ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

TOWARDS SAFE AND ROBUST NEURAL NETWORK CONTROLLERS AT CERN: A REVIEW OF METHODS AND CHALLENGES

X. Fink[1]*, B.F. Adiego, B. Schofield, CERN, Geneva, Switzerland
J.-P. Katoen, RWTH Aachen University, Aachen, Germany
[1]also at RWTH Aachen University, Aachen, Germany

Abstract

Advances in optimization and machine learning algorithms have shown great potential when applied to control systems of many industries, such as automotive, avionics and aerospace. At CERN, we also find many applications in our particle accelerators and industrial facilities. In recent years, neural networks are increasingly being explored as components or even full replacements for model-based control systems, which rely on handcrafted rules or hard optimization schemes. In contrast, neural networks promise approximately-optimal performance while being trainable purely on already existing or simulated data. However, for critical control systems the behavior of any control-policy or dynamics-model needs to satisfy either hard or probabilistic guarantees. While model-based control realizes this by construction, neural networks models are known to exhibit unpredictable behavior, such as adversarial examples. Due to this, the use of formal methods for guaranteeing properties on neural networks has been widely explored in the literature. In this paper, we present an overview of the safety, robustness and stability challenges posed by neural network-based control systems at CERN. We examine how these challenges can be specified as formal properties and discuss state-of-the-art techniques for verifying and mitigating them.

INTRODUCTION

Model Predictive Control (MPC) and other optimization-based control techniques were developed to achieve near-optimal system performance while explicitly satisfying safety and stability constraints. In recent years, through advances in machine learning and computing hardware, *neural networks (NNs)* have become a viable option as powerful and effective function approximators for directly modeling complex dynamics or computing control actions. By learning from data, NNs can bypass the need for explicit physical models and enable approximate optimal control in scenarios where traditional methods struggle.

However, the adoption of neural networks in critical control systems raises new challenges. Prior work has demonstrated their susceptibility to *adversarial perturbations* [1]: small input changes that can lead to unexpected or unsafe outputs. Such perturbations may arise not only from malicious attacks but also from sensor noise, environmental disturbances, or modeling inaccuracies.

For industrial control systems with strict safety and reliability requirements, this raises fundamental questions about how to assure correct and predictable behavior when neural networks are part of the control loop. Currently, the IEC 61511 *functional safety standard* for the process industry sector does not consider the use of neural networks for safety instrumented systems.

At this moment, only a few standards address the unique challenges of machine learning in safety-critical contexts. Some examples are the automotive industry with the new ISO/PAS 8800, the ISO/IEC 24029 and the ISO/IEC 5469 technical report. These standards recommend *formal methods* and *formal verification* (mathematically sound algorithms with clear semantics) for high-risk systems. Extending these methods to neural networks and neural-network-controlled systems (NNCS) has attracted considerable research attention, particularly in the area of *adversarial robustness verification*. However, the intersection of perspectives from control theory, functional safety engineering, and computer science remains fragmented and challenging to navigate in practice.

At CERN, the unique safety requirements of large-scale accelerator facilities have motivated recent initiatives to explore formal verification techniques for neural network controllers. Based on our initial experience and a review of the state-of-the-art, this paper contributes a *CERN perspective* on the emerging field of safe neural-network-based control systems. In particular, we address 1) the roles that neural networks can play within control system architectures, 2) the challenge of providing unambiguous safety requirements and how to formalize them into verification properties, and 3) the current landscape of methods for verifying these properties.

Our goal is to provide practitioners and researchers with an analysis of the challenges, as well as a high-level overview of verification methods as a reference for future work and applications.

NEURAL NETWORKS IN CONTROL

In this section, we provide background on control system architectures and discuss how neural networks can be incorporated into them. We also introduce relevant terminology and formal definitions that will be used throughout the paper.

Control system notation Let $x_t \in \mathbb{R}^n$ denote the system state, $r_t \in \mathbb{R}^m$ the system state reference value, $u_t \in \mathbb{R}^k$ the control action, and $y_t \in \mathbb{R}^k$ the measured output at discrete time t. The behavior of a (time-varying) plant can then be defined based on state dynamics $f(x_t, u_t, t)$, measurements $h(x_t, t)$ and a control policy $\kappa(x_t, y_t, t)$:

* xaver.eugen.fink@cern.ch

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

$$x_{t+1} = f(x_t, u_t, t), \quad y_t = h(x_t, t), \quad u_t = \kappa(r_t, y_t, t). \tag{1}$$

Neural networks can be integrated into control systems as part of the control action computation or as learned approximations of the process dynamics, respectively replacing or augmenting κ or f. Both have been studied extensively in literature [2–4]. When a neural network replaces the control logic, the resulting system is often referred to as a neural network controlled system (NNCS). When the neural network learns from an existing model predictive control (MPC) this is often studied under the term approximate model predictive control (AMPC) [5, 6].

There have been several efforts at CERN to bring neural networks into control systems. Van der Veken et al. and Elen et al. give general overviews on future machine learning applications for the Large Hadron Collider [7, 8]. Szumega et al. use neural networks for parameter estimation of radiation monitors at CERN. This study already includes first steps into formal verification of the NN [9]. Ricci et al. have worked on using neural networks for the alignment of the LHC crystal collimators that are planned for LS3 [10]. There is an ongoing study of robustness verification of this neural network architecture. More use-cases include a neural network MPC approximation for cooling tower regulation [11], and neural network based reinforcement learning for beamline steering [12].

We distinguish three common roles for neural networks in control architectures with references to relevant CERN projects:

- **Model learning:** The neural network approximates the process dynamics, replacing or augmenting physical models, i.e., $x_{t+1} = f_\theta(x_t, u_t, t)$.
- **Closed-loop control:** The neural network is part of a feedback-loop, generating control inputs while taking into account the system's evolving dynamics over time, i.e., $u_t = f_\theta(r_t, y_t, t)$.
- **Parameter estimation:** The neural network learns to infer system parameters from signal data.

The scope of this paper is limited to *feedforward neural networks* (FFNNs), which are networks without cycles in the computation graph. The circular behavior in recurrent neural networks (RNNs) or their variants (e.g. LSTMs) make them significantly more challenging to verify but is being actively studied [13–16]. Note, that the property formalization process and some of the methods shown in this paper are still applicable to RNNs.

Notations of feedforward neural networks [17] We define a FFNN as usual as a function $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}, x \mapsto \hat{x}^{(m)}$, composed of m hidden layers $f_\theta(x) = h^{(m)} \circ h^{(m-1)} \circ \cdots \circ h^{(1)}$, with $h^{(i)}(x) = \sigma(W^{(i)}x)$, where σ is a nonlinear activation function and θ represents the learnable parameters of the weight matrices $W^{(i)}$. For simplification but without loss of generality we can omit the network bias as it can be represented as an additional node in each layer.

CHALLENGES FOR REQUIREMENT SPECIFICATION

Designing safe and effective neural networks for control systems requires a precise understanding what constitutes correct behavior, i.e. clearly defined system requirements. Intuitively, we require any system to satisfy *functional requirements* that specify the functionality of the system (e.g. "the temperature should stabilize around 30°C"), and *safety requirements* specifying configurations that must not occur (e.g. "the temperature must never go above 60°C"). A major practical difficulty in formal specification of these requirements is that the syntax and semantics coming from experts of different domains differ substantially.

Generally, in the design of a control system, control engineers work together with process engineers to define the functional requirements and with functional safety engineers to define the safety requirements when dealing with critical processes. The functional requirements of a control system are based on control theory and involve concepts that arise naturally from smooth differential systems such as *invariance, convergence, and asymptotic stability*. Functional safety engineers provide the safety requirements to mitigate the risks of the system, following the recommendations of functional safety standards. However, for the process industry, the current version of the standards still do not consider the use of neural networks as a mitigation measure. In this paper, we adopt the guidelines and recommendation to specify safety requirements for neural networks from the new standards mentioned in the previous section.

To enable formal verification of the requirements it is necessary to remove any ambiguity, often leading to additional complexity and effort. Formal methods, rooted in theoretical computer science, formalize the requirements as *safety or liveness properties* via mathematical logic systems such as *modal logic (e.g. linear temporal logic (LTL), computation tree logic (CTL)) or predicate logic (e.g. for pre- and post-conditions)*. Control, process and even functional safety engineers are not necessarily familiar with formal methods.

These differing languages and expectations commonly lead to misunderstandings during the property specification process where people from different areas of expertise are involved. Moreover, tools for verifying neural networks or NN-controlled systems often reflect the domain they originate from, supporting only domain-specific requirement types or levels of abstracting. Thus, one of the challenges of neural network system assurance is *to unify the domain-requirements into precise semantics*. This challenge is addressed in the guidelines provided by the ISO/PAS 8800, ISO/IEC 24029, and ISO/IEC 5469 standards. The standards recommend the following workflow for the specification process:

1. Functional safety engineer defines high-level safety requirements with experts based on a risk analysis.
2. Control engineer defines a concrete safety spec (e.g., "remain within safe temperature bounds").

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

3. Formal methods experts encode the safety spec into formal verification properties usable as a tool input – control equations for tools from the control domain or logic formulas for tools from the formal methods domain.

This process often has to be iterated several times until a precise and valid specification is reached (Fig. 1). A goal for the future is to create a verification toolset that can be used directly by the functional safety or control engineer, inspired by existing toolsets for similar purposes at other large scale organizations (e.g. FRET for NASA Ames [18] and the COMPASS toolset for ESA [19]).

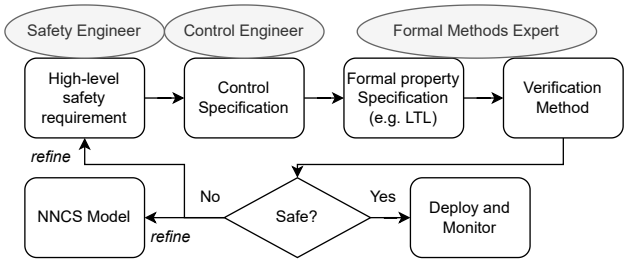

Figure 1: Specification pipeline.

Static and Temporal Properties

Properties over a neural network in a control system can be either static or temporal, dependent on whether loop-feedback behavior is taken into account or not. Analyzing feedback behavior takes into account that the system state changes over time, requiring temporal logic reasoning. The formal model of an unbounded feedback-loop may have infinite execution paths. Static behavior on the other hand can be formulated through comparatively simple input-output relations. This difference changes drastically (a) how verification problems are formulated, and (b) what tools and methods are applicable.

Static System Behavior When analyzing static properties, the neural network acts as a one-step static function:

$$u = f_\theta(x),$$

with no feedback from a plant during runtime. On this system, we are then interested in properties such as safety ("an unsafe set $\mathcal{U}$ cannot be reached from a given configuration") or reachability ("the output is always in a safe set $\mathcal{S}$ for a given configuration"). These properties can be formulated as input-output predicates.

Formally, we can unify these properties under a verification problem framework capable of handling a large set of properties, as defined by Liu et al. [17]. The framework defines a verification problem by fixing input and output sets $X \subseteq D_x$ and $Y \subseteq D_y$ where the verification problem is to evaluate:

$$x \in X \implies y = f(x) \in Y.$$

In practice this verification problem is hard to solve [17]. Most algorithms and tools restrict themselves to solving a local verification problem which restricts X to be an l_p norm perturbation around a fixed x. Additionally, the output set is often fixed to a set that can be described via a linear constraint system. A commonly supported specification format for this is defined in the VNN-LIB standard [20].

This class of properties is captured by the yearly competition for neural network verification (VNNCOMP) [21]. Additionally, many nonlinear properties can still be modeled via modifications to the FFNN model. This is possible whenever the nonlinearity can be expressed by operations supported by the chosen verification algorithm.

Feedback-Loop Behavior For feedback-loop properties, the neural network is part of a dynamical system where its output influences future inputs:

$$x_{t+1} = f_{plant}(x_t, u_t), \quad u_t = f_\theta(x_t).$$

Thus, properties must reason about sequences of states over time, which are also referred to as *traces*. Formally, this can be modeled via first order (or higher order) logic such a linear temporal logic (LTL) or computation tree logic (CTL), e.g. to guarantee that a system always stays within a safe set $\mathcal{X}_{safe}$,

$$\forall t \in T : x_t \in \mathcal{X}_{safe}.$$

Verification of such properties is generally no longer possible via the methods for static neural network verification but requires abstractions or reasoning over traces. A common simplification is to restrict temporal properties to bounded temporal behavior. Several tools for verifying this property class compete annually in the Artificial Intelligence and Neural Network Control Systems branch of the Competition on Verifying Continuous and Hybrid Systems (ARCH-COMP) [22].

In certain cases a temporal property can be reformulated into the verification of propositional logic. This can allow the verification via simpler methods also used for static properties. However, intuitively this is not always possible, e.g. due to feedback effects, delayed consequences, or instability.

Property Formalization Example In previous work at CERN, Lopez et al. verified selected control properties on a cooling tower [11]. The properties can all be characterized as propositional or first-order logic formulas over an instance of the verification problem. We give an example for the fan-speed reachability property which asks if a certain fan speed operation range can be reached, i.e. if for a fixed range $[v_{min}, v_{max}]$ there exists an input x with $f(x) \in [v_{min}, v_{max}]$. This can be formalized under the verification problem framework by specifying the inverse problem as an instance with $X = D_x$ and $Y = \{y \mid \text{fanspeed}(y) \notin [v_{min}, v_{max}]\}$. A valid solution to this verification problem is then a counterexample to the fan-speed reachability property.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

VERIFICATION ALGORITHMS

This section provides an overview of methods for verifying FFNNs and NNCS, classifies them into static and temporal property verification, and analyses their potential for usage at CERN. We assume the neural network is fully trained, and focus on analyzing whether it satisfies specified safety or functional properties under given conditions. While also relevant to the broader discussion, due to the limited scope of this paper we do not discuss:

- design, training, or retraining methods for improving a network's safety by construction [23];
- methods for verifying or correcting a network's behavior during runtime [24].

Static Verification

As static neural network verification we refer to algorithms that analyze the input-output relation of a neural network in isolation without considering surrounding control system information.

Liu *et al.* [17] and König *et al.* [25] give surveys on robustness verification algorithms, in-depth algorithm descriptions, tool-survey, and performance comparisons. Bunel *et al.* [26] unify verification algorithms for piecewise linear neural networks (i.e. network with ReLU activation functions or similar) with an in-depth explanation of the branch and bound framework.

The methods are classified into whether they compute a reachable set, solve an optimization problem, or simply search for a counterexample.

- **Reachable set propagation** The reachable output-set is computed by propagating the input set layer-by-layer. The methods differ in the abstraction of the propagated set *(e.g. intervals, polyhedra, polynomials)* and the propagation algorithm. Reachability computations usually introduce over-approximations caused by neural network nonlinearities, making these methods sound but incomplete.
- **Optimization** A verification problem is verified by encoding the neural network into an optimization problem *(e.g. mixed-integer programs (MIP), satisfiability modulo theories (SMT))*. The solution to this optimization problem can then be used to trivially check the verification problem.
- **Search** This category encompasses algorithms that directly search for a counterexample *(e.g. gradient-based adversarial attacks)*. This is usually combined with a reachability and/or optimization algorithm to guide the search.

Individually, reachability and optimization fail when verifying high-dimensional or deep neural networks: reachability tends to over-approximate too conservatively, while global optimization methods suffer from combinatorial complexity and thus do not scale. Combined methods, however, form the basis for current state-of-the-art tools that have demonstrated verification of properties on networks with up to tens of thousands of ReLUs and high-dimensional inputs (e.g. ImageNet) [VNNCOMP]. Additionally, most methods use some level of abstraction or even abstraction refinement. They can be combined with a branch and bound (BaB) procedure to regain soundness with a potentially exponential increase in runtime.

Feedback-loop Verification

As discussed in Chapter 2, feedback-loop property verification requires reasoning over multiple time-steps. Here we give an intuition on the methods to achieve this.

Reachable Set Propagation for Feedback-Loop Systems Given the success of static neural network verification methods there have been efforts to use them for feedback-loop verification. While some approaches give unbounded reachability results via inductive reasoning, most of these hybrid methods compute bounded reachability results (similar to bounded model checking), i.e. the reachable set after k feedback-loop steps.

However, this requires the plant dynamics and any pre- or post-processing steps to be compatible with the reachability abstraction or optimization framework used for the neural network analysis. Some approaches combine tools for dynamical system analysis such as Flow* with static neural network verification tools. Another option uses a polynomial abstraction to directly propagate bounds through both the network and dynamics [27]. Finally, general purpose tools and algorithms for the verification of hybrid systems can be applied to NNCS, however most methods do not scale to high-parameter neural networks.

Control Lyapunov Barrier Functions An alternative to reachability is to prove unbounded system stability and safety via control Lyapunov functions (CLFs) and control barrier functions (CBFs). CLFs ensure the system state converges to a goal set; CBFs guarantee it remains within safe bounds. Their combination, control Lyapunov-barrier functions (CLBFs), allows both properties to be verified simultaneously [28–31].

For NNCS, this often involves constructing or learning a candidate Lyapunov or barrier certificate and then verifying it holds over the input domain. This may be done by encoding the certificate as a static property verified with standard NN reachability or optimization tools. While this method is attractive for continuous control tasks at CERN (e.g. temperature regulation), practical construction of valid certificates for large nonlinear NNs remains an active research challenge [32–34].

Model Checking Another approach is to use general purpose model checking tools such as NuSMV. The main benefits of this are ease-of-use and expressiveness. Software model checking tools commonly support first-order logic properties as well as rich expressiveness of the system model specification, which would directly enable the verification of complex properties on feedback-loop systems. Furthermore, by verifying the code of the final execution platform, bugs

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

introduced by translation layers can be found which could go unnoticed at a design-level verification [35, 36].

However, literature and our experience shows that model checking is not a suitable approach for high-dimensional or deep neural networks. The main problems are the extremely large state spaces introduced by the neural network parameters and expensive floating point computations.

Probabilistic Approaches

Instead of delivering hard qualitative results, stochastic approaches give stochastic, often quantitative, certainty results on the probability of property violations occurring. This can be a reasonable approach when parts of the verification problem can be modeled as a stochastic process, e.g. uncertainty in the form of input noise. Note that many stochastic approaches no longer fall under the umbrella of formal methods, but hybrid approaches have been explored [37, 38].

Probabilistic approaches can be applied in several ways, for example:

- **Sampling** Some approaches sample the input space and give probabilistic guarantees based on the resulting output-sample distribution [39]. Other approaches use sampling for randomized smoothing to derive provably more robust classifiers [40].
- **Distillation and probabilistic verification** The input space is sampled for execution-traces through the neural network controlled system. Based on the traces a compact surrogate model is generated that can be analyzed via probabilistic verification methods such as probabilistic model checking [41]. Counterexamples can be used to improve the neural network system in an iterative procedure.
- **Distribution transformation** The input domain is treated as a probability distribution that can be propagated through the neural network in a layer-by-layer manner as in reachability analysis. Unlike conventional reachability analysis, the output of the analysis is a probability distribution that allows the verification of quantitative probabilistic properties.

Many of the probabilistic analysis results can be seen as a probabilistic reachable set result. This result could be reused in the same manner as the previously discussed reachable set analysis, however probabilistic distribution are generally harder to accurately propagate through computation trees.

Table 1 compares the different methods for the aspects of property expressiveness, scalability and whether they can handle feedback-loop properties. The table highlights that no single method covers all use-cases.

Table 1: Comparison of Key Verification Methods, Their Property Type ($\top$ = Correctness, $\top^*$ = Incomplete Correctness, $\bot$ = Falsification), Typical Scalability, and Feedback-Loop Suitability. The "K-Reach" Method Stands for Reachable Set Propagation over k Time Steps and Typically Only Allows Bounded Feedback-Loop Verification

Method	Property	Scalability	Feedback-loop
Reach-Set propagation	$\bot, \top^*$	+	No
Optimization	$\bot, \top$	-	No
Search	$\bot$	++	No
k-Reach	$\bot, \top^*$	o	Yes*
Lyapunov-Barrier Functions	$\bot, \top^*$	o	Yes
Model checking	$\bot, \top$	- -	Yes

CONCLUSION

Neural network-controlled systems offer new opportunities for high-performance control at CERN, but their unpredictable behavior poses major safety and reliability challenges. This paper reviewed the main roles of neural networks in control systems, discussed the challenge of property specification, and outlined possible verification methods for use-cases at CERN. Future work will focus on bridging the gap between theoretical guarantees and operational needs through concrete case-studies, verification tool-chains and algorithms for safe training or runtime monitoring. Additionally, the first steps will be taken to applying the newly developed functional safety-standards to CERN neural network projects.

REFERENCES

[1] C. Szegedy *et al.*, "Intriguing properties of neural networks", in *Proc. ICLR'14*, Banff, AB, Canada, Apr. 2014. doi:10.48550/arXiv.1312.6199

[2] M. T. Hagan and H. B. Demuth, "Neural networks for control", in *Proc. American Control Conference*, San Diego, CA, USA, Jun. 1999, pp. 1642–1656. doi:10.1109/ACC.1999.786109

[3] S. S. Ge, C. C. Hang, T. H. Lee, and T. Zhang, *Stable Adaptive Neural Network Control*. Boston, MA: Springer US, 2002.

[4] M. T. Hagan, H. B. Demuth, and O. De Jesus, "An introduction to the use of neural networks in control systems", *Int. J. Robust Nonlin. Control*, vol. 12, no. 11, pp. 959–985, 2002. doi:10.1002/rnc.727

[5] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller", *J. Process Control*, vol. 16, no. 9, pp. 937–946, Oct. 2006. doi:10.1016/j.jprocont.2006.06.001

[6] H. Hose *et al.*, "Approximate non-linear model predictive control with safety-augmented neural networks", arXiv:2304.09575v2 [eess.SY], Apr. 2023. doi:10.48550/arXiv.2304.09575

[7] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile, "Neural Networks for Modeling and Control of Particle Accelerators", *IEEE Trans. Nucl. Sci.*, vol. 63, no. 2, pp. 878–897, Apr. 2016. doi:10.1109/TNS.2016.2543203

[8] F. Van Der Veken *et al.*, "Application of machine learning techniques at the CERN Large Hadron Collider", *Proc. Sci.*, vol. 364, p. 006, Nov. 2020. doi:10.22323/1.364.0006

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

[9] J. M. Szumega, H. Boukabache, and D. Perrin, "Neural network approach for efficient calculation of the current correction value in the femtoampere range for a new generation of ionizing radiation monitors at CERN", *Radiat. Phys. Chem.*, vol. 188, p. 109539, Nov. 2021. doi:10.1016/j.radphyschem.2021.109539

[10] G. Ricci *et al.*, "Machine learning based crystal collimator alignment optimization", *Phys. Rev. Accel. Beams*, vol. 27, no. 9, p. 093001, Sep. 2024. doi:10.1103/PhysRevAccelBeams.27.093001

[11] I. D. Lopez-Miguel *et al.*, "Verification of Neural Networks Meets PLC Code: An LHC Cooling Tower Control System at CERN", in *Proc. EANN'23*, Limassol, Cyprus, Jun. 2023, pp. 420–432. doi:10.1007/978-3-031-34204-2_35

[12] V. Kain *et al.*, "Sample-efficient reinforcement learning for CERN accelerator control", *Phys. Rev. Accel. Beams*, vol. 23, no. 12, p. 124801, Dec. 2020. doi:10.1103/PhysRevAccelBeams.23.124801

[13] T. Du *et al.*, "Cert-RNN: Towards Certifying the Robustness of Recurrent Neural Networks", in *Proc. ACM CCS'21*, Virtual Event, Republic of Korea, Nov. 2021, pp. 516–534. doi:10.1145/3460120.3484538

[14] Z. Shi *et al.*, "Neural Network Verification with Branch-and-Bound for General Nonlinearities", in *Proc. TACAS'25*, Hamilton, ON, Canada, May 2025, pp. 315–335. doi:10.1007/978-3-031-90643-5_17

[15] S. Carr, N. Jansen, and U. Topcu, "Verifiable RNN-Based Policies for POMDPs Under Temporal Logic Constraints", in *Proc. IJCAI'20*, Yokohama, Japan, Jul. 2020, pp. 4121–4127. doi:10.24963/ijcai.2020/570

[16] M. E. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano, "Verification of RNN-Based Neural Agent-Environment Systems", *Proc. AAAI'19*, vol. 33, no. 01, pp. 6006–6013, Jul. 2019. doi:10.1609/aaai.v33i01.33016006

[17] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for Verifying Deep Neural Networks", *Found. Trends Optim.*, vol. 4, no. 3-4, pp. 244–404, Feb. 2021. doi:10.1561/2400000035

[18] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Rhein, J. Schumann, and N. Shi, "Formal requirements elicitation with FRET", in *Proc. REFSQ-2020*, ARC-E-DAA-TN77785, 2020.

[19] M. Bozzano *et al.*, "COMPASS 3.0", in *Proc. TACAS'19*, vol. 11427, 2019, pp. 379–385. doi:10.1007/978-3-030-17462-0_25

[20] S. Demarchi *et al.*, "Supporting Standardization of Neural Networks Verification with VNNLIB and CoCoNet", in *Proc. FoMLAS'23*, Aug. 2023, pp. 47–58. doi:10.29007/5pdh

[21] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the international verification of neural networks competition (VNN-COMP)", *Int. J. Softw. Tools Technol. Transf.*, vol. 25, pp. 329–339, May 2023. doi:10.1007/s10009-023-00703-4

[22] D. Manzanas Lopez *et al.*, "ARCH-COMP24 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants", in *Proc. ARCH'24*, 2024, pp. 64–65. doi:10.29007/mxld

[23] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", in *Proc. ICLR'18*, Vancouver, BC, Canada, Apr. 2018. doi:10.48550/arXiv.1706.06083

[24] K. L. Hobbs *et al.*, "Runtime Assurance for Safety-Critical Systems: An Introduction to Safety Filtering Approaches for Complex Control Systems", *IEEE Contr. Syst. Mag.*, vol. 43, no. 2, pp. 28–65, Apr. 2023. doi:10.1109/MCS.2023.3234380

[25] M. König, A. W. Bosman, H. H. Hoos, and J. N. van Rijn, "Critically Assessing the State of the Art in Neural Network Verification", *J. Mach. Learn. Res.*, vol. 25, no. 12, pp. 1–53, 2024.

[26] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda, "A Unified View of Piecewise Linear Neural Network Verification", in *NeurIPS 2018)*, Montreal, Canada, May 2018, vol. 31.

[27] C. Huang *et al.*, "POLAR: A Polynomial Arithmetic Framework for Verifying Neural-Network Controlled Systems", arXiv:2106.13867 [eess.SY], 2022. doi:10.48550/arXiv.2106.13867

[28] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems", in *Proc. ACM HSCC'19*, Montreal, QC, Canada, Apr. 2019, pp. 147–156. doi:10.1145/3302504.3311802

[29] M. Everett, "Neural Network Verification in Control", in *Proc. IEEE CDC'21*, Austin, TX, USA, Dec. 2021, pp. 6326–6340. doi:10.1109/CDC45484.2021.9683154

[30] R. Schwan, C. N. Jones, and D. Kuhn, "Stability Verification of Neural Network Controllers Using Mixed-Integer Programming", *IEEE Trans. Autom. Control*, vol. 68, no. 12, pp. 7514–7529, Dec. 2023. doi:10.1109/TAC.2023.3283213

[31] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control Barrier Functions: Theory and Applications", in *Proc. 18th European Control Conference (ECC)*, Naples, Italy, Jun. 2019, pp. 3420–3431. doi:10.23919/ECC.2019.8796030

[32] L. Yang *et al.*, "Lyapunov-stable Neural Control for State and Output Feedback: A Novel Formulation", arXiv:2404.07956 [cs.LG], 2024. doi:10.48550/arXiv.2404.07956

[33] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning Safe Multi-Agent Control with Decentralized Neural Barrier Certificates", in *Proc. ICLR'21*, Virtual Event, May 2021.

[34] U. Mandal *et al.*, "Formally Verifying Deep Reinforcement Learning Controllers with Lyapunov Barrier Certificates", arXiv:2405.14058 [cs.AI], 2024. doi:10.48550/arXiv.2405.14058

[35] E. Manino, R. S. Menezes, F. Shmarov, and L. C. Cordeiro, "NeuroCodeBench: a plain C neural network benchmark for software verification", arXiv:2309.03617 [cs.SE], Sep. 2023. doi:10.48550/arXiv.2309.03617

[36] L. Sena *et al.*, "Verifying Quantized Neural Networks using SMT-Based Model Checking", arXiv:2106.05997 [cs.LG], 2021. doi:10.48550/arXiv.2106.05997

[37] B. Sun, J. Sun, T. Dai, and L. Zhang, "Probabilistic Verification of Neural Networks Against Group Fairness", arXiv:2107.08362 [cs.LG], Jul. 2021. doi:10.48550/arXiv.2107.08362

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-WEPD074

[38] D. Boetius, S. Leue, and T. Sutter, "Probabilistic Verification of Neural Networks using Branch and Bound", arXiv:2405.17556 [cs.LG], 2025. doi:10.48550/arXiv.2405.17556

[39] V. Sivaramakrishnan, K. C. Kalagarla, R. Devonport, J. Pilipovsky, P. Tsiotras, and M. Oishi, "SAVER: A Toolbox for Sampling-Based, Probabilistic Verification of Neural Networks", arXiv:2412.02940 [cs.LG], Dec. 2024. doi:10.48550/arXiv.2412.02940

[40] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified Adversarial Robustness via Randomized Smoothing", in *Proc. ICML'19*, Long Beach, CA, USA, Jun. 2019.

[41] J.-P. Katoen, "The Probabilistic Model Checking Landscape", in *Proc. LICS'16*, New York, NY, USA, Jul. 2016, pp. 31–45. doi:10.1145/2933575.2934574