Counter-Example Guided Training of Neural Network Controllers

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Neural Networks have the potential to act as feedback controllers for complex nonlinear dynamical systems due to the excellent approximation accuracy they can achieve thanks to advanced deep learning technology. However, due to the difficulties in analyzing and verifying the closed-loop system, neural network controllers are not widely adopted as feedback controllers. In this paper, we present a technique based on semi-formal methods to synthesize reliable neural network controllers with safety and performance guarantees captured in Signal Temporal Logic (STL). Initially, we train the neural network using the training data generated from a nominal controller based on a coverage matrix. Subsequently, we employ a falsification technique to identify the scenarios that represent the violation of the STL specifications by the closed-loop system. We utilize these scenarios entirely or selectively via clustering to generate additional training data and retrain the neural network to improve its performance. The training-falsification loop continues until we find a controller that satisfies all the specifications. We apply our technique to synthesize neural network controllers for two benchmark dynamical systems, viz. a water tank and a robot arm, with encouraging results.

1 Introduction

Neural networks (NNs) have long been used to control dynamical systems from inverted pendulums to quadcopters, learning from scratch to control the plant by maximizing an expected reward, e.g. [1, 2]. In this paper, we consider the case where NNs are trained to replace an existing controller that is unsatisfactory for non-functional reasons, e.g., computationally expensive (consider model-predictive control), slow, or energy intensive. A well-trained NN controller can provide the same performance much faster and is readily implemented on cheap and energy-efficient embedded platforms [3]. We assume that a controller is available and that we can use it to drive the plant (or a model of the plant), observing its inputs and outputs. Our goal is to train a NN that behaves the same as the nominal controller. Our approach is characterized by three key features: (i) we observe the nominal controller in closed-loop with the plant, (ii) we can train to satisfy complex temporal properties, and (iii) we leverage the power of existing falsification tools to create training data that matters. In the following paragraphs, we will explain these choices.

Training a NN to behave like a nominal controller may look, at first glance, like a simple problem of approximating a function – the nominal controller – with a NN. One could observe the output of the controller on a dense enough sampling of the input space and use the input-output pairs as training data for the NN. In practice, however, there are several difficulties that can be addressed by observing the nominal controller in closed-loop, e.g., while running the plant:

- Large input space: The number of samples required to cover the input space with sufficient density may be prohibitively large.
- Stateful controller: When the controller has memory, we need to sample not individual input vectors but its input history. The sampling space increases exponentially with the length of the required history.

- Nonuniform accuracy: Depending on the controller specification, the NN may need to be very precise around some inputs while in other input regions a rough approximation works fine.

Since a stable controller brings the plant very quickly back to a small neighborhood around its equilibrium or reference trajectory, it suffices to learn in those subspaces. This is easily achieved by observing closed-loop behaviors, which drastically reduces the space to be sampled. The problem of nonuniform accuracy is particularly pronounced when the specification depends on time or sequences of events. This is frequently the case in control applications, where properties such as rise time, settling time, and overshoot are typical. We consider complex properties that can include not only time but also causal relationships. They are described in *Signal Temporal Logic* (STL), a powerful formal language that finds widespread use in formal methods and increasing adoption in industry [4, 5]. While observing closed-loop behaviors may reduce the number of inputs to be sampled, we still need to find good training samples. This problem is not trivial, since uniform sampling turns out to lead to unsatisfactory results, in particular around steady state behaviors, unless very high density is used. In addition, we need to choose the initial state from which to drive the plant, as well as the reference trajectory. To generate samples that are relevant to the STL properties that we want to satisfy, we leverage existing, highly optimized, *falsification tools* (falsifiers) [6, 7]. The goal in falsification is to find a *counter-example* (cex), i.e., an input (here, the reference trajectory) and an initial plant state such that closed-loop behavior violates the given STL property. We feed our NN controller, the plant, and the specification to a falsification tool. If the falsifier does not find a cex, we consider our NN controller satisfactory and stop. If the falsifier finds a cex, we replay it with the nominal controller in order to obtain new training data, and retrain the network.

In summary, we make the following contributions in this paper:

- We introduce the problem of synthesizing a neural network controller for a nonlinear dynamical system where the specification is given in terms of a set of STL formulas.
- We provide a counterexample-guided controller refinement algorithm to solve the synthesis problem. Our algorithm is based on generating initial training data satisfying a coverage criterion and then subsequent retraining based on the training data generated from the counterexamples.
- We implement our algorithm in MATLAB® using the state-of-the-art falsification tool Breach. Using our tool, we successfully synthesize neural network controllers for a water tank and a robot arm system satisfying several interesting STL properties.

2 Related Work

Recent research has shown the feasibility of using deep neural networks as feedback controllers for nonlinear dynamical systems through supervised learning [2, 8, 9, 10, 11, 3] and reinforcement learning [12, 13, 14, 15]. However, for the neural network-based controllers to receive wide acceptability as an alternative of the traditional controllers like PID [16], LQR [17], and MPC [18], one should be able to establish safety and performance guarantees on the closed-loop system. In the last decade, there has been significant progress in verifying neural networks against safety [19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29] and robustness properties [30, 31, 32, 33, 34, 35]. Though these techniques are useful in evaluating the soundness of a neural network in various vision applications, they are not directly applicable to the closed-loop dynamical system with a neural network controller as the soundness of the closed-loop system depends on the close coupling of the controller with the plant. Recently researchers have developed tools to address the verification problem for the closed-loop control systems [36, 37, 38, 39]. These verification tools involve the reachability analysis of a hybrid system [40, 41, 42], which is computationally hard, resulting in the tools to be not scalable. Several research groups have developed methodologies for systematic testing of a closed-loop system with a neural network controller [43, 44, 45, 46]. Though these methods are effective in identifying scenarios that demonstrate the weakness of the neural network controller, they do not provide any guarantee on the correctness and performance.

To avoid the complexity of verification of the closed-loop system with a neural network feedback controller, a number of recent papers have explored the possibility of synthesizing the correct-by-construction AI-based feedback controller. Ferlez and Shoukry [47] present a method for deciding the structure of the neural network controller with a guarantee that there exist the weights of the neural network that allow it to approximate an MPC precisely. Berkenkamp et at. demonstrated how a deep reinforcement learning algorithm can be applied to learn a neural network control policy without violating the safety requirements [48].

The principle of learning to control through counterexamples goes back to at least Henzinger et al. [49], who applied it to automata. Chang et al. [50] propose a methodology for the co-synthesis of NNs representing a feedback controller and a Lyapunov function certifying the stability of the closed-loop system. Their method relies on the refinement of the NNs based on the training data generated from the counterexample showing that the trajectory of the closed-loop system is violating the Lyapunov condition. The closest to our work is the paper by Clavière et al. [51], where the authors approximate the behavior of a model predictive controller with a deep neural network to enable a robot to follow a reference trajectory closely. They employ counterexample-guided refinement of the NN by generating additional training data from counterexamples. However, unlike our framework, their approach is limited to memoryless controllers, and they use tracking closeness as the sole criterion to identify counter-examples. In our experience, a NN need not always closely track the nominal behaviour to satisfy the specifications. It may have some characteristics that are better than the nominal controller, such as a smaller overshoot, or quicker stabilization in some areas of the state space. Such behaviours would be eliminated using their approach.

3 Closed-Loop Specification Guided Approach for NN Controller Learning

We consider a continuous-time plant Π with state $x \in \mathbb{R}^n$ that is controlled by an input signal u and observed through an output signal y, with dynamics

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$
$$y(t) = w(x(t)). \tag{2}$$

The control input u is computed by a *nominal* discrete-time controller $\mathcal{C}$ with state $z \in \mathbb{R}^{n_z}$

$$z_{k+1} = f_c(z_k, y_k, r_k), \tag{3}$$
$$u_k = g(z_k, y_k, r_k) \tag{4}$$

where r_k and y_k are discrete-time signals resulting from sampling the reference $r(\cdot)$ and the output $y(\cdot)$ with a constant time step h, that is $\forall k = 0, 1, \dots r_k = u(kh), y_k = y(kh)$. The continuous-time control $u(\cdot)$ is a piece-wise constant function defined as $\forall t \in [kh, (k+1)h)\ u(t) = u_k$ with $k = 0, 1, \dots$. Starting from an initial state $x(0) = x_0 \in X_0$, the behaviour of the system is a vector of functions $\gamma = (r(\cdot), x(\cdot), u(\cdot), y(\cdot))$ that satisfies the system equations. We use the usual Lipschitz assumptions about the function f so as to guarantee the existence and uniqueness of the solution. We introduce the notion of "control setting" to refer to a pair $s = (x_0, r(\cdot))$ where x_0 is the initial state of the plant and $r(\cdot)$ is the reference signal. Let $\mathcal{S}$ denote the set of all possible control settings. Under each control setting s, the controller (represented by the function f_c) generates a behaviour, and for convenience we denote it by γ_s.

Control Specification. Our control objective is to make the output y of the plant, starting at an initial condition $x_0 \in X_0$ under the reference input signal $r(\cdot)$, track the reference r while satisfying a property ϕ specified using STL. Let us now briefly recall STL. Examples of typical control specifications expressed in STL can be found in Sect. 4. An STL formula φ consists of atomic predicates along with logical and temporal connectives. Atomic predicates are defined over signal values and have the form $f(y(t)) \sim 0$, where f is a scalar-valued function over the signal y evaluated at time t, and $\sim \in \{<, \leq, >, \geq, =, \neq\}$. Temporal operators "always" ($\Box$), "eventually" ($\Diamond$), and "until" ($\mathcal{U}$) have the usual meaning and are scoped using intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, or (a, ∞), where $a, b \in \mathbb{R}_{\geq 0}$ and $a < b$. If I is a time interval, the following grammar defines the STL language.

$$\varphi \ := \ \top \mid f(y(t)) \sim 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 : \quad \sim \in \{<, \leq, >, \geq, =, \neq\} \tag{5}$$

The $\Diamond$ operator is formally defined as $\Diamond_I \varphi \triangleq \top \mathcal{U}_I \varphi$, and the $\Box$ operator is defined as $\Box_I \varphi \triangleq \neg(\Diamond_I \neg\varphi)$. When omitted, the interval I is taken to be $[0, \infty)$. The "always" operator in $\Box\varphi$ conveys that from the current time point on wards φ always holds. The "eventually" operator in $\Diamond\varphi$ means that there exists a time point in the future where φ holds. The "until" operator in $\varphi_1 \mathcal{U} \varphi_2$ means that, starting from the current time point φ_1 should hold continuously until a future time point where φ_2 holds. Additionally, an interval I can be combined with the operators to bound their scope to a segment of time in the future rather than the whole. For example, $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ holds true at time t if φ_2 holds true at some point $t' \in [t + a, t + b]$ and φ_1 is always true in $[t, t']$. Given a signal y and an STL formula φ, we use the quantitative semantics for STL, which is defined formally in [52]. The quantitative semantics defines a function ρ such that when $\rho(\varphi, y, t)$

is positive it indicates that (y, t) satisfies φ, and its absolute value estimates the *robustness* of this satisfaction. If φ is an inequality of the form $f(y) > b$, then its robustness is $\rho(\varphi, y, t) = f(y(t)) - b$. When t is omitted, we assume $t = 0$ (i.e., $\rho(\varphi, y) = \rho(\varphi, y, 0)$). For the conjunction of two formulas $\varphi := \varphi_1 \wedge \varphi_2$, we have $\rho(\varphi, y) = \min(\rho(\varphi_1, y), \rho(\varphi_2, y))$, while for the disjunction $\varphi := \varphi_1 \vee \varphi_2$, we have $\rho(\varphi, y) = \max(\rho(\varphi_1, y), \rho(\varphi_2, y))$. For negation, $\rho(\neg\varphi, y) = -\rho(\varphi, y)$. For a formula with until operator as $\varphi := \varphi_1 \mathcal{U}_I \varphi_2$, the robustness is computed as $\rho(\varphi, y, t) = \max_{t' \in I} \left(\min\left(\rho(\varphi_2, y, t'), \min_{t'' \in [t,t']} \left(\rho(\varphi_1, y, t'')\right)\right)\right)$.

Nominal Controller. We assume that we are given a nominal controller C that can produce a desired closed-loop behaviour (satisfying the control specification) for every control setting $s = (x_0, r(\cdot)) \in S$. When the plant is nonlinear, it is hard to find one single controller of a fixed structure (such as PID, MPC) that performs well for all control settings, and it is time-consuming to fine-tune the parameters to adapt to new scenarios. Our nominal controller can be thought of as a collection of local controllers, each of which is responsible for a subset of control settings. Here we can see the interest of using NN controllers as a means of combining controllers. In the learning context, the nominal controller plays the role of a teacher that generates a desired closed-loop behaviour for a given control setting that the NN should learn.

Neural Net Structure and Training. Intuitively, the NN controller is trained based on the desired closed-loop behaviours we want it to learn. Let $\gamma = (r(\cdot), x(\cdot), u(\cdot), y(\cdot))$ be a desired closed-loop behaviour, from which we extract the data of the form

$$d_\gamma = \left\{(r_k, y_k, u_k) \;\middle|\; k < K\right\} \tag{6}$$

where r_k, y_k and u_k are respectively the reference value, output value and control value at time t_k; K is the discrete time horizon (that is, the number of sampled time points). When many desired behaviours are considered, the data set $\mathcal{D}$ is the union of all d_γ. We generate a neural net $\mathcal{N}$ to match the data set $\mathcal{D}$. The structure of the NN should capture the input-output relationship of the controller in (3). We add past values to represent the memory needed to compute the output at each discrete step. The input of the neural net is $(r_k, \overline{y}_k, \overline{y}_{k-1}, \ldots, \overline{y}_{k-p}, \overline{u}_{k-1}, \ldots, \overline{u}_{k-q})$ and the output is $\overline{u}_k$. To train the NN, we use a loss function defined via the Mean Square Error (MSE): $1/N \sum_{k=1}^{N} (u_k - \overline{u}_k)^2$, where u is the output of the nominal controller and $\overline{u}$ is the output of the NN. A data point is a pair of input and output values, the total number N of data points is the number of data points per system behaviour multiplied by the number of behaviours. The NN accuracy is defined by the *training* and *validation errors*, that are obtained from the loss function.

In the following, we give the main steps of our algorithm for designing a NN controller by iteratively constructing a sequence of neural nets $\mathcal{N}_i$, training sets $\mathcal{D}_i$ and test sets $\mathcal{T}_i$ such that the NN controller passes the specification test with some level of coverage. Furthermore, under some assumptions about the plant dynamics and robustness of nominal controllers, it is possible to guarantee that the algorithm always terminates with a correct NN controller. This algorithm relies on a coverage based and cex based method for data generation that we will detail in the following.

3.1 Coverage Based Data Generation

In order to achieve a robust NN controller, we need to provide data representing diverse control settings that the NN should learn to cope with. Note that the set $\mathcal{S}$ of all possible control settings is generally infinite, we thus use a measure that quantifies how well a finite set of control settings covers the set $\mathcal{S}$. Considering the initial state x_0 as a constant signal, the set $\mathcal{S}$ can be treated as a continuous-time signal space. Let us now describe this measure, which is inspired by the notions of ε-entropy and ε-capacity used to characterise the complexity of a function set [53]. We consider a continuous-time signal space $\mathbf{\Sigma}$ where $\Delta \subset \mathbb{R}_+$ is the time domain of the signals in $\mathbf{\Sigma}$. To capture the distance between two signals $u, w \in \mathbf{\Sigma}$ we use the metric

$$\mu(u, w) = sup_{t \in \Delta} |u(t) - w(t)| \tag{7}$$

where $|\cdot|$ denotes the infinity norm. Let Σ be a set of signals in $\mathbf{\Sigma}$ that represents a class of signals of interest (such as reference signals, or expected temporal behaviours).

- We call a signal set Σ_n an *ε-net* of Σ if for each element $r \in \Sigma$, there exists $r' \in \Sigma_n$ such that $\mu(r, r') \leq \varepsilon$.
- A set Σ_s of signals in Σ is said *δ-separated* if for all elements $r, r' \in \Sigma_s$, $\mu(r, r') > \delta$.

Now for a given (possibly infinite) set Σ of signals in $\mathbf{\Sigma}$, a finite ε-net of Σ and a δ-separated subset of Σ can be combined to approximate the set Σ. An ε-net assures the "coverage" of the set Σ, but by definition it may contain functions which are not in Σ. An δ-separated set is guaranteed to be a subset of Σ and the separation requirement aims at limiting the cardinality of the approximate sets, which will be used later in Section 3.2 to select from a set of similar behaviours only a number of representative ones.

In [53], a method for constructing a minimal ε-net of a set of Lipschitz and smooth classes of functions is described. Here we are interested in the class of piece-wise constant signals which are suitable to represent reference signals in the control settings under consideration. We propose a simple grid-based method to construct ε-nets and sets satisfying some separation requirement. The set Σ of reference signals of our interest contains piece-wise constant signals with a fixed number of pieces m of equal duration. For simplicity of explanation we assume that the signals are one dimensional and omit the set X_0 of initial conditions (which can be considered, as mentioned earlier, as a constant signal). Let $[\underline{r}, \overline{r}]$ be the range of the reference signal. We use a grid $\mathcal{G}$ to partition the box $B_r = [\underline{r}, \overline{r}]^m$ into a set G of rectangular cells with equal side length 2ε (assuming for simplicity that $(\overline{r} - \underline{r})/(2\varepsilon)$ is integral). Each point p in this box B_r corresponds to a signal in Σ defined as: $r(t) = p_i$ for $t \in [t_i, t_i + \Delta/m)$ with $i = 0, 1, \ldots, (m-1)$ and $t_0 = 0$. In other words, the signal takes the i^{th} coordinate of p during the i^{th} time interval. We denote by β this mapping from a point $p \in B_r$ to $r \in \Sigma$, and β^{-1} the mapping in the opposite direction.

Proposition 1 *Let P be the set of center points of all the cells in G. The signal set $\Sigma_n = \{\beta(p) \mid p \in P\}$ is an ε-net of Σ.*

It is easy to see that any signal u in Σ corresponds to a point p_u in B_r. Let p_w be the center point of the grid cell that contains p_u. Thus, p_u is at distance (defined by the infinity norm) less than ε to p_w, which p_w in turn corresponds to a signal in Σ_n. It indeed can be proved that Σ_n is a ε-net with minimal cardinality and also a 2ε-separated set with maximal cardinality (this cardinality determines the ε-entropy and 2ε-capacity of Σ [53]). The data generation can be done using such an ε-net to guarantee a coverage of level at least ε. However, as we shall see later, to find cex, the falsification process uses optimization algorithms which can explore many control settings within a grid cell. To have a coverage measure that better reflects the portion of the tested control settings, we use a finer grid $\mathcal{G}_c$ and take the ratio between the number of cells visited by both the sampling and falsification procedures and the total number of cells in this grid $\mathcal{G}_c$.

3.2 NN Controller Validation and Counter-Example Guided Neural Net Retraining

Formal verification of the closed-loop system is generally difficult, as discussed in the introduction, since the composition of the neural net and the plant results in hybrid dynamics either with a large number of discrete modes caused by non-smooth activation functions or with stiff nonlinear continuous dynamics in high dimensions. Instead we try to quickly detect if the NN controller violates the expected closed-loop specification ϕ, using a falsification method to check $(\mathcal{N}||\Pi) \models \phi$. An outline of the methodology is shown in Figure 1.

A *counter-example* (*cex* for short) is a behaviour that does not satisfy the specification. Behaviors that do satisfy the specification are called *examples*. If no cex is found, we report the coverage of the set of explored control settings (see Section 3.1). Otherwise, let Ξ be the set of cex that are found. For each cex $\xi \in \Xi$, we extract training data as follows: We use the control setting s of ξ to simulate the corresponding nominal trajectory γ_s of $(\mathcal{C}||\Pi)$, which represents a desired behaviour from which we extract the data d_{γ_s} as in (6). Besides the data from the cex, we can add the data from a number of new examples which are sampled or explored during the falsification process. With the new data set $\mathcal{D}$, we retrain the neural net $\mathcal{N}$. To check whether the newly trained NN controller is satisfactory, we consider two types of checks: *matching quality* and *generalisation quality*. Concerning *matching quality*, we check if the neural net $\mathcal{N}$ has learnt well from the training data, meaning that for the control settings in the data set $\mathcal{D}$ that is used to train it, the closed loop $(\mathcal{N}||\Pi)$ satisfies ϕ. Concerning *generalisation quality*,we check if $\mathcal{N}$ can generalise well, meaning that for the control settings not in the data set $\mathcal{D}$, the closed loop $(\mathcal{N}||\Pi)$ still satisfies ϕ.

One iteration of the loop of retraining the NN controller is shown in Algorithm 1. Indeed, during the validation process, examples, that are trajectories satisfying the property, are generated and can be integrated in the data set for the next training. Let $\mathcal{N}_i$ be the neural net at the i^{th} iteration. The first test is a matching quality test on the control settings in the data set $\mathcal{D}_i$ used for training $\mathcal{N}_i$. This check

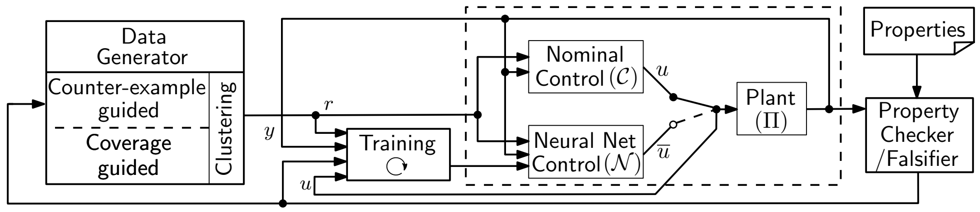


Figure 1: Counter-example Guided Neural Network Controller Retraining Methodology

```
Algorithm 1: Neural Net Retraining
Result: Retrain from $\mathcal{N}_i$ to obtain $\mathcal{N}_{i+1}$
    $Test(\mathcal{N}_i||\Pi, \mathcal{D}_i,\ \phi) \rightarrow \Xi_m$ /* matching test */
    $Test(\mathcal{N}_i||\Pi,\ \overline{\mathcal{D}}_i,\ \phi) \rightarrow \Xi_g \cup \Gamma$ /* generalisation test, $\overline{\mathcal{D}}_i$ is the complement of $\mathcal{D}_i$ */
    $\Xi = \Xi_m \cup \Xi_g$ /* collecting all the counter-examples */
    $\mathcal{D}_{i+1} = \mathcal{D}_i \cup Select_{cex}(\Xi) \cup Select_{ex}(\Gamma)$ /* update the data set */
    $\mathcal{N}_{i+1} = Training(\mathcal{N}_i, \mathcal{D}_{i+1})$ /* retraining with the updated data set */
```

can generate a set Ξ_m of *matching counter-examples*. The second check concerns generalisation quality, which can be defined as the ability of a trained NN to deal with situations which were not seen during training. Note that a trained NN may not generalize well because it is subject to over-fitting (that is, it essentially memorizes the training data, and indeed performs with high accuracy on the seen data but poorly on unseen data). To test the generalisation ability of a NN controller, in the $(i+1)^{th}$ iteration, we generate a set of control settings which are not in $\mathcal{D}_i$ and test the current NN controller under these settings. This test generates a set Γ of examples (that is satisfying trajectories) and a set Ξ_g of *generalisation counter-examples*. The procedures $Select_{cex}(\Xi)$ and $Select_{ex}(\Gamma)$ extract data from the sets of cex and examples so as to fulfil two objectives: exploitation (keeping similar cex for cex elimination) and diversification (keeping diverse examples for generalisation enhancement). To this end, we propose a behaviour clustering technique. This technique is based on a measure of distance between control settings: given two control settings $s = (x_0, r(\cdot))$ and $s' = (x'_0, r'(\cdot))$, $\mu(s, s') = |(x_0, \beta^{-1}(r)) - (x'_0, \beta^{-1}(r))|$. We use silhouette analysis to estimate first an optimal number of clusters which is then used to cluster the behaviours via the k-means algorithm. Then, to form a new data set, we select a subset of behaviours in each cluster, using two criteria: (1) the selected behaviours are separated by a given amount δ (for diversification purposes), (2) the robustness values of the selected behaviours are among K_ρ worst values (for exploitation purposes).

Proposition 2 *Let $\mathcal{N}$ be a neural net trained using the data set $\mathcal{D}$. Let $\mathcal{D}' = \mathcal{D} \cup d_{\gamma_s}$ where d_{γ_s} be data corresponding to a new desired closed-loop behaviour γ_s under a control setting s. Let $\mathcal{N}'$ be the new net trained using $\mathcal{D}'$ and γ'_s be the behaviour of $(\mathcal{N}'||\Pi)$ under the control setting s. If the training error of $\mathcal{N}'$ is bounded by ϵ, then there exists bounded η depending on ϵ such that $\mu(\gamma_s, \gamma'_s) \leq \eta$ where the distance metric μ is defined in* (7).

The proof of the proposition relies on the Lipschitz continuity of the plant dynamics and can be found in the Appendix. The above proposition shows that if we can train the neural net with arbitrary accuracy, then for a given control setting we can train the neural net to make the system follow a desired closed-loop behaviour as closely as we want. In other words, it is possible to eliminate a cex corresponding to a control setting s, we can choose for the same control setting a desired behaviour which is sufficiently robust. Training the neural net with the data from this desired behaviour allows eliminating the cex in question.

4 Implementation and Experimental Results

Given that our work is tailored towards control systems, we have implemented our code in MATLAB®/Simulink® which is widely used for such applications. Starting from a closed-loop system defined in Simulink and a set of formal specifications, we automatically perform coverage-based data generation, training via the MATLAB® Deep Learning Toolbox [54], falsification via the tool Breach [7], counter-example and example selection via clustering, and NN retraining. We have applied our methodology to two case studies, a water-tank level controller [55] and a robot-arm controller [56]. Both are closed-loop systems and the nominal controllers are discrete-time PID (one-degree of freedom PID and two-degree of freedom PID respectively). The watertank has 1 state

					# of counter-examples					Runtimes (sec)		
Model	ID	Clust.	Spec	i	n_T	n_C	$n_{\hat{C}}$	n_R	$n_{\tilde{C}}$	$t_{T,C,\hat{C}}$	t_R	$t_{\tilde{C}}$
Water-Tank	$W_{1,1}$ $[8^2]\langle 0.5\rangle$	No	$O \wedge St \wedge Se$	0	–	–	–	64	–	–	–	–
				1	300	262	**262**	324	**0**	469.75	**4275**	494.75
				2	176	0	–	–	–	285.94	–	–
	$W_{1,2}$ $[8^2]\langle 0.5\rangle$	Yes	$O \wedge St \wedge Se$	0	–	–	–	64	–	–	–	–
				1	300	262	**9**	73	**0**	469.75	**429**	504.35
				2	176	0	–	–	–	285.94	–	–
	$W_{1,3}$ $[8^2]\langle 0.5\rangle$	Random	$O \wedge St \wedge Se$	0	–	–	–	64	–	–	–	–
				1	300	262	**9** (rand)	73	**60**	482.71	996.74	498.66
				2	100	113*	–	–	–	183.55	–	–
	W_2 $[4^4]\langle 1\rangle$	Yes	$M \wedge St$	0	–	–	–	256				
				1	100	3	3	259	4	199.94	721.29	98.27
				2	100	8	4	263	0	195.41	4161.5	85.7367
				3	200	0	–	–	–	404.84	–	–
Robot-Arm	$A_{1,1}$ $[3^2]\langle 0.33\rangle$	Yes	O	0	–	–	–	9	–	–	–	–
				1	100	33	8	**17**	0	494.7	9.48	269.87
				2	200	0	–	–	–	956.79	–	–
	$A_{1,2}$ $[3^2]\langle 0.33\rangle$	Yes	St	0	–	–	–	9	–	–	–	–
				1	100	46	12	21	6	476.84	15.23	280.84
				2	100	36	9	**30**	0	487.12	13.77	277.38
				3	200	20	–	–	–	1029.5	–	–
	A_2 $[9^2]\langle 0.14\rangle$	No/Yes	$O \wedge St \wedge Se$	0	–	–	–	47	–	–	–	–
				1	100	1	1	48	0	535.88	21.40	263.60
				2	200	3	3	51	0	1068.2	33.54	283.49
				3	200	0	–	–	–	990.65	–	–

Table 1: Counter-example (cex) elimination through guided iterative re-training. ID: initial training instance of the neural network. $[l^m]\langle\delta\rangle$: Piece-wise constant reference with domain partitioned into l segments, with m time intervals, with minimum separation δ between reference points. i: retraining iteration. n_T: # of behaviours explored by the falsifier in the iteration. n_C: # of cex identified from n_T behaviours. $n_{\hat{C}}$: # of cex left after clustering (if clustering is used). n_R: # of behaviours for (re-)training. $n_{\tilde{C}}$: # of remaining cex after testing on the training data. t_{XYZ}: time for computing columns labelled n_{XYZ}. 9 (rand): 9 randomly generated traces – no clustering. *: 60 cex from previous iteration and 53 new cex.

variable, 1 output variable, 1 reference, and nonlinear dynamics (trigonometric). The robot-arm has 2 state variables, 1 output variable, 1 reference, and nonlinear dynamics (with a square root function).

In our experiments we use four properties: *Matching Error (M)*, *Stabilization (St)*, *Settling time (Se)* and *Overshoot (O)*, which are expressed using STL, where the notation $x[t]$ refers to the value of signal x at time instance t. The placeholder T_{sim} is the temporal length of simulated behaviours. The constant parameter dt is used to express a small amount of time.

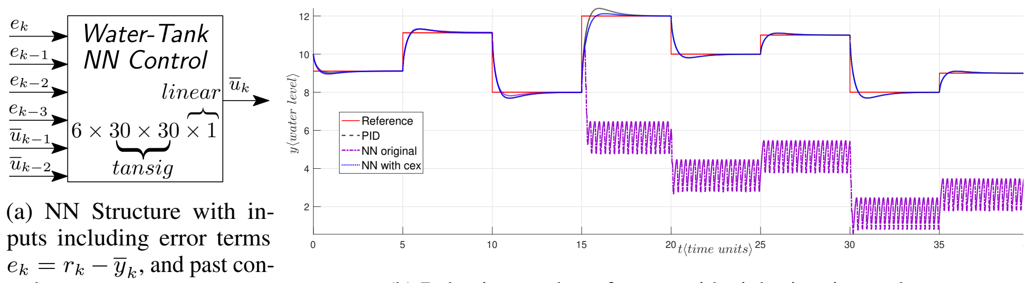
Property 1 ***Matching Error:*** *The closed-loop output of the plant, using the nominal controller y and that using the NN controller $\overline{y}$ do not differ by more than e:* $\Box_{[0,T_{sim}]}(|\overline{y}[t] - y[t]| < e)$.

Property 2 ***Stabilization:*** *Whenever there is a step change of more than ε_r in the reference r, then after time T_1 and within time T_2, the plant output $\overline{y}$ (under the NN controller) remains within $r \pm e$ for time T_3 to time T_4:* $\Box_{[0,T_{sim}]}((|r[t+dt] - r[t]| > \varepsilon_r) \Rightarrow (\Diamond_{[T_1,T_2]}\Box_{[T_3,T_4]}|\overline{y}[t] - r[t]| < e))$.

Property 3 ***Overshoot:*** *Whenever there is an increasing step change in the reference r, the plant output $\overline{y}$ should be at most $r + \varepsilon_y$ (or at least $r - \varepsilon_y$) for time T_1 to T_2 in the future:* $\Box_{[0,T_{sim}]}((|r[t+dt] - r[t]| > \varepsilon_r) \Rightarrow (\Box_{[T_1,T_2]}(\overline{y}[t] - r[t] < \varepsilon_y)))$

Property 4 ***Settling time:*** *Whenever there is a increasing (or decreasing) step change in the reference r, the plant output $\overline{y}$ should settle to $r \pm e$ within time T_1, and stay settled upto time T_2:* $(\Box_{[0,T_{sim}]}(|(r[t+dt] - r[t]| > \varepsilon_r) \Rightarrow (\Box_{[T_1,T_2]}(|\overline{y}[t] - r[t]| < e))))$

It is assumed that the nominal controller in each case study does not violate these properties, and our goal is to construct a NN controller which satisfies them. The structure of the NNs for both case studies is similar. Both nets have two hidden layers and the output layer has one output. The hidden layers have hyperbolic tangent activation functions and the output layers linear ones. Each hidden layer has 30 neurons. Note that the NNs differ in the number of inputs since the PID controllers have inputs of different dimensions. The water-tank takes as input the error $e_k = r_k - y_k$ which is one dimensional. The robot-arm controller has 2 input variables: r_k and y_k (used to compute setpoint weights). Fig 2a illustrates the NN structure used for the water-tank model.



(a) NN Structure with inputs including error terms $e_k = r_k - \overline{y}_k$, and past control outputs $\overline{u}_{k-1}$, $\overline{u}_{k-2}$.

(b) Behaviour under reference with eight time intervals.

Figure 2: Water-tank control for tracking a reference water level (red). Plant outputs using the PID controller (black), NN controller before (magenta) and after retraining (blue).

For each NN controller, we generate training data from the nominal controller using coverage metrics discussed in Section 3.1. Initial training, although coverage-based and exhaustive, may not be sufficient to guarantee a NN's adherence to specifications. The input stimuli for each case study are described by piecewise-constant functions with a varying number of setpoints. Then, we use our cex guided retraining approach to identify cex, conduct clustering to select the most representative ones (not necessarily the ones with the worst robustness), retrain the NN, and show that we are able to obtain a NN controller that does not have any closed-loop cex.

Figure 2b shows a cex behaviour (magenta) observed when using the original NN. Beyond the first three reference steps, the original NN is unable to effectively control the plant. This cex along with others are used for retraining. The result of applying our method is, a correct-by-specification, closed-loop response shown in blue.

Table 1 shows the results of re-training on the two examples we consider in this article. Note that we check several properties at the same time and eliminate the cex that correspond to each individual property. The experiments $W_{1,1}$,$W_{1,2}$ $W_{1,3}$ are run with the same configuration. The first round of falsification identifies 262 cex. By looking at $W_{1,1}$ and $W_{1,2}$, we observe the importance of clustering. In $W_{1,2}$ clustering selects only 9 cex, but they are enough to eliminate all 262 cex. Selecting all cex for retraining would also eliminate them but at the expense of a 10 times greater computational time. Comparing $W_{1,2}$ and $W_{1,3}$ highlights the importance of a guided-approach. Selecting 9 random traces from the allowed possible ones for retraining is not adequate to eliminate all 262 cex. Further, for $W_{1,3}$, in the next iteration we test with 100 new traces and we identify 50 new cex. In W_2 there are 4 setpoints necessitating the need for more traces to guarantee 100% cell occupancy. After 3 iterations, it is possible for the NN to eliminate all the cex.

The experiments $A_{1,1}$ and $A_{1,2}$ are trained using the same training set of piece-wise constant signals having two pieces, chosen from 3 values with a separation of 0.33. However, they differ with respect to the properties used for falsification. Experiment A_2 uses a training set with piece-wise constant behaviours having two pieces, with each piece chosen from 9 values separated by 0.14. Additionally, A_2 uses three properties for falsification. For A_2, the number of cex found during falsification is small, hence clustering has no impact on $n_{\hat{C}}$. $A_{1,1}$ and $A_{1,2}$ indicate our empirical finding that properties that the NN needs more training data to capture properties that relate to the steady state in comparison with properties that correspond to the transient behaviors. describe the steady state behavior typically need more training data

5 Conclusion

We introduced the problem of synthesizing a neural network controller for a nonlinear dynamical system where the specification is given as a set of STL formulas. We presented a counterexample-guided controller refinement algorithm to solve the synthesis problem. The algorithm is based on generating initial training data satisfying a coverage criterion and then subsequent retraining based on the training data generated from the counterexamples. Our experimental results demonstrate that the proposed technique is capable of synthesizing NN-controllers for non-trivial dynamical systems with a set of complex STL specifications. In the future, we plan to extend our framework to synthesize a neural network controller from several nominal controllers, each of which is optimal for a subset of the given specifications. We also plan to apply our technique to synthesize NN-controllers for complex dynamical systems like a quadrotor.

Broader Impact

Traditionally, deep neural networks have been applied to solving problems such as computer vision, speech recognition, natural language processing, etc., which are not safety-critical. However, deep neural networks have the potential to be applied to safety-critical cyber-physical systems (CPSs) in the form of a feedback controller. The potential applications include but not limited to autonomous vehicles, intelligent medical devices, smart grids, smart factories, etc. Unfortunately, despite their vast potential, AI-based controllers have not been widely adopted in the industry due to the lack of guarantees on their behavior. Our paper paves the way for synthesizing neural network feedback controllers for safety-critical systems with various reliability guarantees. We demonstrate that unlike the traditional controllers, it is possible to synthesize a neural network controller that ensures the satisfiability of a wide range of safety and performance requirements for cyber-physical systems.

Though our technique is capable of enhancing the reliability of AI-based CPSs, we should keep in mind that the controller synthesized by our method is not "correct" in the absolute sense. Our technique only ensures reliability with respect to a set of specifications. The system is not guaranteed to satisfy any other requirement that we do not consider at the time of synthesis. Nonetheless, our controller synthesis technology is expected to bring more confidence to AI-based safety-critical CPSs. We hope to see many novel safety-critical CPS applications that would not have been possible without AI-based solutions.

References

[1] M. T. Hagan, H. B. Demuth, and O. D. Jesús, "An introduction to the use of neural networks in control systems," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 12, no. 11, pp. 959–985, 2002.

[2] C. Nicol, C. J. B. Macnab, and A. Ramirez-Serrano, "Robust neural network control of a quadrotor helicopter," in *2008 Canadian Conference on Electrical and Computer Engineering*, pp. 1233–1238, 2008.

[3] P. Varshney, G. Nagar, and I. Saha, "Deepcontrol: Energy-efficient control of a quadrotor using a deep neural network," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 43–50, IEEE, 2019.

[4] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*, pp. 135–175, Springer, 2018.

[5] D. Nickovic and O. Maler, "AMT: A property-based monitoring tool for analog systems," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 304–319, Springer, 2007.

[6] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-Taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 254–257, Springer, 2011.

[7] A. Donzé, "Breach, A toolbox for verification and parameter synthesis of hybrid systems," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, vol. 6174 of *Lecture Notes in Computer Science*, pp. 167–170, Springer, 2010.

[8] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5189, 2017.

[9] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards generalization and simplicity in continuous control," in *Advances in Neural Information Processing Systems 30*, pp. 6550–6561, Curran Associates, Inc., 2017.

[10] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, "Approximating explicit model predictive control using constrained neural networks," in *Annual American Control Conference (ACC)*, pp. 1520–1527, 2018.

[11] C. S. Sánchez and D. Izzo, "Real-time optimal control via deep neural networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 5, pp. 1122–1135, 2018.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[13] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, “Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search,” in *International Conference on Robotics and Automation (ICRA)*, pp. 528–535, 2016.

[14] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[15] Y. Chow, O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh, “A Lyapunov-based approach to safe reinforcement learning,” in *NeurIPS*, pp. 8103–8112, 2018.

[16] A. Visioli, *Practical PID Control*. Advances in Industrial Control, Springer London, 2006.

[17] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[18] C. E. Garcia, D. M. Prett, and M. Morari, “Model predictive control: Theory and practice - a survey,” *Automatica*, vol. 25, no. 3, pp. 335 – 348, 1989.

[19] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, “Safety verification of deep neural networks,” in *International Conference on Computer Aided Verification*, pp. 3–29, Springer, 2017.

[20] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient SMT solver for verifying deep neural networks,” in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[21] R. Ehlers, “Formal verification of piece-wise linear feed-forward neural networks,” in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, vol. 10482 of *Lecture Notes in Computer Science*, pp. 269–286, Springer, 2017.

[22] A. Lomuscio and L. Maganti, “An approach to reachability analysis for feed-forward ReLU neural networks,” *CoRR*, vol. abs/1706.07351, 2017.

[23] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Output range analysis for deep feedforward neural networks,” in *NASA Formal Methods Symposium*, pp. 121–138, Springer, 2018.

[24] W. Xiang, H.-D. Tran, and T. T. Johnson, “Output reachable set estimation and verification for multilayer neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[25] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, “Efficient formal safety analysis of neural networks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6369–6379, 2018.

[26] W. Ruan, X. Huang, and M. Kwiatkowska, “Reachability analysis of deep neural networks with provable guarantees,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, p. 2651–2659, AAAI Press, 2018.

[27] W. Xiang, H. Tran, and T. T. Johnson, “Output reachable set estimation and verification for multilayer neural networks,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, 2018.

[28] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Output range analysis for deep feedforward neural networks,” in *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, vol. 10811 of *Lecture Notes in Computer Science*, pp. 121–138, Springer, 2018.

[29] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Case study: verifying the safety of an autonomous racing car with a neural network controller,” in *HSCC ’20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pp. 28:1–28:7, ACM, 2020.

[30] C. Cheng, G. Nührenberg, and H. Ruess, “Maximum resilience of artificial neural networks,” in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, vol. 10482 of *Lecture Notes in Computer Science*, pp. 251–268, Springer, 2017.

[31] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, “Fast and effective robustness certification,” in *Advances in Neural Information Processing Systems*, pp. 10802–10813, 2018.

[32] A. Raghunathan, J. Steinhardt, and P. Liang, “Semidefinite relaxations for certifying robustness to adversarial examples,” in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 10900–10910, 2018.

[33] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pp. 3–18, IEEE Computer Society, 2018.

[34] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[35] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *CoRR*, vol. abs/1903.01287, 2019.

[36] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Learning and verification of feedback control systems using feedforward neural networks," in *6th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2018, Oxford, UK, July 11-13, 2018*, vol. 51 of *IFAC-PapersOnLine*, pp. 151–156, Elsevier, 2018.

[37] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 169–178, 2019.

[38] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 157–168, 2019.

[39] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, "ReachNN: Reachability analysis of neural-network controlled systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 106:1–106:22, 2019.

[40] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*, pp. 379–395, Springer, 2011.

[41] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, vol. 8044 of *Lecture Notes in Computer Science*, pp. 258–263, Springer, 2013.

[42] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[43] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, J. Kapinski, and X. Jin, "Falsification of safety properties for closed loop control systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015* (A. Girard and S. Sankaranarayanan, eds.), pp. 299–300, ACM, 2015.

[44] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 303–314, ACM, 2018.

[45] S. Yaghoubi and G. Fainekos, "Gray-box adversarial testing for control systems with machine learning components," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pp. 179–184, ACM, 2019.

[46] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *J. Autom. Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[47] J. Ferlez and Y. Shoukry, "AReN: assured ReLU NN architecture for model predictive control of LTI systems," in *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pp. 6:1–6:11, ACM, 2020.

[48] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *NIPS*, pp. 908–918, 2017.

[49] T. A. Henzinger, R. Jhala, and R. Majumdar, "Counterexample-guided control," in *International Colloquium on Automata, Languages, and Programming*, pp. 886–902, Springer, 2003.

[50] Y. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 3240–3249, 2019.

[51] A. Clavière, S. Dutta, and S. Sankaranarayanan, “Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks,” in *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pp. 680–688, AAAI Press, 2019.

[52] A. Donzé and O. Maler, “Robust satisfaction of temporal logic over real-valued signals,” in *Formal Modeling and Analysis of Timed Systems* (K. Chatterjee and T. A. Henzinger, eds.), (Berlin, Heidelberg), pp. 92–106, Springer Berlin Heidelberg, 2010.

[53] A. N. Kolmogorov and V. M. Tikhomirov, “ε-entropy and ε-capacity of sets in function spaces,” *Uspekhi Matematicheskikh Nauk*, vol. 14, no. 2, pp. 3–86, 1959.

[54] Mathworks, “Deep Learning Toolbox - Design, train, and analyze deep neural networks.” [Online] Available at `https://www.mathworks.com/products/deep-learning.html`.

[55] Mathworks, “`watertank` Simulink Model.” [Online] Available at `https://fr.mathworks.com/help/slcontrol/gs/watertank-simulink-model.html`.

[56] Mathworks, “Design Model-Reference Neural Controller in Simulink.” [Online] Available at `https://fr.mathworks.com/help/deeplearning/ug/design-model-reference-neural-controller-in-simulink.html`.