Verification of neural networks meets PLC code: An LHC cooling tower control system at CERN

Ignacio D. Lopez-Miguel[1], Borja Fernández Adiego[2], Faiq Ghawash[3], and Enrique Blanco Viñuela[2]

[1] TU Wien, Vienna, Austria ignacio.lopez@tuwien.ac.at
[2] European Organization for Nuclear Research (CERN), Geneva, Switzerland borja.fernandez.adiego@cern.ch
[3] Norwegian University of Science and Technology, Trondheim, Norway faiq.ghawash@ntnu.no

Abstract. In the last few years, control engineers have started to use artificial neural networks (NNs) embedded in advanced feedback control algorithms. Its natural integration into existing controllers, such as programmable logic controllers (PLCs) or close to them, represents a challenge. Besides, the application of these algorithms in critical applications still raises concerns among control engineers due to the lack of safety guarantees. Building trustworthy NNs is still a challenge and their verification is attracting more attention nowadays. This paper discusses the peculiarities of formal verification of NNs controllers running on PLCs. It outlines a set of properties that should be satisfied by a NN that is intended to be deployed in a critical high-availability installation at CERN. It compares different methods to verify this NN and sketches our future research directions to find a safe NN.

Keywords: Verification of neural networks · PLCs · Control system.

1 Introduction

Programmable logic controllers (PLCs) are widely used in the process industry. In critical industrial installations, where a failure in the control system could have dramatic consequences, PLCs are used to control and protect industrial plants. This is mainly due to their hardware robustness, communication capabilities, their modularity, but also the simplicity of PLC programming compared with other programmable devices, giving them a high-reliability characteristic.

Using neural networks (NNs) as controllers is not novel [34], but it has seen exponential growth over the last years due to the increase in computation power (e.g., [28]). NNs are fast, they can operate in non-linear domains, and there is no need to know the dynamics of the systems as long as data are available. However, control engineers are still reluctant to use them in critical industrial installations due to the lack of safety, stability and robustness guarantees.

Whereas it is possible to prove certain properties in classic controllers (like efficiency, monotonicity, stability, robustness, etc.) and their behavior can be explained, it is not yet the complete case for NN-based controllers.

The goal of this paper is to analyze and compare different approaches to formally verify a NN for critical applications encoded in a PLC program. We will specifically focus on PLC code running on Siemens SIMATIC S7 PLCs. This paper makes special emphasis on the type of safety guarantees (verification properties) for this specific domain and the limitations of each verification method. The approaches are tested in an ongoing work for a NN-based controller for a cooling tower of the Large Hadron Collider (LHC) at CERN. The used NN is not the final version to be deployed in production, but this verification work will help us to find the appropriate one.

The main contributions of this paper can be summarized as follows:

- Provision of different properties that can be verified for a NN-based controller that is implemented on a PLC.
- Verification of a NN-based controller directly on PLC code using PLCverif.
- Verification of the same NN-based controller using a state-of-the-art NN verifier, `nnenum`, and using a state-of-the-art SMT solver, Z3.
- Comparison of the different techniques.
- Application of the previous methods in a real case study of a safety-critical system at CERN.

2 Background

2.1 Verification of NNs

Over the last years, the verification of NNs has raised its popularity due to the increasing number of applications of NNs in critical systems[4]. Robustness, especially against adversarial attacks, as well as reachability have been some of the main topics that have been targeted. Overapproximation of the activation functions and encoding the neural network as a mixed integer linear program [6], symbolic interval propagation [32,33], and SMT encoding [9] are some of the approaches to verify this type of properties.

Since neural networks can be used as feedback controllers, different reachability properties shall be checked. A wide variety of approaches exists, such as using a MILP encoding [2], modeling the systems with a neural network [8], and including perturbations [1].

The Verification of Neural Networks COMpetition (VNN-COMP) [24], shows the existance of many efficient tools like `nnenum` [3], VeriNet[16] or α,β-CROWN [35]. Normally, these tools are not very flexible, i.e., they only accept one type of activation function, one predefined architecture type, and a specific data type.

If during the NN verification, one finds a problem, it is necessary to repair it. For example, in [5] they re-train the neural network guided by the counterexamples until reaching a safe network. Other approaches do not reuse the training data [17]. However, this is an open research topic and there is no clear way to repair neural networks. Besides, when an engineer finds a counterexample, it helps them to better understand the behavior of the system [12].

[4] For a complete overview of this topic, please see [18].

2.2 PLCverif

PLCverif[5] is a plugin-oriented tool that allows the formal verification of PLC programs [22,31,7]. It has been used to verify various safety-critical programs [12,10,11]. In PLCverif, different requirement specification methods can be used. Moreover, different formal verification tools can be integrated. The PLCverif verification workflow consists of five main steps, as shown in figure 1:

Fig. 1: Formal verification workflow of PLCverif

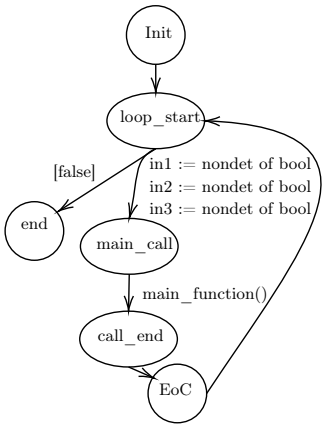

Fig. 2: CFA–Main PLC cycle.

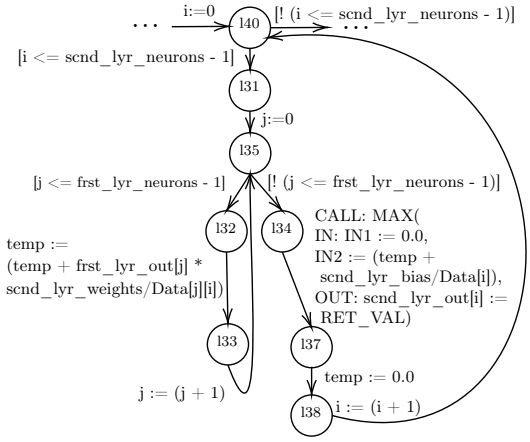

Fig. 3: CFA–Part of a NN.

1. **PLC program parser.** The PLC program is parsed and translated into a control flow-based representation, producing a so-called Control Flow Automata (CFA) [4]. Figure 2 shows a PLC cycle represented as a CFA, which calls its main function in every cycle (figure 3). This main function expresses as a CFA a part of a feedforward NN with ReLU functions (listing 1.1).
2. **Requirement representation.** The PLCverif user describes precisely the requirement to be checked in a natural manner. Thanks to the CFA representation, different types of properties can be included.
3. **CFA reductions.** The CFA is reduced to speed up the verification process.
4. **Model checkers.** A state-of-the-art model checker is executed.
5. **Reporting.** The results are given to the user in a human-readable manner.

[5] PLCverif is publicly available under https://gitlab.com/plcverif-oss.

```
scnd_lyr_neurons := 8;
FOR i := 0 TO scnd_lyr_neurons - 1 DO
 FOR j := 0 TO frst_lyr_neurons - 1 DO
  temp := temp + frst_lyr_out[j] * scnd_lyr_weights.Data[j, i];
 END_FOR;
 scnd_lyr_out[i]:=MAX(IN1:=0,IN2:=(temp+scnd_lyr_bias.Data[i])); temp:=0;
END_FOR;
```

Listing 1.1: First hidden layer processing of the NN for the LHC cooling tower.

3 Case study: the LHC cooling towers controls

At CERN, large-scale chilled water cooling facilities are installed at various locations along the LHC site to meet the cooling requirements of different clients (e.g chillers, cryogenics, air handling units, etc). Among various components of a large-scale cooling facility, induced draft cooling towers (IDCTs) are employed to cool the incoming hot water by rejecting the excess heat into the atmosphere. The typical arrangement of an IDCT involves the water entering the IDCT from the top and the ambient air enters from the bottom. The main components of an IDCT includes a fan, distribution system, spray nozzles, fill (packing), and collection basin. The cooled water is collected in the shared water collection basin before being supplied to different clients. Figure 4 shows multiple IDCTs with a shared water collection basin [15].

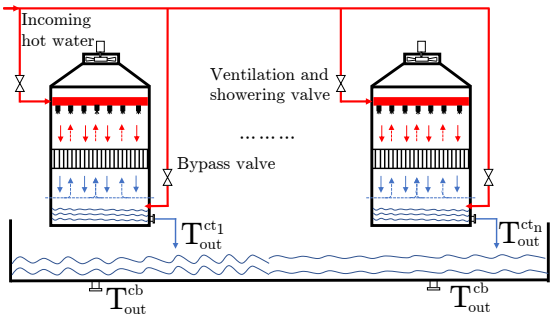

Fig. 4: Multiple induced draft cooling towers with a shared water collection basin.

The working principle of an IDCT is mainly based on simultaneous heat and mass transfer taking place between the hot water and the (cool) ambient air. Depending on the cooling requirement and ambient air temperature, an IDCT can be operated in different operational modes namely: *ventilation*, *showering*, and *bypass*. In the ventilation and showering mode, the bypass valve remains off and the hot water is sprayed downwards through spray nozzles. In the showering mode, the fan remains off and the cooling of the hot water takes place through

a natural draft. The mathematical model of the outlet water temperature under different operational modes is proposed in [19][29]. Moreover, a switched system representation is presented in [14] to compactly represent the dynamics in different operational modes of the IDCT.

3.1 Control design for the cooling towers

The primary objective of the cooling tower control design is to keep the outlet water temperature within strict limits ensuring the requirements of the downstream clients while utilizing the minimum amount of energy. The energy-optimal operation of the cooling towers requires the simultaneous determination of the best operational mode and optimal fan speed which poses a challenging control design problem.

The MPC (model predictive control) framework provides a structured way of designing the energy optimal control for the cooling towers. *The main idea behind MPC is to utilize the model of the system to predict future process behavior and minimize a given cost index subject to different physical and operational constraints.* It is based on solving a finite horizon-constrained optimal control problem at each sampling instant, resulting in the so-called *receding horizon control* [30]. Despite the advantages provided in terms of performance and energy optimization, the memory and computational resources required restrict their applicability to resource-constrained embedded hardware.

3.2 Approximate MPC using neural networks

In order to overcome the memory and computational requirements, approximate MPC is becoming a popular choice [27]. The approximate MPC requires lower memory and computational resources while preserving the performance of the controller. The idea of using a neural network to approximate the solution of a MPC has its origins in [25]. However, the efficacy of such techniques has been recently demonstrated for controlling nonlinear multiple-input multiple-output (MIMO) systems [13,26]. Depending on the size of the neural networks, the neural network controller can significantly reduce the computational times and memory requirement to traditional techniques and can be effectively deployed on resource-constrained embedded hardware, such as a PLC. However, the behavior of such controllers must be thoroughly investigated in terms of safety, stability, and robustness to be deployed in the production environment.

The preliminary version of the NN consists of 4 hidden layers with 8 neurons per layer. It combines a classification problem for the mode selection and a regression problem for the fan speed calculation. This is an initial version of the NN and will be improved in subsequent work.

4 Verification of a NN-based controller on a PLC

Different properties with respect to the previously mentioned NN that approximates a MPC will be verified using different methods.

4.1 Properties to verify

1. **Operational modes reachability.** Is there a combination of inputs (measurements from sensors) that reach mode M_i? This property is analyzed for the three previously introduced modes. Each of the counterexamples gives a combination of inputs that leads to each mode. If there would not be a counterexample for a certain mode, it would mean that mode is never going to be selected by the system, which is possibly an error in the design.
2. **Fan speed reachability.** Is there a combination of inputs that reach a certain range of the fan speed $[v_{min}, v_{max}]$? Different ranges are analyzed: $[0, 20)$, $[20, 60)$, $[60, 80)$, and $[80, 100]$. A counterexample gives a combination of inputs that leads to a fan speed in those ranges.
3. **Fan speed constraint satisfaction.** In *ventilation* mode, the fan should always operate within the desired range [60% - 100%], which can be verified. That is, is there a combination of inputs that leads to a fan speed lower than 60% or bigger than 100% when the mode is *ventilation*? If there is a counterexample, this could mean there is a problem in the network since the behavior is different than expected.
4. **Monotonicity.** If the mode is *ventilation* and all the temperatures increase, is the mode changing? It is expected that if all the temperatures increase, the mode remains at *ventilation*. A counterexample would show a case in which the temperatures increase and the mode is not *ventilation*.
5. **Softmax overflow.** Is there a combination of inputs that leads to a negative value of any of the outputs of the softmax layer of the modes? By definition of the softmax, since the exponential functions are always positive, the output of all the softmax layers should be always positive. If it is negative, it means there was an overflow in one of the components of the softmax formula.
6. **Robustness.** If the inputs are slightly changed, does the selected mode change? The counterexamples given by this property show the borders between the selection of the different modes. This could help the control engineer better understand if the controller is behaving as expected.

4.2 Verification of a NN with PLCverif

In order to tell PLCverif which variables should be non-deterministic so that the model checker explores all their possible values, we need to include those as input variables (`VAR_INPUT`) as shown in Listing 1.2. Instead of following the approach from [21], input variables are defined as integers but divided by 10 so that the input to the neural network has one decimal place.

```
1 VAR_INPUT
2  in_lyr_0, in_lyr_1, in_lyr_2 : Int; // non-deterministic
3 END_VAR
4 BEGIN // one decimal place
5  in_lyr[0]:=in_lyr_0/10; in_lyr[1]:=in_lyr_1/10; in_lyr[2]:=in_lyr_2/10;
```

Listing 1.2: PLC code for the input variables.

Since the input variables have a limited possible range (temperatures), we can tell CBMC not to explore all the possible values and assume they are in a given range as shown in Listing 1.3. This is done for all the inputs.

```
instance.input_layer_0 = nondet_int16_t();
__CPROVER_assume(instance.input_layer_0>=200 &&
                                    instance.input_layer_0<=250);
```

Listing 1.3: C code for CBMC to limit the range of the input variables.

Listing 1.4 shows how the previous properties can be encoded in PLC code so that they can be verified with PLCverif. Notice that property 6 has included the variable `max_mode_prev_cycle` from the previous cycle, which is defined as a temporary variable. This value is retained at the end of the cycle by adding an extra assignment after the assertions.

```
//#ASSERT NOT max_mode=0 : modesReachability0; // same for the other modes
//#ASSERT NOT (speed_layer_output[0]>=0 AND speed_layer_output[0]<20):
    fan_speed_reachability_0_20; // same for the other ranges
//#ASSERT max_mode=0 AND (speed_layer_output[0]>1 OR speed_layer_output
    [0]<0.6): fan_speed_constraint_satisfaction;
//#ASSERT NOT (in0>23.6 AND in1>23.0 AND in2>14.1 AND max_mode!=2) :
    monotonicity;
//#ASSERT NOT (modes_nn[0]<0 OR modes_nn[1]<0 OR modes_nn[2]<0):overflow;
//#ASSERT NOT (max_mode_prev_cycle!=max_mode) : robustness;
```

Listing 1.4: Translation of the properties into assertions.

Listing 1.5 shows the command to unwind the loops of the neural network 9 times (the maximum number of layers and of neurons in a layer is 8), and the global loop of the PLC cycle 2 times. The unwinding of 2 times of this loop is necessary to verify properties across 2 consecutive cycles.

```
cbmc neuralNetwork_prop7.c --unwind 9 --unwindset VerificationLoop.0:2
```

Listing 1.5: Command to execute CBMC unwinding the loops of the neural network 9 times, and the loop of the PLC cycle 2 times.

4.3 Verification of a NN using other methods

NN verifier. It was decided to use `nnenum` [3] since it is the best fully open-source neural network verifier according to the VNN-COMP (Report)[24] and due to its simplicity. It was needed to manually translate the NN weights given in the PLC code to the `.nnet` format in order to finally transform it to `ONNX`. Since the original NN had softmax functions in the output layers, they had to be ignored as `nnenum` cannot handle them. Furthermore, the original NN had to be split into two according to the different outputs (mode and fan speed) since `nnenum` only accepts NNs with one output. This led to the impossibility of verifying the properties in which both outputs are involved. In addition to `nnenum`, VeriNet [16] and α, β-CROWN [35] were tested without success due to compatibility and reproducibility issues.

SMT solver. Due to the numeric nature of the NN, an SMT solver (Z3 [23]) was used. According to the SMTcompetition, Z3 is one of the best and is open-source. All the loops were unwinded and the Python API for Z3 was used.

Exhaustive testing. Another possibility is to test every single combination of the inputs. Since we are limiting the number of possible input values by using integers and the number of inputs is small, this option was feasible.

5 Empirical results

All the previously presented properties were verified using the described different methods. Table 1 shows the results from these experiments. Since the first two properties are composed of more than one property, the mean and the standard deviation from those cases are shown. Clearly, `nnenum` is the fastest one since it is designed to work with NNs. PLCverif is the one with the lowest performance but it is the only one in which we can express all properties. Z3 is in the middle way between PLCverif and `nnenum`. A more detailed comparison of the different methods is presented in the next subsection 5.1.

	time (s)			
property	PLCverif	Z3	`nnenum`	cex. found
modes reachability	4932±5908	454±88	< 1	yes
fan speed reachability	6162±5909	1741±1588	< 1	yes
fan speed constraint satisfaction	2469	1049	-	yes
monotonicity	144	727	< 1	yes
softmax overflow	11	-	-	no
robustness	3517	2820	-	yes

Table 1: Results with the three approaches. Mean and standard deviation when different properties were checked. "-" means that it is not possible to verify that property with that method.

The code to run these experiments can be found in [20], as well as the results of their executions with the counterexamples. The experiments were done using CBMC 5.10, the Docker image of `nnenum` as of commit `cf7c0e7`, and the Python API of Z3 version 4.12.1.0. They were run on an AMD Ryzen 7 2700X at 4 GHz with 48GB RAM memory, running Ubuntu 20.4.

The results from exhaustive testing are not presented in the table since all the properties were checked simultaneously. We built our own testing infrastructure in Python, and the execution time for $51 \cdot 41 \cdot 131 = 273921$ tests (all combinations of the inputs) was 288 seconds.

The counterexamples given by PLCverif have been tested in the original PLC code to make sure the counterexamples were not spurious. This has been done using the integrated simulator *S7-PLCSIM Advanced* in the TIA portal

PLC programming environment. As an example, the counterexample given by `nnenum` for property 3 is shown in Figure 5. The three inputs corresponding to the three temperatures are forced to the values given by the counterexample and the expected output values for the fan speed and the operation modes (*ventilation*, *showering*, and *bypass*) are checked. The simulation shows that the counterexample is real and the problem exists in the NN.

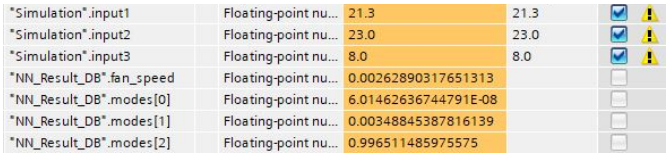

Fig. 5: Fan speed constraint satisfaction: Counterexample tested in S7-PLCSIM Advanced.

5.1 Comparison of the different approaches

Table 2 compares the different methods that were used. By using PLCverif, one can express more complex properties, such as the ones over time cycles. There is also no restriction on the architecture of the NN and the verification is done on the final model that will be deployed. Besides, there is no need to translate the NN to run a verification case. However, performance is low since it was not designed for this purpose. Nevertheless, since it is plug-in based, integrating an SMT solver such as Z3 without using a model checker could improve this issue.

	performance	scalability	expressiveness	same types?	plug-and-play?
PLCverif	low	low	**high**	**yes**	**yes**
`nnenum`	**very high**	**high**	low	no	no
Z3	medium	medium	low	no	no
Testing	high[6]	very low	medium	no	no

Table 2: Comparison of different methods to verify a NN.

On the other hand, `nnenum` is the opposite of PLCverif. That is, its performance was the best but it is not flexible, the type of properties that can be expressed is limited, the data types differ from the ones of a PLC, and a manual translation from the PLC code to ONNX is needed.

The verification with Z3 is in the middle way, where the performance is better than PLCverif, but worse than `nnenum`. As well, its expressiveness is worse than with PLCverif but better than with `nnenum`. Finally, the performance of testing

[6] For this particular example due to the limited number of inputs and their values.

was excellent in this particular example. It also gives all the counterexamples and it is relatively flexible. However, it will become unfeasible after a small increase in the number of variables or their possible values due to the exponential growth of the search space. This is independent of the NN architecture. PLCverif and Z3, on the contrary, suffer due to the NN architecture complexity.

6 Conclusions and future work

Different approaches to verify the ongoing work of a NN running on a PLC that will approximate a MPC for a real installation at CERN have been analyzed. Given the empirical results and the process to obtain them, the ideal approach would be to verify as much as possible with PLCverif. Once it becomes unfeasible due to performance issues, a NN verifier should be used. Finally, especially for the NN verifier due to the discrepancy in data types, the results should be checked using, for example, a simulator to avoid spurious counterexamples.

It is extremely important to verify a NN that will be deployed in a critical system to be sure that it will behave as expected. Verification can help with this endeavor and should be done together as part of the training of the NN until a safe NN is reached. This process will also help the control engineers to better understand the NN behavior as a feedback controller.

To the best of our knowledge, this is the first attempt to verify a neural network controller encoded on a PLC program. This initial study will help us to find a NN that satisfies the properties shown in this paper and new ones. Other future research directions include the analysis of how counterexamples can help improve the NN and the verification of closed-loop system properties.

References

1. Akintunde, M., Botoeva, E., Kouvaros, P., Lomuscio, A.: Formal verification of neural agents in non-deterministic environments. Autonomous Agents and Multi-Agent Systems **36** (04 2022). https://doi.org/10.1007/s10458-021-09529-3
2. Akintunde, M., Lomuscio, A., Maganti, L., Pirovano, E.: Reachability analysis for neural agent-environment systems. In: International Conference on Principles of Knowledge Representation and Reasoning (2018)
3. Bak, S.: nnenum: Verification of relu neural networks with optimized abstraction refinement. In: NASA Formal Methods Symposium. pp. 19–36. Springer (2021)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. International Journal on Software Tools for Technology Transfer **9**(5-6), 505–525 (Sep 2007). https://doi.org/10.1007/s10009-007-0044-z
5. Boetius, D., Leue, S., Sutter, T.: A robust optimisation perspective on counterexample-guided repair of neural networks. CoRR (01 2023)
6. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: AAAI Conference on Artificial Intelligence (2020)
7. Darvas, D., Fernández Adiego, B., Blanco Viñuela, E.: PLCverif: A tool to verify PLC programs based on model checking techniques. In: Proc. ICALEPCS'15 (10 2015). https://doi.org/10.18429/JACoW-ICALEPCS2015-WEPGF092

8. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Learning and verification of feedback control systems using feedforward neural networks. IFAC-PapersOnLine **51**(16), 151–156 (2018). https://doi.org/https://doi.org/10.1016/j.ifacol.2018.08.026, 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018
9. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis (2017)
10. Fernández Adiego, B., Blanco Viñuela, E.: Applying model checking to critical PLC applications : An ITER case study. In: Proc. ICALEPCS'17 (10 2017). https://doi.org/10.18429/JACoW-ICALEPCS2017-THPHA161
11. Fernández Adiego, B., Darvas, D., Blanco Viñuela, E., Tournier, J.C., Bliudze, S., Blech, J., González, V.: Applying model checking to industrial-sized PLC programs. IEEE Transactions on Industrial Informatics **11**, 1400–1410 (12 2015). https://doi.org/10.1109/TII.2015.2489184
12. Fernández Adiego, B., Lopez-Miguel, I.D., Tournier, J.C., Blanco Viñuela, E., Ladzinski, T., Havart, F.: Applying model checking to highly-configurable safety critical software: The SPS-PPS PLC program. In: Proc. ICALEPCS'21 (03 2022). https://doi.org/10.18429/JACoW-ICALEPCS2021-WEPV042
13. Georges, D.: A simple machine learning technique for model predictive control. In: 2019 27th Mediterranean Conference on Control and Automation (MED). pp. 69–74. IEEE (2019)
14. Ghawash, F., Hovd, M., Schofield, B.: Optimal control of induced draft cooling tower using mixed integer programming. In: 2021 IEEE Conference on Control Technology and Applications (CCTA). pp. 214–219. IEEE (2021)
15. Ghawash, F., Hovd, M., Schofield, B.: Model predictive control of induced draft cooling towers in a large scale cooling plant. IFAC-PapersOnLine **55**(7), 161–167 (2022)
16. Henriksen, P., Lomuscio, A.: Efficient neural network verification via adaptive refinement and adversarial search. In: European Conference on Artificial Intelligence (2020)
17. Henriksen, P., Leofante, F., Lomuscio, A.: Repairing misclassifications in neural networks using limited data. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. p. 1031–1038. SAC '22, Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3477314.3507059
18. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020). https://doi.org/https://doi.org/10.1016/j.cosrev.2020.100270
19. Jin, G.Y., Cai, W.J., Lu, L., Lee, E.L., Chiang, A.: A simplified modeling of mechanical cooling tower for control and optimization of hvac systems. Energy conversion and management **48**(2), 355–365 (2007)
20. Lopez-Miguel, I.D.: Verification of a neural network controller encoded on a PLC program. TU Wien Research Data (Mar 2023). https://doi.org/10.48436/fww3h-2y402
21. Lopez-Miguel, I.D., Adiego, B.F., Tournier, J.C., Viñuela, E.B., Rodriguez-Aguilar, J.A.: Simplification of numeric variables for plc model checking. In: Proceedings of the 19th ACM-IEEE International Conference on Formal Methods and Models for System Design. p. 10–20. MEMOCODE '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3487212.3487334

22. Lopez-Miguel, I.D., Tournier, J.C., Fernández Adiego, B.: PLCverif: Status of a formal verification tool for programmable logic controller. In: Proc. ICALEPCS'21 (03 2022). https://doi.org/10.18429/JACoW-ICALEPCS2021-MOPV042
23. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
24. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The third international verification of neural networks competition (vnn-comp 2022): Summary and results (2022). https://doi.org/10.48550/ARXIV.2212.10376
25. Parisini, T., Zoppoli, R.: A receding-horizon regulator for nonlinear systems and a neural approximation. Automatica **31**(10), 1443–1451 (1995)
26. Pasta, E., Carapellese, F., Mattiazzo, G.: Deep neural network trained to mimic nonlinear economic model predictive control: an application to a pendulum wave energy converter. In: 2021 IEEE Conference on Control Technology and Applications (CCTA). pp. 295–300. IEEE (2021)
27. Pin, G., Filippo, M., Pellegrino, F.A., Fenu, G., Parisini, T.: Approximate model predictive control laws for constrained nonlinear discrete-time systems: analysis and offline design. International Journal of Control **86**(5), 804–820 (2013)
28. Poznyak, A., Chairez, I., Poznyak, T.: A survey on artificial neural networks application for identification and control in environmental engineering: Biological and chemical systems with uncertain models. Annual Reviews in Control **48**, 250–272 (2019). https://doi.org/https://doi.org/10.1016/j.arcontrol.2019.07.003
29. Schofield, B., Peljo, M., Blanco, E., Booth, W.: Waste heat recovery for lhc cooling towers control system validation using digital twins. In: 17th International Conference on Accelerator and Large Experimental Physics Control Systems on and Information. New York, USA (2019)
30. Schwenzer, M., Ay, M., Bergs, T., Abel, D.: Review on model predictive control: An engineering perspective. The International Journal of Advanced Manufacturing Technology **117**(5-6), 1327–1349 (2021)
31. Viñuela, E.B., Darvas, D., Molnár, V.: PLCverif Re-engineered: An Open Platform for the Formal Analysis of PLC Programs. In: Proc. ICALEPCS'19. pp. 21–27. No. 17 in International Conference on Accelerator and Large Experimental Physics Control Systems, JACoW Publishing, Geneva, Switzerland (08 2020). https://doi.org/10.18429/JACoW-ICALEPCS2019-MOBPP01
32. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 6369–6379. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
33. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings of the 27th USENIX Conference on Security Symposium. p. 1599–1614. SEC'18, USENIX Association, USA (2018)
34. Werbos, P.: An overview of neural networks for control. IEEE Control Systems Magazine **11**(1), 40–41 (1991). https://doi.org/10.1109/37.103352
35. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=nVZtXBI6LNn