C_{SW \leftarrow \bar{v}}}$), all the tests in TS_{orig}^+ having higher robustness will likely be still passing as well. The intuition is that if the repair action does not break tests that are close to being violated (i.e., tests close to negative robustness), most probably also tests with higher robustness (i.e., farther from violation) will be preserved.

Based on the previous intuitions, the heuristic APPROXFIT is defined as presented in Alg. 5.

It works as follows:

- it sorts the tests failing with $\mathcal{M}^{C_{orig}}$ by decreasing robustness (line 2) and those passing with $\mathcal{M}^{C_{orig}}$ by increasing robustness (line 3);
- it visits the failing tests in decreasing order of robustness (line 5). For each test t_i^- :
 - if the test passes with the AI-enabled CPS $\mathcal{M}^{C_{SW \leftarrow \bar{v}}}$ under evaluation, i.e., the test is repaired (line 6), the counter of passing tests TP is increased (line 7);
 - otherwise, if the test fails, the approach assumes that also the remaining tests with lower robustness will fail, and so it stops the iteration (line 9);
- it visits the passing tests in increasing order of robustness (line 10). For each test t_i^+ :

Algorithm 5 APPROXFIT – Heuristic for fitness computation**Require:** TS : test suite**Require:** TS_{orig}^- : tests failing in $\mathcal{M}^{C_{orig}}$ **Require:** TS_{orig}^+ : tests passing in $\mathcal{M}^{C_{orig}}$ **Require:** $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$: AI-enabled CPS under assessment**Ensure:** approximated fitness

```

1: function APPROXFIT( $TS, TS_{orig}^-, TS_{orig}^+, \mathcal{M}^{C_{SW} \leftarrow \bar{v}}$ )
2:    $[t_1^-, \dots, t_{|TS_{orig}^-|}^-] \leftarrow \text{sortDecrRob}(TS_{orig}^-)$ 
3:    $[t_1^+, \dots, t_{|TS_{orig}^+|}^+] \leftarrow \text{sortIncrRob}(TS_{orig}^+)$ 
4:    $TP \leftarrow 0$  ▷ # of tests passing with  $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$ 
5:   for  $i \leftarrow \{1, \dots, |TS_{orig}^-|\}$  do
6:     if  $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}(t_i^-) \models \varphi$  then
7:        $TP \leftarrow TP + 1$ 
8:     else
9:       break
10:  for  $i \leftarrow \{1, \dots, |TS_{orig}^+|\}$  do
11:    if  $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}(t_i^+) \models \varphi$  then
12:       $TP \leftarrow TP + ((|TS_{orig}^+| - i) + 1)$ 
13:    break
14:  return  $\frac{TP}{|TS|}$ 

```

- if the test passes with the AI-enabled CPS $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$ under evaluation, i.e., the test has not been broken (line 11), the approach assumes that also all the other tests with higher robustness will not be broken, and so the counter of passing tests TP is increased accordingly without executing those tests (line 12), and the iteration is stopped (line 13);
- otherwise, the iteration continues by executing the next test with higher robustness.
- the ratio of passing tests over the total number of tests is returned as computed approximated fitness of $\mathcal{M}^{C_{SW} \leftarrow \bar{v}}$ (line 14).

7.4 EXPERIMENT DESIGN

We here present the design of the experiments. All the code, benchmarks, and experimental results are reported online [102].

7.4.1 RESEARCH QUESTIONS

We identified the following research questions (RQs) to assess the proposed approach:

RQ1: How is the repair performance of CONTRREP? Is it affected by the quality of the fault localization results?

In this RQ, we want to assess if CONTRREP can actually repair DNN controllers. Moreover, we want to assess whether having imprecise fault localization results (i.e., containing weights that are not faulty) affects the repair effectiveness.

RQ2: Does CONTRREP_{FAST} reduce the execution time of the repair approach? Does it affect the repair performance?

In this RQ, we want to assess whether the heuristic implemented by CONTRREP_{FAST} is indeed effective in reducing the repair time, and which is the reduction that must be paid in terms of repair performance.

RQ3: To what extent does CONTRREP fix failing tests and break passing tests?

In this RQ, we want to assess whether, while trying to repair the failing tests, CONTRREP also breaks some passing tests.

7.4.2 BENCHMARKS

To assess the effectiveness of CONTRREP, we consider two industrial-level CPSs to be used as plant $\mathcal{M}$, taken from the automotive domain. Both CPSs have been considered in previous works: *Adaptive Cruise Control* [74, 109, 111, 143, 144] and *Abstract Fuel Control* [73, 114]. The CPS is developed in Simulink [112], which in the industry is the de facto standard formalism for modeling control systems.

For each CPS, we have created two different AI-enabled CPS $\mathcal{M}^C$, by empowering them with a different DNN controller C . We employed the approach described in [143, 176] to train the controllers.

Table 7.1: AI-enabled CPSs $\mathcal{M}^C$ and correctness measure of its faulty versions $\mathcal{M}_f^C$ ($CM(\mathcal{M}_f^C, \varphi, TS)$ (%))

$\mathcal{M}^C$	ACC#1	ACC#2	AFC#1	AFC#2
Structure of C	[15 15 15]	[30 30 30]	[15 15 15]	[15 15 15 15]
#weights of C	450	1800	450	675
#blocks of $\mathcal{M}$	49	49	153	153
$CM(\mathcal{M}_f^C, \varphi, TS)$	20	24	52	35

Therefore, there is a total of four benchmarks, as shown in Table 7.1. We also report the complexity of the AI-enabled CPS in terms of: (i) number of Simulink blocks *#blocks* of $\mathcal{M}$, (ii) *structure* of C as the number of neurons in each layer, and (iii) number of weights *#weights* of C .

Adaptive Cruise Control (ACC). ACC [111], as introduced in Example 3, controls the acceleration of the ego car to keep a safe distance of it from a lead car. We consider the following specification, i.e., the relative distance d_{rel} between the two cars should always be greater than the sum of a constant d_{safe} and the braking distance $1.4 \cdot v_{ego}$ of the ego car during $[0, 50]$, and moreover, the speed v_{ego} of the ego car should be lower than 30.

$$\varphi_{ACC} \equiv \Box_{[0,50]}(d_{rel} \geq d_{safe} + 1.4 \cdot v_{ego} \wedge v_{ego} \leq 30)$$

Abstract Fuel Control (AFC). AFC is a powertrain control system developed by Toyota [73]. It takes two signals, pedal angle, and engine speed, as the external inputs and produces two output signals AF that indicate the *air-to-fuel* ratio and AF_{ref} that is the reference value of AF . The specification requires that AF should not deviate from AF_{ref} too much, during $[0, 30]$.

$$\varphi_{AFC} \equiv \Box_{[0,30]}(|AF - AF_{ref}| \leq 0.2 \cdot AF_{ref})$$

BUILDING FAULTY BENCHMARKS

In order to properly assess the capability of the compared approaches to repair a DNN controller, we produce faulty versions of the trained controllers. Specifically, for each AI-enabled CPS $\mathcal{M}^C$ and its corresponding specification φ , we proceed as follows.

We generate a test suite TS of 100 tests for $\mathcal{M}^C$ that uniformly cover the input space; for all the benchmarks, all the tests pass, i.e., $CM(\mathcal{M}^C, \varphi, TS) = 1$. Then, we randomly sample 10 weights of C , and we mutate their values, by randomly sampling in $[W_{min}, W_{max}]$, where W_{min} and W_{max} are the minimum and maximum weights values of C ; we identify the mutated weights as $\mathcal{W}^f = \{w_1^f, \dots, w_{10}^f\}$ and the modified AI-enabled CPS as $\mathcal{M}_f^C$. If the correctness measure of $\mathcal{M}_f^C$ is less than 0.6 (i.e., $CM(\mathcal{M}_f^C, \varphi, TS)$), we take $\mathcal{M}_f^C$ as the faulty benchmark to repair (i.e., $\mathcal{M}^{C_{orig}}$ in §7.3); otherwise, we sample other random values for the selected weights. Table 7.1 reports the obtained values of $CM(\mathcal{M}_f^C, \varphi, TS)$.

7.4.3 COMPARED APPROACHES

In the experiments, we assess the effectiveness of the proposed repair approaches CONTRREP and CONTRREP_{FAST}. As explained in §7.3.1, the approaches require an input of a set of weights SW to repair, usually obtained using some fault localization results.

Following the common practice of assessment of repair approaches in automated program repair [56], for each faulty benchmark $\mathcal{M}_f^C$, we built SW in a systematic way. We built three sets of SW , for assessing a repair approach in settings of different complexity:

- $SW_{noNoise}$: we assume that the fault localization approach precisely identified the faulty weights and not others, i.e., $SW_{noNoise} = \mathcal{W}^f$ (see §7.4.2). This setting allows to assess the capabilities of a repair approach, not influenced by possible deficiencies of the fault localization approach;
- SW_2 : we assume that the fault localization approach identified, in addition to the faulty weights, two additional weights that are not actually faulty, i.e., $SW_2 = \mathcal{W}^f \cup \{w_1, w_2\}$ with $w_1, w_2 \in$

$\mathcal{W} \setminus \mathcal{W}^f$. This setting allows to assess the capability of a repair approach to handle noise introduced by fault localization;

- SW_4 : this setting is similar to SW_2 , but four weights in the fault localization results are not faulty, i.e., $SW_4 = \mathcal{W}^f \cup \{w_1, w_2, w_3, w_4\}$ with $w_1, w_2, w_3, w_4 \in \mathcal{W} \setminus \mathcal{W}^f$.

Setting of the Search Algorithm. CONTRREP and CONTRREP_{FAST} use DE as an underlying search algorithm to perform the repair. We set the population size depending on the number of search variables, i.e., the number of suspicious weights in SW ; namely, we set the population size to $PopSize = 5 \cdot |SW|$. For all the experiments, we set the maximum number of generations as $NumGens = 50$. For DE, we used the implementation in PlatEMO [139]. We used the default setting because evidence from the literature shows that default settings can provide reasonable results [14].

7.4.4 EVALUATION METRICS AND ASSESSMENT

Running of the experiments. An *experiment* consists in the execution of one approach (CONTRREP or CONTRREP_{FAST}. See §7.4.3), over a specific faulty AI-enabled CPS (see §7.4.2), using a set of suspicious weights SW (see §7.4.3). So, in total, there are $2 \times 4 \times 3 = 24$ experiments. As search algorithms include an element of randomness, by following a guideline on running experiments with randomized algorithms [13], we executed each experiment 10 times.

To assess the repair effectiveness of a repair approach (RQ1 and RQ2), as an evaluation metric, we consider the value of the correctness measure obtained by the best-repaired model, i.e., $CM(\mathcal{M}^{C_{best}}, \varphi, TS)$. For CONTRREP_{FAST}, we report the correct value of CM (i.e., evaluated over all the tests in TS) and not the approximate value assessed by the heuristic method employed by the approach.

To assess the computational cost of a repair approach (RQ2), we consider the total number of test executions along the search *ExecTests*.

When comparing two approaches APP1 and APP2 over a given metric EM (i.e., *CM* or *ExecTests*), we compare the values obtained across the repetition of the experiments. Specifically, for each faulty benchmark $\mathcal{M}_f^C$ and set of suspicious weights *SW*:

- we compare the distributions of 10 values of EM obtained by APP1 and by APP2, using the Mann-Whitney U test, a non-parametric test that assesses whether there is a significant difference among the distributions. We use $\alpha = 0.05$ as the confidence value of the null hypothesis that there is no significant difference.
- In case of significant difference, we use Vargha and Delaney’s $\hat{A}_{12}$ effect size to assess the strength of the significance; if $\hat{A}_{12}$ is greater than 0.5, the results of APP1 are significantly higher than those of APP2. By following Kitchenham et al.’s classification [83], we identify the following categories of strength: *negligible* when $\hat{A}_{12} \in (0.5, 0.556)$, *small* when $\hat{A}_{12} \in [0.556, 0.638)$, *medium* when $\hat{A}_{12} \in [0.638, 0.714)$, and *large* when $\hat{A}_{12} \geq 0.714$. Similar categories can be identified for $\hat{A}_{12} < 0.5$, i.e., when APP1 is significantly lower.

When evaluating *CM*, values of $\hat{A}_{12}$ higher than 0.5 mean that APP1 is better. Instead, when evaluating *ExecTests*, values of $\hat{A}_{12}$ lower than 0.5 means that APP1 is better.

7.5 EXPERIMENT RESULTS

7.5.1 RQ1: EFFECTIVENESS OF CONTRREP

In this RQ, we want to assess if CONTRREP is actually able to repair the faulty DNN controllers. Table 7.2 reports, for each approach, the average *CM* across the 10 runs. From the results of Table 7.2, we observe that we are always able to improve the correctness measure *CM* in $\mathcal{M}^{C_{best}}$ w.r.t. the value of the faulty benchmark $\mathcal{M}_f^C$ (see Table 7.1). We notice that the effectiveness is lower for AFC, where the improvement of *CM* is lower. The physical plant in AFC has indeed complex dynamics that are difficult to control by the DNN controller [73]. For ACC, instead, the

Table 7.2: RQ1 and RQ2 – Average CM ($\mathcal{M}^{C_{best}}$, φ , TS) (%) of CONTRREP and CONTRREP_{FAST} across the 10 runs. Values of CM for CONTRREP_{FAST} report the exact value and not the approximate value computed by the approach

		CONTRREP	CONTRREP _{FAST}
ACC#1	$SW_{noNoise}$	80.6	77.3
	SW_2	83.8	77.9
	SW_4	91.5	81.5
ACC#2	$SW_{noNoise}$	80.5	83.3
	SW_2	81.4	79.7
	SW_4	73.2	73.0
AFC#1	$SW_{noNoise}$	71.6	71.3
	SW_2	78.0	77.1
	SW_4	82.6	66.6
AFC#2	$SW_{noNoise}$	72.9	50.7
	SW_2	43.8	43.0
	SW_4	59.6	57.4

control problem is easier. This difference in complexity is witnessed by the difference in # *blocks* of the controlled system $\mathcal{M}$ (see Table 7.1).

Regarding the performance under different sets of suspicious weights (i.e., $SW_{noNoise}$, SW_2 , SW_4), we observe two types of results. In ACC#2 and AFC#2, the values of CM decrease when using sets that contain weights that are not faulty (i.e., SW_2 and SW_4), compared to the case in which we use only the faulty weights (i.e., $SW_{noNoise}$). This is somehow expected, as, when using SW_2 and SW_4 , the search tries to modify some weights that are not faulty and this is not beneficial for the repair effectiveness: it can break tests passing with the original model, and it wastes search time in trying individuals that are not effective (i.e., those with large changes in the non-faulty weights).

However, in ACC#1 and AFC#1, the value of CM increases when using sets that contain weights that are not faulty (i.e., SW_2 and SW_4). Although this is surprising at first, it can be explained by the fact that, as explained in §7.4.3, the population size increases with the number of variables (that corresponds to the size of SW) and this allows to perform more exploration

Table 7.3: RQ2 – Comparison of CONTRREP_{FAST} and CONTRREP in terms of number of executed tests *ExecTests*. Legend: ($\equiv$: no sign. diff. between the two approaches. $\checkmark$: the approach on the left is *better* than the one on top, $\times$ means that it is worse; the num. of symbols is the strength: *negligible* ($\checkmark$, $\times$), *small* ($\checkmark\checkmark$, $\times\times$), *medium* ($\checkmark\checkmark\checkmark$, $\times\times\times$), *large* ($\checkmark\checkmark\checkmark\checkmark$, $\times\times\times\times$))

		CONTRREP _{FAST} vs.	<i>ExecTests</i> (avg. 10 runs)	
		CONTRREP	CONTRREP _{FAST}	CONTRREP
ACC#1	$SW_{noNoise}$	✓✓✓✓	7.0628e+04	2.5e+05
	SW_2	✓✓✓✓	7.4871e+04	3.0e+05
	SW_4	✓✓✓✓	9.9049e+04	3.5e+05
ACC#2	$SW_{noNoise}$	✓✓✓✓	3.9681e+04	2.5e+05
	SW_2	✓✓✓✓	4.8032e+04	3.0e+05
	SW_4	✓✓✓✓	6.2070e+04	3.5e+05
AFC#1	$SW_{noNoise}$	✓✓✓✓	1.0905e+05	2.5e+05
	SW_2	✓✓✓✓	1.2923e+05	3.0e+05
	SW_4	✓	1.7531e+05	3.07e+05
AFC#2	$SW_{noNoise}$	✓✓✓✓	8.4564e+04	2.5e+05
	SW_2	✓✓✓✓	1.0180e+05	3.0e+05
	SW_4	✓✓✓✓	1.2021e+05	3.5e+05

of the solution space. In these benchmarks, this higher exploration largely compensates for the noise introduced by trying to modify the non-faulty weights.

Answer to RQ1: CONTRREP is able to improve the correctness measure of controllers of AI-enabled CPSs. In some cases, the noise of fault localization results can negatively affect the performance of repair.

7.5.2 RQ2: USEFULNESS OF CONTRREP_{FAST}

In this RQ, we assess to what extent the heuristic applied by CONTRREP_{FAST} can speed up the search and which impacts the repair performance. Table 7.3 reports the statistical comparison between CONTRREP_{FAST} and CONTRREP in terms of executed tests *ExecTests*, and the average number of *ExecTests* across the 10 runs. We observe that CONTRREP_{FAST} is largely significantly better than CONTRREP in almost all the benchmarks and fault localization results (except for AFC#1 with SW_4). By looking at the average across the 10 runs of the number of executed tests *ExecTests*, we

Table 7.4: RQ2 – Comparison of CONTRREP and CONTRREP_{FAST} in terms of CM ($\mathcal{M}^{C_{best}}$, φ , TS) (legend as in Table 7.3)

CONTRREP vs. CONTRREP _{FAST}		
ACC#1	$SW_{noNoise}$	✓✓
	SW_2	✓✓✓✓
	SW_4	✓✓✓✓
ACC#2	$SW_{noNoise}$	✗✗
	SW_2	✓✓✓
	SW_4	✓✓
AFC#1	$SW_{noNoise}$	✗
	SW_2	≡
	SW_4	✓✓✓✓
AFC#2	$SW_{noNoise}$	✓✓✓✓
	SW_2	✓✓✓✓
	SW_4	✓✓

observe that CONTRREP_{FAST} reduces them by one order of magnitude in 7 out of 12 benchmarks, and, in the other 5 benchmarks, it reduces them to at least half. These results mean that when evaluating the fitness (see Alg. 5), the approach was often able to skip the evaluation of some tests, because some tests in TS_{orig}^- with high robustness was not repaired (so skipping all the other tests in TS_{orig}^- with lower robustness) and/or some test in TS_{orig}^+ with low robustness was not broken (so skipping all the other tests in TS_{orig}^+ with higher robustness).

Table 7.4 reports the statistical comparison in terms of CM between the approaches. By analyzing the results, we observe that usually CONTRREP_{FAST} is worst in terms of repair performance, confirmed by the results in Table 7.2. This is expected, as it is the price we need to pay to be faster. However, by comparing the values of CM in Table 7.2 for CONTRREP_{FAST} with the values of CM of the original model $\mathcal{M}_f^C$ (see Table 7.1), we observe that CONTRREP_{FAST} is always able to improve the controller.

From Table 7.4, we observe an unexpected result: CONTRREP_{FAST} is slightly better than CONTRREP in ACC#2 for $SW_{noNoise}$ (with strength *small*).¹ We investigated this case and we observed that,

¹A similar result occurs for AFC#1 and $SW_{noNoise}$, but with *negligible* strength.

in CONTRREP, the search finds good individuals that repair failing tests with very low robustness (i.e., the hard ones to repair), but not those with high robustness (i.e., that are close to being satisfied); however, these individuals are not further improved during the search to also repair the other failing tests. In CONTRREP_{FAST}, the fitness of such individuals is lower than the real CM: indeed, since we do not repair the easy tests, we assume that the hard tests are not repairable as well. CONTRREP_{FAST}, instead, favors the individuals that can repair the easiest tests first. Such individuals turn out to be better in the long term during the search. Thus, it could be that, for some benchmarks, it would be beneficial to prioritize the repair of the easy tests. We leave as future work the investigation of such prioritization during the search.

Answer to RQ2: Overall, CONTRREP_{FAST} allows us to significantly speed up the search, at the cost of a decrease in repair effectiveness. Still, CONTRREP_{FAST} can improve the controllers.

7.5.3 RQ3: EFFECTIVENESS OF CONTRREP IN FIXING FAILING TESTS WITHOUT BREAKING PASSING TESTS

In this RQ, we are interested in assessing how CONTRREP and CONTRREP_{FAST} improve the correctness measure, i.e., to what extent they repair the failing tests in TS_{orig}^- and preserve those in TS_{orig}^+ . Table 7.5 reports, for all the benchmarks and the two approaches, the percentages of tests in TS_{orig}^- that are *repaired* and *not repaired*, and the percentages of tests in TS_{orig}^+ that are *preserved* and *broken*. From the table, we observe that for ACC benchmarks, tests in TS_{orig}^+ that were passing with the original AI-enabled CPS $\mathcal{M}_f^C$ are almost never broken and still pass with the repaired system $\mathcal{M}^{C_{best}}$. In AFC benchmarks, instead, some tests in TS_{orig}^+ are broken, up to 36%.

Depending on the application domain, this could be a problem, as some stakeholders may prefer not to break previously correct behaviours to avoid regression. Note that, in our approaches, we are only interested in maximizing the number of passing tests in $\mathcal{M}^{C_{best}}$, so we do not take

Table 7.5: RQ3 – Number of repaired and broken tests (average across 10 runs)

		CONTRREP				CONTRREP _{FAST}			
		TS_{orig}^-		TS_{orig}^+		TS_{orig}^-		TS_{orig}^+	
		repaired	not repaired	preserved	broken	repaired	not repaired	preserved	broken
ACC#1	$SW_{noNoise}$	75.89%	24.11%	99.29%	0.71%	71.61%	28.39%	100.00%	0.00%
	SW_2	79.64%	20.36%	100.00%	0.00%	72.32%	27.68%	100.00%	0.00%
	SW_4	89.38%	10.63%	100.00%	0.00%	76.88%	23.13%	100.00%	0.00%
ACC#2	$SW_{noNoise}$	74.34%	25.66%	100.00%	0.00%	78.03%	21.97%	100.00%	0.00%
	SW_2	75.79%	24.21%	100.00%	0.00%	73.29%	26.71%	100.00%	0.00%
	SW_4	65.13%	34.87%	100.00%	0.00%	64.80%	35.20%	100.00%	0.00%
AFC#1	$SW_{noNoise}$	47.02%	52.98%	94.23%	5.77%	43.75%	56.25%	96.70%	3.30%
	SW_2	62.50%	37.50%	92.31%	7.69%	64.84%	35.16%	88.46%	11.54%
	SW_4	67.50%	32.50%	97.69%	2.31%	48.75%	51.25%	79.62%	20.38%
AFC#2	$SW_{noNoise}$	66.84%	33.16%	84.13%	15.87%	29.23%	70.77%	90.48%	9.52%
	SW_2	32.31%	67.69%	65.14%	34.86%	18.62%	81.38%	88.29%	11.71%
	SW_4	36.92%	63.08%	64.00%	36.00%	29.85%	70.15%	85.71%	14.29%

any measure to avoid breaking tests in TS_{orig}^+ , and we accept it if this allows us to increase the number of repaired tests in TS_{orig}^- . We will consider as future work a version of the approach that aims at repairing, without breaking previously correct tests.

Regarding the number of repaired tests, we can never repair all the tests, with lower performance in AFC#2 that, as seen in RQ1, is the benchmark in which we obtain the worst results.

Answer to RQ3: Depending on the benchmark, the percentage of tests in TS_{orig}^+ that are broken varies, with higher break rates for the most difficult benchmarks. Similarly, the percentage of tests in TS_{orig}^- that are repaired depends on the difficulty of the benchmark.

7.6 THREATS TO VALIDITY

The validity of CONTRREP could be affected by different threats [151].

Construct validity. A threat of this type could be that the metrics we use for assessing the compared approaches (i.e., CONTRREP and CONTRREP_{FAST}) are not suitable. In our context, the goal of repair is to improve the performance of the AI-enabled CPS over the test suite TS , which is measured by CM (see Eq. 7.1); therefore, using CM as assessment metric is correct. CONTRREP_{FAST} has been introduced to speed up the search by avoiding executing all the tests whose execution is computationally expensive; therefore, using the number of executed tests $ExecTests$ to compare CONTRREP and CONTRREP_{FAST} is suitable.

Conclusion validity. CONTRREP and CONTRREP_{FAST} adopt the search algorithm DE to perform the repair. The randomness of DE can affect the results. Therefore, to account for the non-deterministic nature of the repair approaches, we repeat each experiment 10 times as suggested by guidelines on running experiments with randomized algorithms [13].² Moreover, we have also used suitable statistical tests to compare the approach by considering both significant differences and effect size.

Internal validity. A threat of this type is that the obtained results are by chance, for example, due to an *instrumentation* threat [151] (e.g., faulty implementation). To mitigate this threat, we have carefully tested the implementation of CONTRREP and CONTRREP_{FAST}.

External validity. It could be that the performance of CONTRREP does not generalise to other AI-enabled CPS and DNN controllers. To mitigate this threat, we have considered two industrial-scale CPSs that are widely used in competitions on testing CPSs [114] and DNN controllers [109].

7.7 CONCLUSION

AI-enabled CPSs use DNNs as controllers of the physical plant. Despite their advantages, DNN controllers can be faulty, leading to wrong control decisions. We propose CONTRREP, a

²Although 30 runs are suggested, it is also recognized in [13] that this may be not possible when using computationally expensive systems as in our approach. In this case, particular attention must be paid to the statistical assessment of the results.

search-based repair approach for DNN controllers, guided by system-level specifications. We further introduce $\text{CONTRREP}_{\text{FAST}}$, a modification of the approach that tries to speed up the search. Experiments show that CONTRREP is indeed able to repair faulty DNN controllers, and that $\text{CONTRREP}_{\text{FAST}}$ can greatly reduce the computational complexity at a small price in terms of repair effectiveness.

In this work, we assume that the plant is correct and that the DNN controller must be fixed. However, some plants have hyperparameters that can be tuned, which can affect the behaviour of the AI-enabled CPS. In future work, we plan to investigate the combined repair of the physical plant and the DNN controller.

8 | CONCLUSION

In this chapter, we conclude this dissertation and discuss some promising future research directions.

8.1 CONTRIBUTION

This dissertation explores the entire lifecycle of quality assurance of AI-enabled CPSs (i.e., benchmarking, falsification, fault localisation and repair), with the following conclusions and findings:

- We present a public benchmark of AI-enabled CPS in various domains, which can serve as a fundamental evaluation and testing framework for enhancing the understanding and development of AI-enabled CPS. Based on this benchmark, we collected a series of evaluation metrics and measured the performance and reliability of state-of-the-art deep reinforcement learning (DRL) controllers on various types of CPS. Our findings reveal some strengths and weaknesses of AI-enabled CPS and highlights an opportunity of strategically combining AI-enabled CPS with traditional CPS. Furthermore, our analysis of two widely used falsification techniques on AI-enabled CPS motivates further improvement of these techniques to account for the unique characteristics of AI controllers, in order to build safe and reliable CPSs in the age of AI.
- We proposed a coverage-guided falsification framework FalsifAI, which exhaustively em-

employs eight time-aware coverage criteria as guidance to explore the temporal behaviors of AI-enabled CPS. These coverage criteria aim to capture different time-series features of DNN controller and provide better guidance to find violation cases to the system specification. The large-scale evaluation demonstrates the effectiveness of FalsifAI and our proposed coverage criteria. In the future, we will extend this work to other types of neural network controllers and design different coverage criteria for falsification.

- we propose the framework SpectAcle, that localises the DNN controller weights responsible for the system specification violation, by exploiting the sequence of DNN controller inferences. Our technical novelty consists in the construction of the execution spectrum, in which we take the system-level specification as the correctness criterion for the DNN controller, and use forward impact to calculate the relevance of each neuron weight to the DNN controller inference. Our experimental results show the effectiveness of SpectAcle, based on the comparison with two baseline approaches.
- AI-enabled CPSs use DNNs as controllers of the physical plant. Despite their advantages, DNN controllers can be faulty, leading to wrong control decisions. We propose ContrRep, a search-based repair approach for DNN controllers, guided by system-level specifications. We further introduce ContrRepfast, a modification of the approach that tries to speed up the search. Experiments show that ContrRep is indeed able to repair faulty DNN controllers, and that ContrRepfast can greatly reduce the computational complexity at a small price in terms of repair effectiveness.

8.2 FUTURE WORK

We now discuss some promising future directions based on this dissertation.

- *Falsification.* In our previous falsification work, we only consider feed-forward neuron networks, We also consider extending the framework to other types of neural networks, e.g.,

recurrent neural networks (RNN). Unlike the NN used in this work which is stateless, RNN has a mechanism of storing the internal states and use them during a sequence of inferences. Therefore, in the case of RNN, we can consider defining their time-aware coverage criteria w.r.t. the internal states and use those criteria for test case generation.

- *Fault localisation.* As a future work, we plan to devise other methods for computing forward impact, because this is the core factor that affects the effectiveness of the approach. For example, in this work, we consider the activation of a neuron as a possible cause of a wrong DNN control action. However, it could be that the non-activation of a neuron is responsible for a wrong action. As future work, we plan to devise a fault localisation approach in which also non-activation is considered as a possible cause of misbehaviour.
- *Repair.* In the aforementioned repair work, we assume that the plant is correct and that the DNN controller must be fixed. However, some plants have hyperparameters that can be tuned, which can affect the behaviour of the AI-enabled CPS. In future work, we plan to investigate the combined repair of the physical plant and the DNN controller.

BIBLIOGRAPHY

- [1] International Organization for Standardization. *ISO/PAS 21448: Road vehicles - Safety of the intended functionality*. Norm. 2019.
- [2] Rui Abreu, Peter Zoetewij, and Arjan JC Van Gemund. “An evaluation of similarity coefficients for software fault localization”. In: *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC’06)*. IEEE, 2006, pp. 39–46. DOI: 10.1109/PRDC.2006.18.
- [3] Rui Abreu et al. “A practical evaluation of spectrum-based fault localization”. In: *Journal of Systems and Software* 82.11 (2009), pp. 1780–1792. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2009.06.035>.
- [4] Arvind Adimoolam et al. “Classification and coverage-based falsification for embedded control systems”. In: *CAV*. Springer. 2017, pp. 483–503.
- [5] J. Allen. “Short term spectral analysis, synthesis, and modification by discrete Fourier transform”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 25.3 (1977), pp. 235–238. DOI: 10.1109/TASSP.1977.1162950.
- [6] Mohammed Alshiekh et al. “Safe reinforcement learning via shielding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018.
- [7] Rajeev Alur and David L. Dill. “A theory of timed automata”. In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235. ISSN: 0304-3975. DOI: 10.1016/0304-3975(94)90010-8.

-
- [8] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. 2nd. USA: Cambridge University Press, 2016. ISBN: 1107172012.
 - [9] Greg Anderson et al. “Neurosymbolic reinforcement learning with formally verified exploration”. In: *Advances in neural information processing systems* 33 (2020), pp. 6172–6183.
 - [10] Yashwanth Annpureddy et al. “S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–257. ISBN: 978-3-642-19835-9.
 - [11] Yashwanth Annpureddy et al. “S-taliro: A tool for temporal logic falsification for hybrid systems”. In: *TACAS*. Springer. 2011, pp. 254–257.
 - [12] Andrea Arcuri and Lionel Briand. “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. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 1–10. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985795.
 - [13] Andrea Arcuri and Lionel Briand. “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. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 1–10. ISBN: 9781450304450. DOI: 10.1145/1985793.1985795.
 - [14] Andrea Arcuri and Gordon Fraser. “Parameter tuning or default values? An empirical investigation in search-based software engineering”. In: *Empirical Software Engineering* 18 (2013), pp. 594–623. DOI: 10.1007/s10664-013-9249-9.
 - [15] Kai Arulkumaran et al. “Deep reinforcement learning: A brief survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

- [16] Anand Balakrishnan and Jyotirmoy V Deshmukh. “Structured reward shaping using signal temporal logic specifications”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 3481–3486.
- [17] Ezio Bartocci et al. “Automatic Failure Explanation in CPS Models”. In: *Software Engineering and Formal Methods*. Ed. by Peter Csaba Ölveczky and Gwen Salaün. Cham: Springer International Publishing, 2019, pp. 69–86. ISBN: 978-3-030-30446-1.
- [18] Ezio Bartocci et al. “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. Porto, Portugal: Association for Computing Machinery, 2018, pp. 197–206. ISBN: 9781450356428. DOI: 10.1145/3178126.3178131.
- [19] Ezio Bartocci et al. “Search-based Testing for Accurate Fault Localization in CPS”. In: *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. 2022, pp. 145–156. DOI: 10.1109/ISSRE55969.2022.00024.
- [20] Raja Ben Abdesslem et al. “Automated repair of feature interaction failures in automated driving systems”. In: *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 88–100. ISBN: 9781450380089. DOI: 10.1145/3395363.3397386.
- [21] David Berend et al. “Cats Are Not Fish: Deep Learning Testing Calls for out-of-Distribution Awareness”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’20. Virtual Event, Australia: ACM, 2020, pp. 1041–1052. ISBN: 9781450367684. DOI: 10.1145/3324884.3416609.
- [22] Felix Berkenkamp et al. “Safe model-based reinforcement learning with stability guarantees”. In: *Advances in neural information processing systems* 30 (2017).
- [23] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016).

-
- [24] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016).
- [25] Davide Li Calsi et al. “Distributed Repair of Deep Neural Networks”. In: *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE. Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 83–94.
- [26] Jialun Cao et al. “DeepFD: Automated Fault Diagnosis and Localization for Deep Learning Programs”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 573–585. ISBN: 9781450392211. DOI: 10.1145/3510003.3510099.
- [27] Yiqun T. Chen et al. “Revisiting the Relationship between Fault Detection, Test Adequacy Criteria, and Test Set Size”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’20. Virtual Event, Australia: Association for Computing Machinery, 2020, pp. 237–249. ISBN: 9781450367684. DOI: 10.1145/3324884.3416667.
- [28] John J. Chilenski and Steven P. Miller. “Applicability of modified condition/decision coverage to software testing”. In: *Software Engineering Journal* 9.5 (Sept. 1994), pp. 193–200. ISSN: 0268-6961.
- [29] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. New York: Academic Press, 1969.
- [30] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. “Exploration and Exploitation in Evolutionary Algorithms: A Survey”. In: *ACM Comput. Surv.* 45.3 (July 2013). ISSN: 0360-0300. DOI: 10.1145/2480741.2480752.
- [31] Jyotirmoy V. Deshmukh et al. “Learning Deep Neural Network Controllers for Dynamical Systems with Safety Guarantees: Invited Paper”. In: *2019 IEEE/ACM International Confer-*

- ence on Computer-Aided Design (ICCAD)*. 2019, pp. 1–7. DOI: 10.1109/ICCAD45719.2019.8942130.
- [32] Adel Dokhanchi et al. “Requirements driven falsification with coverage metrics”. In: *2015 International Conference on Embedded Software (EMSOFT)*. 2015, pp. 31–40. DOI: 10.1109/EMSOFT.2015.7318257.
- [33] Alexandre Donzé. “Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems”. In: *Proceedings of the 22nd International Conference on Computer Aided Verification. CAV’10*. Edinburgh, UK: Springer-Verlag, 2010, pp. 167–170. ISBN: 364214294X. DOI: 10.1007/978-3-642-14295-6_17.
- [34] Alexandre Donzé. “Breach, a toolbox for verification and parameter synthesis of hybrid systems”. In: *CAV*. Springer. 2010, pp. 167–170.
- [35] Alexandre Donzé and Oded Maler. “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*. Klosterneuburg, Austria: Springer-Verlag, 2010, pp. 92–106. ISBN: 3642152961.
- [36] Alexandre Donzé and Oded Maler. “Robust satisfaction of temporal logic over real-valued signals”. In: *FORMATS*. Springer. 2010, pp. 92–106.
- [37] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. “Compositional falsification of cyber-physical systems with machine learning components”. In: *JAR* 63.4 (2019), pp. 1031–1053.
- [38] Tommaso Dreossi et al. “Efficient Guiding Strategies for Testing of Temporal Properties of Hybrid Systems”. In: *NASA Formal Methods*. Cham: Springer International Publishing, 2015, pp. 127–142. ISBN: 978-3-319-17524-9.

- [39] Tommaso Dreossi et al. “VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems”. In: *Computer Aided Verification*. Ed. by Isil Dillig and Serdar Tasiran. Cham: Springer International Publishing, 2019, pp. 432–442. ISBN: 978-3-030-25540-4.
- [40] Xiaoning Du et al. “DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 477–487. ISBN: 9781450355728. DOI: 10.1145/3338906.3338954.
- [41] Xiaoning Du et al. “Marble: Model-Based Robustness Analysis of Stateful Deep Learning Systems”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’20. Virtual Event, Australia: Association for Computing Machinery, 2020, pp. 423–435. ISBN: 9781450367684. DOI: 10.1145/3324884.3416564.
- [42] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control”. In: *ICML*. PMLR. 2016, pp. 1329–1338.
- [43] Matias Duran et al. “What to Blame? On the Granularity of Fault Localization for Deep Neural Networks”. In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 2021, pp. 264–275. DOI: 10.1109/ISSRE52982.2021.00037.
- [44] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. “DeepFault: Fault Localization for Deep Neural Networks”. In: *Fundamental Approaches to Software Engineering*. Ed. by Reiner Hähnle and Wil van der Aalst. Cham: Springer International Publishing, 2019, pp. 171–191. ISBN: 978-3-030-16722-6.
- [45] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. “DeepFault: Fault Localization for Deep Neural Networks”. In: *Fundamental Approaches to Software Engineering*. Ed. by Reiner

- Hähnle and Wil van der Aalst. Cham: Springer International Publishing, 2019, pp. 171–191. ISBN: 978-3-030-16722-6.
- [46] Gidon Ernst et al. “ARCH-COMP 2020 Category Report: Falsification”. In: *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*. Ed. by Goran Frehse and Matthias Althoff. Vol. 74. EPiC Series in Computing. EasyChair, 2020, pp. 140–152. DOI: 10.29007/trr1.
- [47] Gidon Ernst et al. “Arch-comp 2020 category report: Falsification”. In: *EPiC Series in Computing* (2020).
- [48] Gidon Ernst et al. “ARCH-COMP 2021 Category Report: Falsification with Validation of Results”. In: *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*. Ed. by Goran Frehse and Matthias Althoff. Vol. 80. EPiC Series in Computing. EasyChair, 2021, pp. 133–152. DOI: 10.29007/xw11.
- [49] Gidon Ernst et al. “ARCH-COMP 2022 Category Report: Falsification with Unbounded Resources”. In: *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*. Ed. by Goran Frehse et al. Vol. 90. EPiC Series in Computing. EasyChair, 2022, pp. 204–221. DOI: 10.29007/fhmk.
- [50] Hazem Fahmy et al. “Supporting Deep Neural Network Safety Analysis and Retraining Through Heatmap-Based Unsupervised Learning”. In: *IEEE Transactions on Reliability* (2021), pp. 1–17. DOI: 10.1109/TR.2021.3074750.
- [51] Georgios E. Fainekos and George J. Pappas. “Robustness of Temporal Logic Specifications for Continuous-Time Signals”. In: *Theor. Comput. Sci.* 410.42 (Sept. 2009), pp. 4262–4291. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2009.06.021.
- [52] James Ferlez, Xiaowu Sun, and Yasser Shoukry. “Two-Level Lattice Neural Network Architectures for Control of Nonlinear Systems”. In: *2020 59th IEEE Conference on Decision*

- and Control (CDC)*. Jeju Island, Korea (South): IEEE Press, 2020, pp. 2198–2203. DOI: 10.1109/CDC42340.2020.9304079.
- [53] Gordon Fraser and Andrea Arcuri. “Whole Test Suite Generation”. In: *IEEE Transactions on Software Engineering* 39.2 (2013), pp. 276–291. DOI: 10.1109/TSE.2012.14.
- [54] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *ICML*. PMLR, 2018, pp. 1587–1596.
- [55] Xiang Gao et al. “Fuzz Testing Based Data Augmentation to Improve Robustness of Deep Neural Networks”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE ’20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 1147–1158. ISBN: 9781450371216. DOI: 10.1145/3377811.3380415.
- [56] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. “Automatic Software Repair: A Survey”. In: *IEEE Trans. Softw. Eng.* 45.1 (Jan. 2019), pp. 34–67. ISSN: 0098-5589. DOI: 10.1109/TSE.2017.2755013.
- [57] Ali Ghanbari et al. “Mutation-based Fault Localization of Deep Neural Networks”. In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2023, pp. 1301–1313. DOI: 10.1109/ASE56229.2023.00171.
- [58] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. SIAM, 2019.
- [59] Aditya Gilra and Wulfram Gerstner. “Non-linear motor control by local learning in spiking neural networks”. In: *International Conference on Machine Learning*. PMLR, 2018, pp. 1773–1782.
- [60] Christoph Gladisch et al. “Experience Paper: Search-Based Testing in Automated Driving Control Applications”. In: *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’19. San Diego, California: IEEE Press, 2019, pp. 26–37. ISBN: 9781728125084. DOI: 10.1109/ASE.2019.00013.

- [61] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and harnessing adversarial examples*. 2014.
- [62] Shixiang Gu et al. “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE Press, 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.
- [63] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *ICML*. 2018, pp. 1861–1870.
- [64] Joseph Y. Halpern. *Actual Causality*. The MIT Press, 2016. ISBN: 0262035022.
- [65] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)”. In: *Evolutionary computation* 11.1 (2003), pp. 1–18.
- [66] Thomas A Henzinger. “The theory of hybrid automata”. In: *Verification of digital and hybrid systems*. Springer, 2000, pp. 265–292.
- [67] Michael Hertneck et al. “Learning an approximate model predictive controller with guarantees”. In: *IEEE Control Systems Letters* 2.3 (2018), pp. 543–548.
- [68] Eric M. Hope, Xichen Jiang, and Alejandro D. Dominguez-Garcia. “A Reachability-Based Method for Large-Signal Behavior Verification of DC-DC Converters”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 58.12 (2011), pp. 2944–2955. DOI: 10.1109/TCSI.2011.2157780.
- [69] Chao Huang et al. “Reachnn: Reachability analysis of neural-network controlled systems”. In: *TECS* 18.5s (2019), pp. 1–22.
- [70] Xiaowei Huang et al. “Safety verification of deep neural networks”. In: *CAV*. Springer. 2017, pp. 3–29.

- [71] Nargiz Humbatova et al. “Taxonomy of Real Faults in Deep Learning Systems”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE ’20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 1110–1121. ISBN: 9781450371216. DOI: 10.1145/3377811.3380395.
- [72] International Organization for Standardization. *ISO 26262:Road vehicles - Functional safety*. Norm. 2011.
- [73] Xiaoqing Jin et al. “Powertrain control verification benchmark”. In: *Proceedings of the 17th international conference on Hybrid systems: computation and control*. 2014, pp. 253–262.
- [74] Taylor T Johnson et al. “ARCH-COMP20 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants”. In: *EPiC Series in Computing* 74 (2020).
- [75] Taylor T. Johnson et al. “ARCH-COMP21 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants”. In: *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*. Ed. by Goran Frehse and Matthias Althoff. Vol. 80. EPiC Series in Computing. EasyChair, 2021, pp. 90–119. DOI: 10.29007/kfk9.
- [76] James A. Jones, Mary Jean Harrold, and John Stasko. “Visualization of Test Information to Assist Fault Localization”. In: *Proceedings of the 24th International Conference on Software Engineering*. ICSE ’02. Orlando, Florida: Association for Computing Machinery, 2002, pp. 467–477. ISBN: 158113472X. DOI: 10.1145/581339.581397.
- [77] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. “Deep Neural Network Compression for Aircraft Collision Avoidance Systems”. In: *CoRR* abs/1810.04240 (2018).
- [78] Chijung Jung et al. “Swarmbug: debugging configuration bugs in swarm robotics”. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. Athens, Greece:

- Association for Computing Machinery, 2021, pp. 868–880. ISBN: 9781450385626. DOI: 10.1145/3468264.3468601.
- [79] James Kapinski et al. “Simulation-Based Approaches for Verification of Embedded Control Systems: An Overview of Traditional and Advanced Modeling, Testing, and Verification Techniques”. In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 45–64. DOI: 10.1109/MCS.2016.2602089.
- [80] Tanmay Khandait et al. “ARCH-COMP 2024 Category Report: Falsification”. In: *Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*. Ed. by Goran Frehse and Matthias Althoff. Vol. 103. EPiC Series in Computing. EasyChair, 2024, pp. 122–144. DOI: 10.29007/hgfv.
- [81] Jinhan Kim, Robert Feldt, and Shin Yoo. “Guiding Deep Learning System Testing Using Surprise Adequacy”. In: *Proceedings of the 41st International Conference on Software Engineering*. ICSE ’19. Montreal, Quebec, Canada: IEEE Press, 2019, pp. 1039–1049. DOI: 10.1109/ICSE.2019.00108.
- [82] Jinhan Kim et al. “Repairing DNN Architecture: Are We There Yet?” In: *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2023, pp. 234–245. DOI: 10.1109/ICST57152.2023.00030.
- [83] Barbara Kitchenham et al. “Robust statistical methods for empirical software engineering”. In: *Empirical Software Engineering* 22 (2017), pp. 579–630.
- [84] George Klees et al. “Evaluating Fuzz Testing”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 2123–2138. ISBN: 9781450356930. DOI: 10.1145/3243734.3243804.

- [85] Tien-Duy B. Le, Ferdian Thung, and David Lo. “Theory and Practice, Do They Match? A Case with Spectrum-Based Fault Localization”. In: *Proceedings of the 2013 IEEE International Conference on Software Maintenance*. ICSM ’13. USA: IEEE Computer Society, 2013, pp. 380–383. ISBN: 9780769549811. DOI: 10.1109/ICSM.2013.52.
- [86] Claire Le Goues et al. “Automatic Program Repair”. In: *IEEE Software* 38.4 (2021), pp. 22–27. DOI: 10.1109/MS.2021.3072577.
- [87] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.
- [88] Davide Li Calsi et al. “Adaptive Search-Based Repair of Deep Neural Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1527–1536. ISBN: 9798400701191. DOI: 10.1145/3583131.3590477.
- [89] Davide Li Calsi et al. “Adaptive Search-Based Repair of Deep Neural Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1527–1536. ISBN: 9798400701191. DOI: 10.1145/3583131.3590477.
- [90] Davide Li Calsi et al. “Distributed Repair of Deep Neural Networks”. In: *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2023, pp. 83–94. DOI: 10.1109/ICST57152.2023.00017.
- [91] Davide Li Calsi et al. “Federated Repair of Deep Neural Networks”. In: *2024 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. 2024.
- [92] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

- [93] Tsang-Wei Lin, Sheue-Ling Hwang, and Paul A. Green. “Effects of time-gap settings of adaptive cruise control (ACC) on driving performance and subjective acceptance in a bus driving simulator”. In: *Safety Science* 47.5 (2009), pp. 620–625. ISSN: 0925-7535. DOI: <https://doi.org/10.1016/j.ssci.2008.08.004>.
- [94] Bing Liu et al. “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)*. 2017, pp. 359–370. DOI: 10.1109/SANER.2017.7884636.
- [95] Bing Liu et al. “Localizing Multiple Faults in Simulink Models”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. 2016, pp. 146–156. DOI: 10.1109/SANER.2016.38.
- [96] Bing Liu et al. “Simulink fault localization: an iterative statistical debugging approach”. In: *Software Testing, Verification and Reliability* 26.6 (2016), pp. 431–459.
- [97] Wenliang Liu, Noushin Mehdipour, and Calin Belta. “Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints”. In: *IEEE Control Systems Letters* 6 (2021), pp. 91–96.
- [98] Marco A Luersen and Rodolphe Le Riche. “Globalized Nelder–Mead method for engineering optimization”. In: *Computers & structures* 82.23-26 (2004), pp. 2251–2260.
- [99] Chen Lv et al. “Levenberg–Marquardt Backpropagation Training of Multilayer Neural Networks for State Estimation of a Safety-Critical Cyber-Physical System”. In: *IEEE Transactions on Industrial Informatics* 14.8 (2018), pp. 3436–3446. DOI: 10.1109/TII.2017.2777460.
- [100] Deyun Lyu et al. *Code and benchmarks of the paper “SPECTACLE: Fault Localisation of AI-enabled CPS by Exploiting Sequences of DNN Controller Inferences”*. 2024.

- [101] Deyun Lyu et al. “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. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1435–1444. ISBN: 9798400704949. DOI: 10.1145/3638529.3654078.
- [102] Deyun Lyu et al. *Supplementary Material for the paper “Search-Based Repair of DNN Controllers of AI-Enabled Cyber-Physical Systems Guided by System-Level Specifications”*. <https://github.com/lyudeyun/ContrRep>. 2024.
- [103] Deyun Lyu et al. *Supplementary material for the paper “SPECTACLE: Fault Localisation of AI-enabled CPS by Exploiting Sequences of DNN Controller Inferences”*. 2024.
- [104] Lei Ma et al. “DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems”. In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2019, pp. 614–618. DOI: 10.1109/SANER.2019.8668044.
- [105] Lei Ma et al. “DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE 2018. Montpellier, France: Association for Computing Machinery, 2018, pp. 120–131. ISBN: 9781450359375. DOI: 10.1145/3238147.3238202.
- [106] Shiqing Ma et al. “MODE: automated neural network model debugging via state differential analysis and input selection”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 175–186.
- [107] Keyvan Majd et al. “Safe Robot Learning in Assistive Devices through Neural Network Repair”. In: *6th Annual Conference on Robot Learning*. 2022.

- [108] Diego Manzananas Lopez et al. “ARCH-COMP22 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants”. In: *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*. Ed. by Goran Frehse et al. Vol. 90. EPiC Series in Computing. EasyChair, 2022, pp. 142–184. DOI: 10.29007/wfgr.
- [109] Diego Manzananas Lopez et al. “ARCH-COMP23 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants”. In: *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*. Ed. by Goran Frehse and Matthias Althoff. Vol. 96. EPiC Series in Computing. EasyChair, 2023, pp. 89–125. DOI: 10.29007/x38n.
- [110] Logan Mathesen et al. “Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart”. In: *CASE*. IEEE. 2019, pp. 991–997.
- [111] Mathworks. *Adaptive Cruise Control System Using Model Predictive Control*. 2024.
- [112] Mathworks. *Simulink*. 2024.
- [113] Claudio Menghi et al. “Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE ’20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 372–384. ISBN: 9781450371216. DOI: 10.1145/3377811.3380370.
- [114] Claudio Menghi et al. “ARCH-COMP23 Category Report: Falsification”. In: *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*. Ed. by Goran Frehse and Matthias Althoff. Vol. 96. EPiC Series in Computing. EasyChair, 2023, pp. 151–169. DOI: 10.29007/6nqs.

- [115] Martin Foddslette Møller. “A scaled conjugate gradient algorithm for fast supervised learning”. In: *Neural networks* 6.4 (1993), pp. 525–533.
- [116] Martin Monperrus. “Automatic software repair: a bibliography”. In: *ACM Computing Surveys (CSUR)* 51.1 (2018), pp. 1–24.
- [117] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. “A Model for Spectra-Based Software Diagnosis”. In: *ACM Trans. Softw. Eng. Methodol.* 20.3 (Aug. 2011). ISSN: 1049-331X. DOI: 10.1145/2000791.2000795.
- [118] Shiva Nejati et al. “Evaluating Model Testing and Model Checking for Finding Requirements Violations in Simulink Models”. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 1015–1025. ISBN: 9781450355728. DOI: 10.1145/3338906.3340444.
- [119] Shiva Nejati et al. “Evaluating model testing and model checking for finding requirements violations in Simulink models”. In: *ESEC/FSE*. 2019, pp. 1015–1025.
- [120] Luan Viet Nguyen and Taylor T. Johnson. “Benchmark: DC-to-DC Switched-Mode Power Converters (Buck Converters, Boost Converters, and Buck-Boost Converters)”. In: *ARCH14-15. 1st and 2nd International Workshop on Applied verification for Continuous and Hybrid Systems*. Ed. by Goran Frehse and Matthias Althoff. Vol. 34. EPiC Series in Computing. EasyChair, 2015, pp. 19–24. DOI: 10.29007/23pm.
- [121] Scott A Nivison and Pramod P Khargonekar. “Development of a robust deep recurrent neural network controller for flight applications”. In: *ACC*. IEEE. 2017, pp. 5336–5342.
- [122] Scott A. Nivison and Pramod P. Khargonekar. “Development of a Robust, Sparsely-Activated, and Deep Recurrent Neural Network Controller for Flight Applications”. In: *2018 IEEE Conference on Decision and Control (CDC)*. FL, USA: IEEE Press, 2018, pp. 384–390. DOI: 10.1109/CDC.2018.8619188.

- [123] Augustus Odena et al. “TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing”. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 4901–4911.
- [124] Carlos Pacheco and Michael D. Ernst. “Randoop: Feedback-Directed Random Testing for Java”. In: *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion*. OOPSLA ’07. Montreal, Quebec, Canada: Association for Computing Machinery, 2007, pp. 815–816. ISBN: 9781595938657. DOI: 10.1145/1297846.1297902.
- [125] Spencer Pearson et al. “Evaluating and improving fault localization”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 609–620.
- [126] Kexin Pei et al. “DeepXplore: Automated Whitebox Testing of Deep Learning Systems”. In: *Commun. ACM* 62.11 (Oct. 2019), pp. 137–145. ISSN: 0001-0782. DOI: 10.1145/3361566.
- [127] André Platzer and Jan-David Quesel. “KeYmaera: A hybrid theorem prover for hybrid systems (system description)”. In: *International Joint Conference on Automated Reasoning*. Springer, 2008, pp. 171–178.
- [128] Hua Qi et al. “ArchRepair: Block-Level Architecture-Oriented Repairing for Deep Neural Networks”. In: *ACM Trans. Softw. Eng. Methodol.* 32.5 (July 2023). ISSN: 1049-331X. DOI: 10.1145/3585005.
- [129] Xuhong Ren et al. “DARTSRepair: Core-failure-set guided DARTS for network robustness to common corruptions”. In: *Pattern Recognition* 131 (2022), p. 108864.
- [130] Vincenzo Riccio et al. “Testing machine learning based systems: a systematic mapping”. In: *Empir. Softw. Eng.* 25.6 (2020), pp. 5193–5254. DOI: 10.1007/s10664-020-09881-0.
- [131] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

-
- [132] Ali Shafahi et al. “Adversarial training for free!” In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2019, pp. 3358–3369.
- [133] Jeongju Sohn, Sungmin Kang, and Shin Yoo. “Arachne: Search-based repair of deep neural networks”. In: *ACM Transactions on Software Engineering and Methodology* 32.4 (2023), pp. 1–26.
- [134] Jiayang Song et al. “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*. ICSE-SEIP ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 343–352. ISBN: 9781450392266. DOI: 10.1145/3510457.3513049.
- [135] Rainer Storn and Kenneth Price. “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. In: *J. of Global Optimization* 11.4 (Dec. 1997), pp. 341–359. ISSN: 0925-5001. DOI: 10.1023/A:1008202821328.
- [136] Kohei Suenaga and Ichiro Hasuo. “Programming with infinitesimals: A while-language for hybrid system modeling”. In: *International Colloquium on Automata, Languages, and Programming*. Springer, 2011, pp. 392–403.
- [137] Bing Sun et al. “Causality-Based Neural Network Repair”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 338–349. ISBN: 9781450392211. DOI: 10.1145/3510003.3510080.
- [138] Youcheng Sun et al. “Concolic Testing for Deep Neural Networks”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 109–119. ISBN: 9781450359375.

- [139] Ye Tian et al. “PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization”. In: *Comp. Intell. Mag.* 12.4 (Nov. 2017), pp. 73–87. ISSN: 1556-603X. DOI: 10.1109/MCI.2017.2742868.
- [140] Yuchi Tian et al. “DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars”. In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 303–314. ISBN: 9781450356381. DOI: 10.1145/3180155.3180220.
- [141] Shogo Tokui et al. “NeuRecover: Regression-Controlled Repair of Deep Neural Networks with Training History”. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2022, pp. 1111–1121. DOI: 10.1109/SANER53432.2022.00128.
- [142] Shogo Tokui et al. “NeuRecover: Regression-Controlled Repair of Deep Neural Networks with Training History”. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2022, pp. 1111–1121. DOI: 10.1109/SANER53432.2022.00128.
- [143] Hoang-Dung Tran et al. “NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems”. In: *CAV*. Springer. 2020, pp. 3–17.
- [144] Hoang-Dung Tran et al. “Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control”. In: *ACM Trans. Embed. Comput. Syst.* 18.5s (Oct. 2019). ISSN: 1539-9087. DOI: 10.1145/3358230.
- [145] Hoang-Dung Tran et al. “Star-based reachability analysis of deep neural networks”. In: *FM*. Springer. 2019, pp. 670–686.
- [146] Pablo Valle, Aitor Arrieta, and Maite Arratibel. “Automated Misconfiguration Repair of Configurable Cyber-Physical Systems with Search: An Industrial Case Study on Elevator Dispatching Algorithms”. In: *Proceedings of the 45th International Conference on Software*

- Engineering: Software Engineering in Practice*. ICSE-SEIP '23. Melbourne, Australia: IEEE Press, 2023, pp. 396–408. ISBN: 9798350300376. DOI: 10.1109/ICSE-SEIP58684.2023.00042.
- [147] Peter JM Van Laarhoven and Emile HL Aarts. “Simulated annealing”. In: *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.
- [148] Hao Wang, Berk Ustun, and Flavio Calmon. “Repairing without retraining: Avoiding disparate impact with counterfactual distributions”. In: *International Conference on Machine Learning*. PMLR, 2019, pp. 6618–6627.
- [149] Mohammad Wardat, Wei Le, and Hriday Rajan. “DeepLocalize: Fault Localization for Deep Neural Networks”. In: *Proceedings of the 43rd International Conference on Software Engineering*. ICSE '21. Madrid, Spain: IEEE Press, 2021, pp. 251–262. ISBN: 9781450390859. DOI: 10.1109/ICSE43902.2021.00034.
- [150] Mohammad Wardat et al. “DeepDiagnosis: automatically diagnosing faults and recommending actionable fixes in deep learning programs”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 561–572. ISBN: 9781450392211. DOI: 10.1145/3510003.3510071.
- [151] Claes Wohlin et al. *Experimentation in software engineering*. Vol. 236. Springer, 2012.
- [152] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *TEC 1.1* (1997), pp. 67–82.
- [153] Eric Wong and Zico Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *International Conference on Machine Learning*. PMLR, 2018, pp. 5286–5295.
- [154] W Eric Wong et al. “The DStar method for effective software fault localization”. In: *IEEE Transactions on Reliability* 63.1 (2013), pp. 290–308.

- [155] W. Eric Wong et al. “A Survey on Software Fault Localization”. In: *IEEE Trans. Softw. Eng.* 42.8 (Aug. 2016), pp. 707–740. ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2521368.
- [156] W. Eric Wong et al. “Software Fault Localization Using DStar (D*)”. In: *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability. SERE '12*. USA: IEEE Computer Society, 2012, pp. 21–30. ISBN: 9780769547428. DOI: 10.1109/SERE.2012.12.
- [157] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. “Output reachable set estimation and verification for multilayer neural networks”. In: *TNNLS* 29.11 (2018), pp. 5777–5783.
- [158] Xiaofei Xie et al. “DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks”. In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2019*. Beijing, China: Association for Computing Machinery, 2019, pp. 146–157. ISBN: 9781450362245. DOI: 10.1145/3293882.3330579.
- [159] Shakiba Yaghoubi and Georgios Fainekos. “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*. Montreal, Quebec, Canada: Association for Computing Machinery, 2019, pp. 179–184. ISBN: 9781450362825. DOI: 10.1145/3302504.3311814.
- [160] Shakiba Yaghoubi, Georgios Fainekos, and Sriram Sankaranarayanan. “Training Neural Network Controllers Using Control Barrier Functions in the Presence of Disturbances”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. Rhodes: IEEE Press, 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294485.
- [161] Yoriyuki Yamagata et al. “Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning”. In: *IEEE Transactions on Software Engineering* 47.12 (2021), pp. 2823–2840. DOI: 10.1109/TSE.2020.2969178.

- [162] Xiaodong Yang et al. “Neural network repair with reachability analysis”. In: *Formal Modeling and Analysis of Timed Systems: 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13–15, 2022, Proceedings*. Springer, 2022, pp. 221–236.
- [163] Bing Yu et al. “DeepRepair: Style-Guided Repairing for Deep Neural Networks in the Real-World Operational Environment”. In: *IEEE Transactions on Reliability* (2021), pp. 1–16. DOI: 10.1109/TR.2021.3096332.
- [164] Bing Yu et al. “DeepRepair: Style-Guided Repairing for Deep Neural Networks in the Real-World Operational Environment”. In: *IEEE Transactions on Reliability* (2021), pp. 1–16. DOI: 10.1109/TR.2021.3096332.
- [165] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer, 2014, pp. 818–833.
- [166] Hao Zhang and W. K. Chan. “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. San Diego, California: IEEE Press, 2019, pp. 376–387. ISBN: 9781728125084. DOI: 10.1109/ASE.2019.00043.
- [167] Hao Zhang and W.K. Chan. “Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models”. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019, pp. 376–387. DOI: 10.1109/ASE.2019.00043.
- [168] Jie M. Zhang et al. “Machine Learning Testing: Survey, Landscapes and Horizons”. In: *IEEE Transactions on Software Engineering* 48.1 (2022), pp. 1–36. DOI: 10.1109/TSE.2019.2962027.
- [169] Jie M. Zhang et al. “Machine Learning Testing: Survey, Landscapes and Horizons”. In: *IEEE Trans. Softw. Eng.* 48.1 (Jan. 2022), pp. 1–36. ISSN: 0098-5589. DOI: 10.1109/TSE.2019.2962027.

- [170] Xiao-Yi Zhang and Mingyue Jiang. “SPICA: A Methodology for Reviewing and Analysing Fault Localisation Techniques”. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 366–377. DOI: 10.1109/ICSME52107.2021.00039.
- [171] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. “Constraining Counterexamples in Hybrid System Falsification: Penalty-Based Approaches”. In: *NASA Formal Methods*. Ed. by Ritchie Lee, Susmit Jha, and Anastasia Mavridou. Cham: Springer International Publishing, 2020, pp. 401–419. ISBN: 978-3-030-55754-6. DOI: 10.1007/978-3-030-55754-6_24.
- [172] Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. “Hybrid System Falsification under (In)equality Constraints via Search Space Transformation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3674–3685. DOI: 10.1109/TCAD.2020.3013073.
- [173] Zhenya Zhang, Ichiro Hasuo, and Paolo Arcaini. “Multi-armed Bandits for Boolean Connectives in Hybrid System Falsification”. In: *Computer Aided Verification*. Cham: Springer International Publishing, 2019, pp. 401–420. ISBN: 978-3-030-25540-4. DOI: 10.1007/978-3-030-25540-4_23.
- [174] Zhenya Zhang et al. “Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided by QB-Robustness”. In: *CAV*. Springer. 2021, pp. 595–618.
- [175] Zhenya Zhang et al. *FalsifAI: Falsification of AI-Enabled Hybrid Control Systems Guided by Time-Aware Coverage Criteria*. Apr. 2022. DOI: 10.5281/zenodo.6464099.
- [176] Zhenya Zhang et al. “FalsifAI: Falsification of AI-Enabled Hybrid Control Systems Guided by Time-Aware Coverage Criteria”. In: *IEEE Transactions on Software Engineering* 49.4 (2023), pp. 1842–1859. DOI: 10.1109/TSE.2022.3194640.

- [177] Zhenya Zhang et al. “Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (Nov. 2018), pp. 2894–2905. ISSN: 0278-0070. DOI: 10.1109/TCAD.2018.2858463.
- [178] Zhenya Zhang et al. “Two-layered falsification of hybrid systems guided by monte carlo tree search”. In: *TCAD* 37.11 (2018), pp. 2894–2905.
- [179] Weichao Zhou et al. “Runtime-safety-guided policy repair”. In: *International Conference on Runtime Verification*. Springer. 2020, pp. 131–150.
- [180] He Zhu et al. “An inductive synthesis framework for verifiable reinforcement learning”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019, pp. 686–701.

PUBLISHED PAPERS

- [1] **Deyun Lyu**, Zhenya Zhang, Paolo Arcaini, Xiao-Yi Zhang, Fuyuki Ishikawa and Jianjun Zhao. SpectAcle: Fault Localisation of AI-Enabled CPS by Exploiting Sequences of DNN Controller Inferences. In *ACM Transactions on Software Engineering and Methodology*, to appear. (TOSEM 2024)
- [2] **Deyun Lyu**, Zhenya Zhang, Paolo Arcaini, Fuyuki Ishikawa, Thomas Laurent and Jianjun Zhao. 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*, pp. 1435-1444, July 2024. (GECCO 2024)
- [3] Zhenya Zhang, **Deyun Lyu**, Paolo Arcaini, Lei Ma, Ichiro Hasuo, Jianjun Zhao. FalsifAI: Falsification Of AI-Enabled Hybrid Control Systems Guided By Time-Aware Coverage Criteria. In *IEEE Transactions on Software Engineering*, 49(4): 1842-1859, July 2022. (TSE 2022)
- [4] Jiayang Song*, **Deyun Lyu***, Zhenya Zhang, Zhijie Wang, Tianyi Zhang, Lei Ma. When Cyber-Physical Systems Meet AI: A Benchmark, an Evaluation, and a Way Forward. In *44th International Conference on Software Engineering: Software Engineering in Practice*, pp. 343-352, May 2022. (ICSE-SEIP 2022, Co-first author)
- [5] Zhenya Zhang, **Deyun Lyu**, Paolo Arcaini, Lei Ma, Ichiro Hasuo and Jianjun Zhao. Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided By QB-Robustness. In

33rd International Conference on Computer-Aided Verification, pp. 595–618, July 2021. (CAV 2021)

- [6] Zhenya Zhang, **Deyun Lyu**, Paolo Arcaini, Lei Ma, Ichiro Hasuo and Jianjun Zhao. On the Effectiveness of Signal Rescaling in Hybrid System Falsification. In *13th NASA Formal Methods Symposium*, pp. 392-399, May 2021. (NFM 2021)