Attacks detection in Cyber-Physical Systems with Neural Networks: a case study

Cinzia Bernardeschi*, Gianluca Dini†, Maurizio Palmieri‡ and Alessio Vivani§
Department of Information Engineering, Unversity of Pisa
Email: *cinzia.bernardeschi@unipi.it, †gianluca.dini@unipi.it, ‡maurizio.palmieri@unipi.it, §alessio.vivani@gmail.com

***Abstract*—Cyber-Physical Systems (CPSs) are a large class of systems characterized by networked co-operating sub-systems, that perceive surrounding environment via sensors and actuators. Cybersecurity is relevant in CPSs because, on the one hand, these systems expose a wide cyber-attack surface while, on the other hand, a security infringement may translate into a safety infringement. This work presents a methodology for developing an intrusion detection system for CPSs based on neural networks. The methodology exploits a digital twin of the CPS to generate traces of executions. An instrumented approach is used to extend the digital twin model by introducing functions that simulate the effects of various class of attacks on the system. The instrumented digital twin is used to gather data of the system's behaviour with and without attacks. Collected data are used for training the neural network. To illustrate the methodology we consider a case-study featuring an Adaptive Cruise Control System in autonomously driving vehicles. A Multi-Layer Perceptron neural network is trained to detect attacks to sensors. Results show an accuracy of 94% in detecting attacks.**

***Index Terms*—Cyber-Physical Systems, Cybersecurity, Neural Networks, Autonomous Driving**

I. Introduction

Cyber-physical Systems (CPSs) are increasingly widespread and they are growing in complexity. Examples of CPSs are Smart Grids, Transportation Systems, 4.0 Industries and Energy Systems. They are an integration of sub-systems possibly designed by different vendors and provide a smooth interaction between hardware and software components since they have cooperating computational, networking and physical processes.

A digital twin is a virtual replica of physical systems, created and maintained to gather insights about its physical counterpart [1]. A digital twin can be used for simulation with possibly many different scenarios in order to gain an insight into how the real system behaves, as well as for monitoring, diagnostics, anomaly detection and predictive maintenance [2], [3], [4].

This study received funding from the European Union–Next-GenerationEU - National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTIMENT N. 1.1, National Sustainable Mobility Center CN00000023, Italian Ministry of University and Research Decree n.1033—17/06/2022, Spoke 10, CALL PRIN 2022 D.D. 104 02-02-2022 – FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems, CUP N.I53D23006130001, the project "FuSeCar" funded by the MUR Progetti di Ricerca di Rilevante Interesse Nazionale (PRIN) Bando 2022 - grant 2022W3EPEP - CUP: E53D23008210006, and from the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project and of the CrossLab project (Departments of Excellence).

Cybersecurity is a really important topic for CPSs [5], because, on the one hand, these systems expose a wide cyber-attack surface while, on the other hand, a security breach may translate into a safety issue. Attacks on CPSs could be on sensors, actuators or controllers, or the network.

Common approaches for detecting attacks on a system point towards the usage of Intrusion Detection Systems (IDSs) that monitor the CPS for anomalies in network traffic or system behaviours [6] [7], [8]. Many IDSs use machine learning algorithms, since they learn from data and can recognize patterns that are hard to identify by humans. For example, in [9], different machine learning algorithms are applied for the creation of IDSs for network security. With respect to other solutions, machine learning based IDSs are shown to provide better ability to detect new types of attacks and to reduce the amount of false positives.

Machine learning is a very large field that aims at creating systems that are able to learn and increase performances based on the data that are given to it. Within machine learning resides Neural Networks (NN), which are a particular type of machine learning systems that aims at replicating the human neuron's work.

In this work we present a general methodology for training a neural network to detect attacks in a CPS based on a digital twin. The methodology starts by analysing the effect of different attack scenarios in the system, followed by the collection of the system's traces of execution in absence/presence of attacks and the training of the neural network. The digital twin is instrumented by extending the model of the CPS by introducing functions that simulate the effects of various classes of attacks on the system. Data collected are labeled to distinguish scenarios with and without attacks. In order to give more flexibility during the modelling and simulation of CPSs, co-simulation technique is applied [10]. We use the traces collected in presence and absence of attacks to train a NN based IDS.

The developed methodology is applied to a case study featuring an Adaptive Cruise Control System in autonomously driving vehicles [11]. In autonomous driving, cars are continuously connected [12]. While safety has always been a strict requirement, recently *cybersecurity* is getting more and more interest, as attested by the recent release of the ISO/SAE 21434 standard [13].

The use case consists of two autonomous driving vehicles always trying to keep a safe distance between them. The study

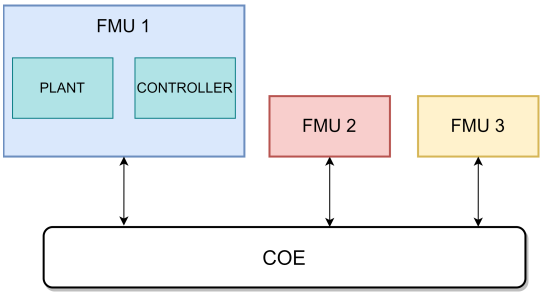

Fig. 1. Co-Simulation Architecture Example

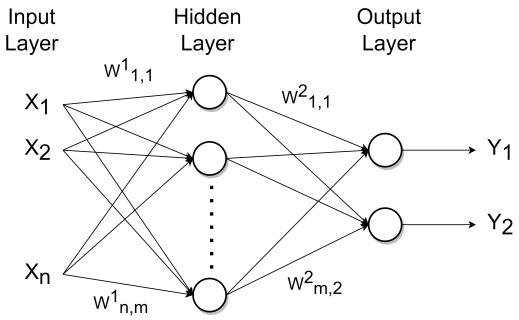

Fig. 2. Multi-Layer Perceptron example

focuses on attacks made on sensors, which, by modifying the sensed values sent to the controller, cause problems in the decision-making process of the vehicle and may lead to a collision.

The paper is organised as follows: Section II reports basic concepts on co-simulation and neural networks; Section III describes the methodology; Section IV provides an insight on a case study used to apply the methodology; Section V shows, and analyzes, the results obtained; finally, Section VI contains conclusions and further work.

II. Background

This section briefly recalls basic concepts on co-simulation and design space exploration, and on neural networks.

A. Co-simulation

In the field of CPSs, simulation often takes the form of co-simulation [10]. Co-simulation allows different system elements to be modelled by different languages and simulated each by a dedicated simulator, with a co-simulation master orchestrating their execution. The term multi-model is used to indicate the model of the whole system.

The Functional Mockup Interface (FMI) [14] is a public-domain standard that defines a container and an interface to exchange each simulation model as a component, called Functional Mockup Unit (FMU). Fig. 1 shows an example of FMI-compliant co-simulation architecture, where the COE module is the Co-simulation Orchestrator Engine. In particular, each FMU is a zip file that contains the methods responsible for the model evolution, together with the model's parameters (input and output variables), a simulator and the implementation of the API of the FMI standard, used by the orchestrator to interface itself with each simulator. In our work, the co-simulation toolbox used is the INtegrated TOol chain for model-based design of CPSs (INTO-CPS) application [15].

The Design Space Exploration tool (DSE) [16] is a tool provided by INTO-CPS that allow us to automatically run simulations over a set of scenarios in order to find the best combination of parameters according to an optimization function defined by the user.

When defining a DSE configuration, the following elements must be specified: the FMUs to use and how they should be connected; a set of values for each parameter of every FMU used; the search algorithm, which is usually the exhaustive search, that simply computes the list of scenarios by doing a Cartesian product between all the set of values per parameter. Other information can be provided such as a custom optimization function and constraints on the parameter combinations.

All the aforementioned parameters are saved into a JSON file, automatically created by the INTO-CPS application. When a space exploration is executed, a simulation per each parameter combination is run and, for each simulation, a csv file containing the time series related to the system's state evolution is stored.

B. Neural Network: Multi-Layer Perceptron

Multi-Layer Perceptrons (MLPs) [17] are a specific type of neural networks which base their decision making process on a matrix of values, i.e. *weights*, that gets updated during the so-called training phase, depending on the data provided, in order to fit a specific task. An MLP is composed of many neurons, called perceptrons, organized in layers, where each neuron in one layer is connected to any neuron in the adjacent layers, as it is shown in Fig. 2, where X_i is the *i-th* input feature, Y_j is the *j-th* output, $W^h_{l,k}$ is the weight of the *h-th* layer that connects neuron l with neuron k, n is the number of input features and m is the number of neurons in the hidden layer. Those types of networks take as input a vector containing the features to analyze and, through the use of the weight matrix and the activation functions of the neurons, compute the output vector. The training phase requires a dataset with a set of vectors of features, where each vector is called *entry*, and for each entry, a *label* is required, which is the desired output vector that the network has to learn. The training process can be summarised as follows:

- The network takes one or more input entries from the training dataset and tries to predict the correct output.
- The predicted value is compared against the desired one, computing the error.
- The error is then back-propagated into the network, changing the weights, to reduce the error.

III. Methodology

This section describes the proposed methodology, which consists of four different steps. The description will not be case-specific, thus making it a general approach that can be applied to any CPS. The steps are the following:

- Extension of the baseline digital twin by introducing attack scenarios, i.e. attack injection;
- Data gathering through simulations in presence and absence of attacks;
- Training dataset creation, filtering wrong data samples and labeling;
- Neural network training.

The methodology was developed to work in a suggested environment, composed of INTO-CPS [15] for co-simulation and MATLAB [18] for neural networks. The former was chosen mainly because it provides the design space exploration tool. MATLAB instead was suggested because of the toolboxes for the creation of neural networks (deep learning toolbox), and because it allows to export models according to the FMI standard. However, the methodology is not limited to the suggested environment, e.g. it is possible to use Python instead of MATLAB for the neural network, thanks to the UniFMU toolbox [19] which allows exporting models in various languages into an FMU. The following subsections provide details on each step.

A. Attack Injection

To develop a neural network for attack detection, the first step should be to add the possibility to model the effects of attacks on the system model [20].

To model attacks there are two main approaches:

- Extend a preexisting FMU introducing a function that alters the behaviour of the sub-system when some conditions are met, for example, an attack can start after a given number of simulation steps and it can last a given number of steps;
- Create a new Attack FMU, that will be inserted as a man in the middle between two other FMUs, which contains one or more functions to implement the attacks.

The methodology supports both approaches, but it is important to note that the first solution can't be used if the target FMU is a black box and doesn't allow access or modify the code. The second solution on the other hand is always possible but it will slightly modify the co-simulation architecture of the system.

The implementation of many different attacks results in several additional parameters for the CPS. In particular, periodic or non-periodic attacks should be modelled. Moreover, the shape of the attack function must vary, e.g. in case of data alteration, step, ramp or sinusoidal functions should be considered. Finally, the attack function should have some parameters to precisely describe it, i.e. in the case of a step function, it is required the size of the step.

B. Data Gathering

The data gathering process consists of the automatic simulations of different combinations of the parameters of the CPS. This can be achieved with the DSE tool provided by INTO-CPS. It is possible to exploit its functionalities to automatically execute simulations and collect time series related to the system's state evolution with and without attacks. The DSE JSON configuration file, available for the selected multi-model, can be accessed and modified, thus allowing us to create a script to inject new data inside the set of values for each parameter of the multi-model. Moreover, it is possible to add random values, thus helping with the exploration of the design space. It is important to note that the final goal of this step is to obtain data that will be used for training a neural network. For this reason, having a large dataset is preferred.

Moreover, introducing randomicity in the attacks and in the overall system will help the network to generalize, rather than letting it memorize only a smaller portion of the whole sample space.

C. Training Dataset creation

A well designed training dataset is crucial for the creation of a neural network. A dataset is composed of a set of entries, which are an array of information that depends on the type of networks that we want to train. In order to define the task to be learned by the classifier, each entry must be labeled, since we work with a supervised training algorithm. It's important to make sure that each different label has approximately the same number of entries, in order to have a balanced dataset. In conclusion, each entry should provide meaningful information. To ensure this, correlated entries should be avoided. Moreover, during the data gathering process it is possible that during the simulation of an attack scenario, the system eventually ends up in an inconsistent state, e.g., if two vehicles crash, the behaviour of the system after the crash is not useful to detect the crash and therefore should not be included in the dataset.

D. Neural Network training

After the generation of the dataset, depending on the type of neural network selected, some pre-processing might be required. Furthermore, the training phase consists of finding the optimal values for the hyperparameters of the network (e.g., variables that determine the network structure) to obtain the best performances, e.g. the number of neurons or hidden layers for a Multi-Layer Perceptron. Algorithms that help with this task are the Grid Search and Genetic Algorithm [21].

Of course, at the end of the training phase, the output will be the best neural network among the ones that have been tested, but it doesn't necessarily mean that it will have acceptable performances, e.g. the accuracy might be below an acceptability threshold. If that is the case, the algorithm should be run again, changing the pool of hyperparameters to test or changing the training dataset by adding new data.

IV. Case study

In order to test the efficiency and validity of the methodology described in the previous section, it was applied to a case study. The CPS under analysis was obtained from a MATLAB project called Adaptive Cruise Control System Using Model Predictive Control [11]. Sensor reading attacks are taken into account.

The system consists of two vehicles, one called Ego Car and the other Lead Car. The Ego Car aims at following the

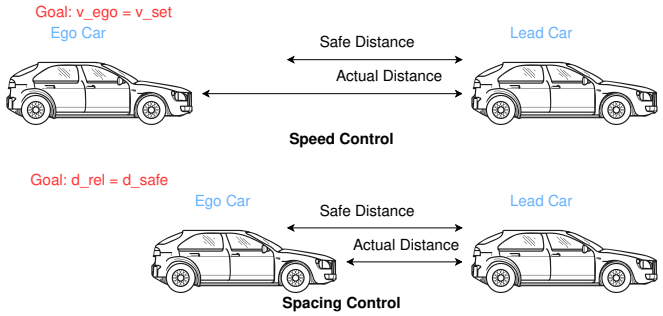

Fig. 3. Operational Modes of the Baseline Model

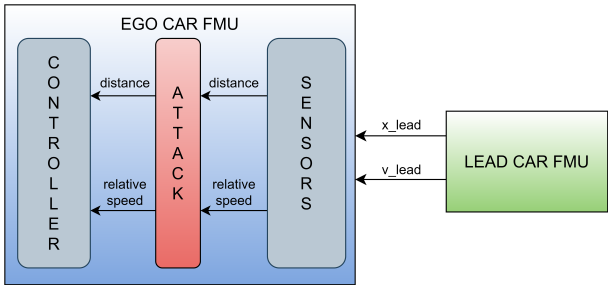

Fig. 4. Schema for the injection of an attack

Lead Car. The Ego Car has two different operational modes depending on the actual distance between the two vehicles, as it can be seen in Fig. 3.

Before analyzing the algorithm we should present the *safe distance* (d_{safe}):

$$d_{safe} = t_{gap} \times v_{ego} + d_{base}$$

where t_{gap} is parameter that refers to the reaction time, v_{ego} is the actual velocity of the Ego Car and d_{base} is a base value of the safe distance.

The two operational modes are called *spacing control* and *speed control*. The vehicle goes on spacing control when the actual distance is smaller than the computed safe distance, making it slow down in order to increase the distance between the two. Instead, it goes on speed control when the distance is greater than the safe distance, thus making the following car's speed increase up to a higher bound, which is a system's parameter called v_{set}.

Finally, the leader moves regardless of what is happening at the Ego Car, and in the case under study its acceleration is modelled with a *sine* function. We chose to use the sine function since it makes sufficiently complex the task of following the leader. The Lead Car has four parameters dedicated to the *sine* function that models the acceleration, which are the *frequency*, *phase*, *offset* and *amplitude*, and they can be modified to study different behavioural scenarios. Another reason why the sine function was selected is the fact that, by changing the parameters and the simulation time, it is possible to approximate many other different behavioral acceleration functions for the leader. The formula for the acceleration is the following:

$$acc = amplitude \times \sin(frequency \times t + phase) + offset$$

where t is the simulation time.

The models of both the Lead car and the Ego car are in Simulink and we exported them as FMUs.

Morever, the controller of the Ego Car is a black box, thus it was not possible to modify it. Furthermore, its FMU had both Plant and Controller inside (see FMU1 in Fig. 1).

A. Attack Injection

The selected attacks are sensor reading attacks. The position and velocitiy of the Lead Car measured by sensors on the the Ego Car are altered. These data are input to the controller of the car.

Since the Ego Car's FMU is composed of both Plant and Controller, we chose to modify its model by introducing a MATLAB function between sensor's output and controller's input, as it is shown in Fig. 4).

In order to easily configure the attack scenarios it is required to add a few parameters to the baseline model, i.e. give the possibility to define the type of the data alteration and the starting time of the attack.

Three different attacks were implemented and each one of those could have been constant or periodic, i.e. the attack starts and never stops or it periodically changes between active and inactive state.

As an example, the first type of attack will be presented here, which is a *step function alteration*. When the attack starts, the attack function will provide data to the controller which are obtained from a sensor incremented by a value that is defined at configuration time. For example, the vehicle might see the leader 5 meters further away than it actually is, thus possibly causing the follower to stay in speed control rather than going in spacing control. This can be dangerous as it reduces the safe distance between the two cars. When the attack starts, the attack function simply modifies the sensed data using an offset defined at configuration time. In Fig. 5 it is shown the evolution of the distance perceived by the Ego Car during a simulation. The attack starts after 20 simulated seconds and it increases the value obtained by the sensors by 5 meters. In red it is shown the distance perceived by the Ego Car. In green is the actual distance between Ego Car and Lead Car. Before time 20, both are equal but after time 20, when the attack starts, the value differ for 5 meters

B. Data Gathering and Neural Network Creation

Since we decided to use a Multi-Layer Perceptron for the classification task, we have to keep in mind that such a network requires as input a set of features. Those can be collected by applying functions, such as the max and min function, to a *subset* of the time-series of csv file. Each subset is then called **window** and the size of the window defines both the amount of information stored and the responsiveness of the neural network, since the latter will only send an output once per window. In our case study, the window size was chosen

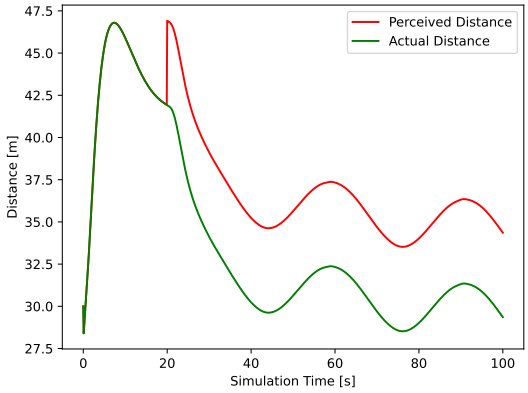

Fig. 5. Distance between the two cars in case of the attack example

to be equal to 20 entries of the time-series, e.g. 2 seconds of simulation time.

In order to properly explore the design space, different batches of simulations were run. Each one of those was referring to only one attack type, being in total four, one batch for each type of attack and one batch without attacks. Then, adjusting the parameters accordingly, it is possible to balance the number of time windows with attack and the number of time windows without attacks. More precisely it is sufficient to set the starting time of the attacks to exactly one-third of the simulation time of the co-simulation run. This way, given the fact that a co-simulation run lasted for 100 seconds of simulated time and each window was 2 seconds long, each simulation would provide a total of 50 windows. Considering that each batch is composed of 3000 runs, the total number of "under attack" windows is $50 \cdot \frac{2}{3} \times 3000 \times 3 = 300000$, which will be the same for the "no attack" windows ($50 \cdot \frac{1}{3} \times 3000 \times 3 + 50 \times 3000 = 300000$).

This is, of course, an approximation, since some time-series were removed because they were referring to inconsistent states. The whole data gathering process took around 5 days to fully complete.

Moreover, in order to correctly save the data that we need for the training phase it was necessary to introduce in the architecture a new FMU, named Dummy FMU, since the DSE only saves into the csv file, data related to output variables that are connected to input variables of another FMU. The Dummy FMU will have many input ports, that will be connected to variables that need to be monitored. In order to monitor the evolution of an internal variable of an FMU, such variable must be connected to an output one, which will be sent as input to the Dummy FMU. The co-simulation schema related to the data gathering process is provided in Fig. 6, and it can be generalized to other CPS.

After the dataset was finally obtained, the Multi-Layer Perceptron (MLP) training phase started.

This type of network requires, as input, a set of features. The latter should be computed on the time series within a specific time window. Usually, when dealing with the feature selection problem, a method provided by MATLAB called **sequentialfs**

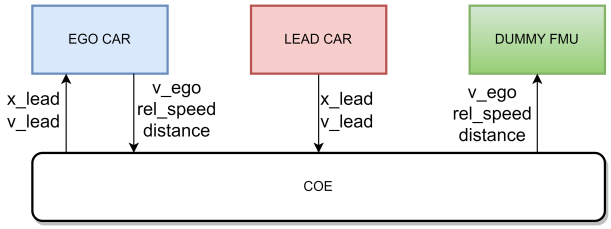

Fig. 6. Co-Simulation schema for data gathering

[22] should be used.

The following features are used: *minimum*, *maximum*, *mean*, *standard deviation*, *skewness* and *harmonic mean*, applied to the velocity of the Ego Car, the distance perceived and the relative velocity perceived.

After a series of tests with different hyperparameters, we found the best structure to be a one hidden layer with 25 neurons.

V. Results

To evaluate the network's performance for our classification problem, the confusion matrix is used. The labels, and so the predictions of our network, are only two, which are "Under Attack" and "No Attack".

The confusion matrix will be a 2 by 2 matrix where the rows refer to the prediction of the network, i.e. the first row contains the "No Attack" outputs, the second one instead has the "Under Attack" ones. The columns refer to the desired output, which the network aims at replicating. Each cell of the matrix *(i, j)* contains a number, which is the number of entries of the dataset of label *j* that were labeled as *i* by the network.

Given this definition, it's easy to say that the cells on the diagonal contain the correct predictions of the network. Each cell has a meaning:

- **top-left**: counts the correct "no attack" predictions, i.g. *true negatives*;
- **top-right**: counts the number of entries labelled as "no attack" but were actually "under attack", called *false negatives*;
- **bottom-left**: counts the number of entries labelled as "under attack" but actually weren't, called *false positives*;
- **bottom-right**: counts the correct "under attack" predictions, i.g. *true positives*.

In Fig. 7 it is shown the confusion matrix obtained after the training phase of the optimal network, having 25 neurons in the only hidden layer.

Moreover, after obtaining the results shown, in order to verify its validity, we started a new data gathering phase, with a total of 3000 simulations of new scenarios never seen by the network, balancing the entries between "no attack" and "under attack". The accuracy obtained during this test phase was of the same order of magnitude of the one shown in Fig. 7, thus validating the previous results.

As it can be seen in the matrix, the percentage of false negatives that we obtained was significantly higher than the

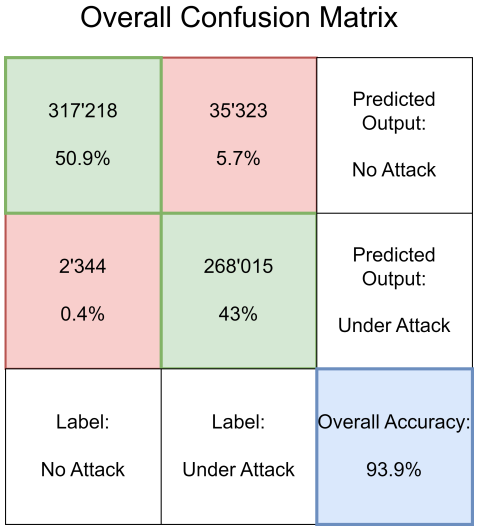

Fig. 7. Testing Accuracy

false positives one. For this reason, a deeper analysis was done to identify its cause. Launching a series of simulations of the system under attack, it is possible to notice that, since the Ego Car's controller is a Model Predictive Controller (MPC), if the attack has a low intensity, i.e. a very low offset value for the step function attack, and is kept constant for the rest of the simulation, eventually the MPC will adapt to the change and the attack will have virtually no effect.

For this reason, after some simulation time the classifier will not be able to detect the presence of the attack, but it will almost always detect it as soon as it starts, which is a good result, since it can be used to implement a mitigation mechanism to avoid hazardous events on the system.

VI. Conclusions

This paper proposes a methodology to train and test neural networks for the detection of attacks in CPSs, based on a data set obtained by simulating the model of the system. The paper also provides an application example of such methodology on a case study inspired by the MATLAB project on autonomous vehicles. The methodology provides a precise but yet very flexible workflow that could be applied to different scenarios with little effort. Moreover, the results obtained show the potential of such an approach.

When dealing with neural networks and machine learning in general, having a large dataset may be the difference between a well trained network and another. Simulation based approaches allow to easily control the size and content of the training data, making it a very powerful solution.

Further work includes the following activities:

- Introducing strategies and algorithms to optimize the data gathering process, which is one of the most time and memory consuming step in the methodology;
- Adding a general solution to export the classifier into an FMU, to allow co-simulating the whole system together with the intrusion detection system;
- To further assess the methodology's validity, apply it to a more complex and realistic case study.

References

[1] J. Fitzgerald, P. G. Larsen, and K. Pierce, *Multi-modelling and Co-simulation in the Engineering of Cyber-Physical Systems: Towards the Digital Twin*. Cham: Springer International Publishing, 2019, pp. 40–55.

[2] D. Botín-Sanabria, A.-S. Mihaita, R. Peimbert-García, M. Ramírez-Moreno, R. Ramírez-Mendoza, and J. Lozoya-Santos, "Digital Twin Technology Challenges and Applications: A Comprehensive Review." *Remote Sens.*, vol. 14, p. 1335, 2022.

[3] M. Palmieri, C. Quadri, A. Fagiolini, and C. Bernardeschi, "Co-simulated digital twin on the network edge: A vehicle platoon," *Computer Communications*, p. 35–47, 2024.

[4] Q. Xu, S. Ali, and T. Yue, "Digital twin-based anomaly detection in cyber-physical systems," in *14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2021, pp. 205–216.

[5] T. Alladi, V. Chamola, and S. Zeadally, "Industrial control systems: cyberattack trends and countermeasures." *Comput. Commun.*, 2020.

[6] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber–physical systems." *ACM Comput. Surv.*, vol. 46, no. 4, 2014.

[7] H. Kayan, M. Nunes, O. Rana, P. Burnap, and C. Perera, "Cybersecurity of industrial cyber-physical systems: a review." *ACM Comput. Surv.*, 2022.

[8] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges." *Cybersecurity*, vol. 2:20, pp. 1–22, 2019.

[9] P. Dini, A. Elhanashi, A. Begni, S. Saponara, Q. Zheng, and K. Gasmi, "Overview on intrusion detection systems design exploiting machine learning for networking cybersecurity," vol. 13, no. 13, p. 7507, 2023.

[10] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, p. 1–33, 2018.

[11] MathWorks. Adaptive cruise control system using model predictive control. [Online]. Available: https://it.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html

[12] Z. Wang, H. Wei, J. Wang, X. Zeng, and Y. Chang, "Security issues and solutions for connected and autonomous vehicle in sustainable city: A survey," *Sustainability*, vol. 14, no. 19, 2022.

[13] "ISO/SAE 21434: Road vehicles - cybersecurity engineering." *International Organization of Standardization, Society of Automotive Engineers.*, 2021.

[14] S. Hansen, C. Gomes, M. Najafi, T. Sommer, M. Blesken, I. Zacharias, O. Kotte, P. Mai, K. Schuch, K. Wernersson, C. Bertsch, T. Blochwitz, and A. Junghanns, "The fmi 3.0 standard interface for clocked and scheduled simulations," *Electronics*, vol. 11, p. 3635, 2022.

[15] P. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, and A. Sadovykh, "Integrated tool chain for model-based design of cyber-physical systems: The into-cps project," 2016, pp. 1–6.

[16] C. Gamble and K. Pierce, *Design Space Exploration for Embedded Systems Using Co-simulation*, 2014, pp. 199–222.

[17] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, p. 579–588, 2009.

[18] H. Moore, *MatLab for Engineers*, 2022.

[19] C. Legaard, D. Tola, T. Schranz, Macedo, H., and P. Larsen, "A universal mechanism for implementing functional mock-up units." 2021, pp. 121–129.

[20] C. Bernardeschi, A. Domenici, and M. Palmieri, "Formalization and co-simulation of attacks on cyber-physical systems." *Journal of Computer Virology and Hacking Techniques*, p. 63–77, 2020.

[21] H. Alibrahim and S. A. Ludwig, "Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 1551–1559.

[22] MathWorks. Sequential feature selection. [Online]. Available: https://it.mathworks.com/help/stats/sequential-feature-selection.html