

Formal Verification of Neural Certificates Done Dynamically

Thomas A. Henzinger[✉], Konstantin Kueffner[✉], and Emily Yu[✉]

Institute of Science and Technology Austria
{tah, kkueffner, zyu}@ist.ac.at

Abstract. Neural certificates have emerged as a powerful tool in cyber-physical systems control, providing witnesses of correctness. These certificates, such as barrier functions, often learned alongside control policies, once verified, serve as mathematical proofs of system safety. However, traditional formal verification of their defining conditions typically faces scalability challenges due to exhaustive state-space exploration. To address this challenge, we propose a lightweight runtime monitoring framework that integrates real-time verification and does not require access to the underlying control policy. Our monitor observes the system during deployment and performs on-the-fly verification of the certificate over a lookahead region to ensure safety within a finite prediction horizon. We instantiate this framework for ReLU-based control barrier functions and demonstrate its practical effectiveness in a case study. Our approach enables timely detection of safety violations and incorrect certificates with minimal overhead, providing an effective but lightweight alternative to the static verification of the certificates.

Keywords: Runtime Monitoring · Neural Control · Neural Certificates.

1 Introduction

Forthcoming intelligent systems are increasingly integrated into industries and numerous application domains, including autonomous driving and medical image processing [2,4]. For example, deep reinforcement learning enables the automated synthesis of neural network controllers to address complex control tasks [16]. However, in these safety-critical domains, ensuring the safety and correctness of machine-learned components presents significant challenges. Neural networks often lack transparency and explainability, undermining their trustworthiness. Without formal safety guarantees, the reliability and operation of cyber-physical systems remain constrained, thereby affecting their deployment in practice.

Certificate functions serve as mathematical proofs to establish the correctness of controllers. A certificate function [9] is a mathematical mapping from system states to real values, where the satisfaction of its defining conditions guarantees that a desired system property holds. Notable examples include Lyapunov functions [28], used to prove stability with respect to a fixed point, and barrier functions [22], which are employed to certify safety by characterizing

forward-invariant sets. While the theory of Lyapunov functions has been extensively studied over the past several decades, recent advances leverage reinforcement learning to synthesize such certificates in the form of neural networks. A growing body of work in learning-based control explores the joint synthesis of both the certificate function and the control policy. We refer to [9] for a more comprehensive overview of learning-based control using certificates.

Since certificate functions are often represented as neural networks, formal verification is typically employed to ensure their correctness. This gave rise to the learner-verifier framework [6,7,20], which is a synthesis framework to obtain valid neural certificates. This framework follows the style of Counterexample-Guided Inductive Synthesis (CEGIS). It consists of two main components: *the learner*, which synthesizes both the control policy and the certificate function, and *the verifier*, which checks whether the certificate satisfies its formal defining conditions. If the verifier identifies a counterexample, it is incorporated into the training set to refine the neural networks. This iterative loop continues until the certificate is successfully verified or a predefined termination criterion, such as a timeout, is met.

While formal verification offers correctness guarantees, it often suffers from scalability limitations due to its computational complexity, restricting its use in more complex control tasks. To overcome this challenge, *runtime monitoring* has emerged as a promising complementary approach. A runtime monitor is a lightweight software module that operates in parallel to the controlled system, issuing warnings upon detecting violations of specified properties or certificate defining conditions. The Simplex architecture [8,27], for instance, employs a verified backup controller to override unsafe actions. Recent work has proposed a learner-monitor framework [32], analogous to the learner-verifier paradigm, in which the monitor continuously collects counterexamples during execution and incorporates them into the training data to refine the learned models.

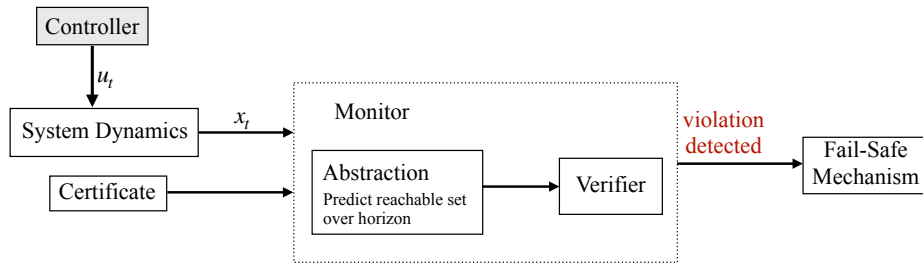


Fig. 1. Overview of our monitoring framework. At runtime, the monitor observes system states and computes an overapproximation of the reachable region using a local abstraction function. We do not assume access to the controller. A verifier checks whether the certificate remains valid in the lookahead region. If a violation is detected, a warning is issued and a fail-safe mechanism can be triggered.

In this paper, we propose a general runtime monitoring framework that integrates partial static verification over a lookahead horizon to ensure the safety of the controlled systems, as illustrated in Fig. 1. The framework is agnostic to the underlying controller and does not require access to future control inputs. At each time step, a monitor observes the system trajectory and uses a local abstraction function to over-approximate reachable states within the lookahead horizon. A certificate verifier then checks whether the safety certificate remains valid over this predicted region. If a violation is detected, the monitor issues a warning and can trigger a fail-safe response. Because our monitor detects violations ahead of time, the fail-safe mechanism is activated *before* leaving the certified region.

We instantiate this general framework using a specific verifier for ReLU-based control barrier functions (CBFs), leveraging the piecewise linear structure of ReLU networks for efficient localized verification. Our case study on a satellite rendezvous task demonstrate that the monitor can detect certificate violations as well as ensuring runtime safety with minimal overhead.

Overall, our contributions are as follows.

1. We introduce a general framework for online verification of certificates, which leverages a local abstraction function and a verifier as a sub-routine to enable local verification over a finite lookahead horizon.
2. We instantiate the framework for ReLU-based control barrier functions, and present a novel online verification algorithm that operates over neuron activation patterns. The algorithm adaptively constructs and verifies regions of the state space encountered at runtime, enabling fast detection of violations of safety as well as certificate defining conditions.

To the best of our knowledge, this is the first runtime monitoring approach that performs formal, on-the-fly certificate verification for neural-based control.

2 Preliminaries

We denote the set of natural number as $\mathbb{N}$ and the set of real numbers as $\mathbb{R}$. Their positive counterparts are denoted as $\mathbb{N}_+$ and $\mathbb{R}_+$ respectively.

Given a state space $\mathcal{X} \subseteq \mathbb{R}^m$ and an action space $\mathcal{U} \subseteq \mathbb{R}^n$, we consider continuous-time, control-affine dynamical systems of the form

$$\dot{x} = f(x) + g(x)u,$$

where $x \in \mathcal{X}, u \in \mathcal{U}, f : \mathbb{R}^m \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^m \rightarrow \mathbb{R}^{m \times n}$. A control policy $\pi : \mathcal{X} \rightarrow \mathcal{U}$ defines a closed-loop system $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{X}$ with dynamics

$$\mathcal{F}(x) = f(x) + g(x)\pi(x). \quad (1)$$

The safe control problem requires that, under a given control policy, the system must never enter an unsafe state along any execution trace originating from an initial state. To prove that the process $\mathcal{F}$ satisfies properties such as

safety or reachability, we rely on certificates. A certificate is a function $B: \mathcal{X} \rightarrow \mathbb{R}$ assigning a numerical value to each state. To be a valid certificate, this function B must satisfy some property dependent conditions over the state space. To prove safety, a barrier function is needed.

Definition 1 (Control Barrier Function). *Let $\mathcal{X}_0 \subseteq \mathcal{X}$ be the set of initial states, and let $\mathcal{X}_u \subseteq \mathcal{X}$ be the unsafe states. A continuously differentiable function $B: \mathcal{X} \rightarrow \mathbb{R}$ is called a control barrier function if the following conditions hold:*

1. $B(x) \geq 0$ for all $x \in \mathcal{X}_0$;
2. $B(x) < 0$ for all $x \in \mathcal{X}_u$;
3. $\nabla B(x)(f(x) + g(x)u) + \alpha(B(x)) \geq 0$ for all $x \in \{x \in \mathcal{X} \mid B(x) \geq 0\}$.

Here, $\alpha: \mathbb{R} \rightarrow \mathbb{R}$ is a class $\mathcal{K}$ function that is strictly increasing and $\alpha(0) = 0$ [3].

The set $\mathcal{C} = \{x \in \mathcal{X} \mid B(x) \geq 0\}$ is referred to as the forward invariant set. The expression $\nabla B(x)(f(x) + g(x)u)$ denotes the Lie derivative of the barrier function B along the system dynamics. If a valid barrier function exists for a given control policy π , then the system is guaranteed to be safe. A formal proof of this result is provided in [3].

In this paper, we also consider ReLU-based barrier functions, where the barrier function is implemented as a feed-forward neural network with L layers. Each layer i has dimension size M_i , and the activation function used is the Rectified Linear Unit (ReLU), defined as $\sigma: \mathbb{R}^a \rightarrow \mathbb{R}^a$ with

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Each neuron in the network is indexed by its layer and position within the layer, denoted by the pair (i, j) . Given an input x , let z_i^j denote the pre-activation value at neuron (i, j) . A neuron is said to be *active* if $z_i^j > 0$, *unstable* if $z_i^j = 0$, and *inactive* otherwise. We denote by $\mathcal{A} = (A_1, \dots, A_L)$ an activation pattern, where each A_i represents the set of active neurons in layer i . Similarly, let $\mathcal{T} = (T_1, \dots, T_L)$ represent the unstable neurons.

For a given activation pattern, evaluating the neural network becomes significantly simpler, as the weights and biases reduce to fixed linear terms. We define masked weights and biases with respect to $\mathcal{A}$. For the first layer:

$$\overline{W}_{1j}(\mathcal{A}) = \begin{cases} W_{1j} & \text{if } j \in A_1, \\ 0 & \text{otherwise,} \end{cases} \quad \overline{r}_{1j}(\mathcal{A}) = \begin{cases} r_{1j} & \text{if } j \in A_1, \\ 0 & \text{otherwise.} \end{cases}$$

This implies that the output of the j -th neuron in the first layer is simply $\overline{W}_{1j}(\mathcal{A})^\top x + \overline{r}_{1j}(\mathcal{A})$. For deeper layers $i > 1$, we recursively define:

$$\overline{W}_{ij}(\mathcal{A}) = \begin{cases} W_{ij}^\top \overline{W}_{i-1}(\mathcal{A}) & \text{if } j \in A_i, \\ 0 & \text{otherwise,} \end{cases} \quad \overline{r}_{ij}(\mathcal{A}) = \begin{cases} W_{ij}^\top \overline{r}_{i-1}(\mathcal{A}) + r_{ij} & \text{if } j \in A_i, \\ 0 & \text{otherwise.} \end{cases}$$

We write $\overline{W}(\mathcal{A})$ to denote the effective linear transformation of the final layer under the activation pattern $\mathcal{A}$. This linearized structure captures the piecewise-linear behavior of the ReLU network within the region defined by $\mathcal{A}$.

Definition 2 (ReLU-based CBF). *Let $\mathcal{X}(\mathcal{A})$ denote the region of the input space induced by activation pattern $\mathcal{A}$. A function B is a ReLU-based control barrier function if, for every activation pattern $\mathcal{A}$, the region $\mathcal{X}(\mathcal{A}) \cap \mathcal{C}$ does not intersect the unsafe set:*

$$\mathcal{X}(\mathcal{A}) \cap \mathcal{C} \cap \mathcal{X}_{\text{unsafe}} = \emptyset.$$

Moreover, there exists an activation pattern $\mathcal{A}$ such that for all x on the boundary where $B(x) = 0$, the following hold:

1. $(\overline{W}_{i-1}(\mathcal{A})W_{ij})^\top (f(x) + g(x)\pi(x)) \geq 0$ for all $(i, j) \in \mathcal{T}(x) \cap \mathcal{A}$;
2. $(\overline{W}_{i-1}(\mathcal{A})W_{ij})^\top (f(x) + g(x)\pi(x)) \leq 0$ for all $(i, j) \in \mathcal{T}(x) \setminus \mathcal{A}$;
3. $\overline{W}(\mathcal{A})^\top (f(x) + g(x)\pi(x)) \geq 0$.

This definition encodes the requirement that the barrier function B does not decrease along the system dynamics at the certificate boundary where $B(x) = 0$ [34]. This definition guarantees that B serves as a valid control barrier function over the region defined by the activation pattern $\mathcal{S}$, and that the system remains within the forward invariant set $\mathcal{C}$ if it starts there.

3 Monitoring Framework

Classically the validity of a certificate function, e.g., the barrier certificate in Definition 1, is verified on the entire state space. This is expensive and wasteful, especially in high-dimensions. As the dimension of the state space increases, both, finding and validating a certificate becomes exponentially more expensive. At the same time the volume of all states visited by the process on an infinite run becomes negligible compared to the volume of the entire domain, implying that most of the work done during verification is unnecessary.

Certificates at Runtime. We use φ to denote an arbitrary certificate condition, e.g., the conditions in Definition 1. The binary value of a certificate condition is determined by the closed-loop system $\mathcal{F}$, the certificate B , and a subset of the state space $\mathcal{Y} \subseteq \mathcal{X}$, i.e., $\varphi(\mathcal{F}, B, \mathcal{Y}) \in \{0, 1\}$. In static verification we check the certificate condition over the entire state space $\mathcal{X}$. This ensures that φ is satisfied on the infinite trace $w := (x_t)_{t \in \mathbb{R}_+}$ generated by $\mathcal{F}$. Intuitively, we would expect that validating the certificate condition for a single trace, i.e., $\varphi(\mathcal{F}, B, w)$, should be significantly simpler than validating it for the entire state space $\varphi(\mathcal{F}, B, \mathcal{X})$. The reason why static verification focuses on the entire state space is because it may be impossible to obtain the trace w at development time due to uncertainty. However, for short time horizons this uncertainty remains manageable. This is where static verification done dynamically shines.

Example 1. Consider $\mathcal{X} := [0, 1]^d$ for $d \in \mathbb{N}_+$ and let $w := (x_t)_{t \in \mathbb{R}_+}$. If we know that the certificate B and the system $\mathcal{F}$ are Lipschitz with a constant L , a naive approach to validating the certificate condition on $\mathcal{X}$ up to a precision $c > 0$ is to construct a c/L -grid [7]. The number of vertices in the grid increases exponentially in the dimension, i.e., for $\mathcal{X}$ we require approximately $(L/c)^d$ vertices. By contrast, a c/L -grid over w requires only L/c vertices. Hence, verifying the trace is exponentially faster than verifying the entire domain.

Certificate Monitor. Our objective is to construct a certificate monitor which detects certificate condition violations before they occur. Let $w := (x_t)_{t \in \mathbb{R}_+}$ be the trace generated by $\mathcal{F}$. The trace up to time $t \in \mathbb{R}_+$ is defined as $w_{[0,t)} := (x_t)_{t \in [0,t)}$. The monitor can only observe a finite subset of this trace as given by its observation frequency parameter $\varepsilon > 0$, which we consider as given, e.g., the parameter may be a function of the monitor's hardware constraints. For every interval $[a, b) \subseteq \mathbb{R}$ we denote $[a, b)_\varepsilon := \{a + k \cdot \varepsilon \mid k \in \mathbb{N} \wedge a + k \cdot \varepsilon < b\}$ as the finite discretization of $[a, b)$ w.r.t. ε . We denote the observable trace by the monitor as $w_{[0,t)}^\varepsilon := (x_t)_{t \in [0,t)_\varepsilon}$. For a given safety horizon $h \in \mathbb{R}_+$, e.g., the time required to stop a vehicle, we want a monitor $\mathbf{M}: \mathcal{X}^* \rightarrow \{0, 1\}$ that maps the trace $w_{[0,t)}^\varepsilon$ into a verdict in $\{0, 1\}$. We call the monitor $\mathbf{M}$ sound, iff a positive verdict guarantees that the certificate is satisfied on the trace $w_{[0,t+h)}$, i.e.,

$$\mathbf{M}(w_{[0,t)}^\varepsilon) = 1 \implies \varphi(\mathcal{F}, B, w_{[0,t+h)}) = 1. \quad (2)$$

Problem 1. For a given closed-loop system $\mathcal{F}$ as defined in Equation 1, a certificate function B , a certificate condition φ , and a safety horizon h with epsilon ε , construct a sound certificate monitor $\mathbf{M}$.

3.1 Monitor Construction

We show how to construct a sound certificate monitor using a local abstraction function and a certificate verifier as subroutines. The local abstraction function provides a sound overapproximation of how the process behaves within a given time horizon. This overapproximation is a bounded region, e.g., a cone, within the state space. The verifier checks whether the certificate condition is satisfied for every state within a given region of the state space. Our monitor simply invokes the verifier on the uncertainty region computed by the local abstraction function. We chose this modular construction because there exists a plethora of predictive monitoring tools which can be utilized as a local abstraction function [14]. Similarly, there exists a variety of certificate verification tools that can assess the validity of a certificate for a given region [7,35,36]. Each tool has its own assumptions on the system, as a consequence our certificate monitor can adapt and improve depending on the available subroutines.

Abstraction and verification. We assume access to a local abstraction function $\mathbf{A}: \mathcal{X}^* \times \mathbb{R}_+ \rightarrow \mathcal{P}(\mathcal{X})$ and a verifier $\mathbf{V}: \mathcal{P}(\mathcal{X}) \rightarrow \{0, 1\}$ where $\mathcal{P}(\mathcal{X}) := \{\mathcal{Y} \mid \mathcal{Y} \subseteq \mathcal{X}\}$ is the set of all subsets of $\mathcal{X}$. First, the abstraction function maps the trace $w_{[0,t]}^\varepsilon \in \mathcal{X}^*$ up to time $t \in \mathbb{R}_+$ and a time horizon $h \in \mathbb{R}_+$ into a subset of the domain, i.e., $\mathbf{A}(w_{[0,t]}^\varepsilon, h) \subseteq \mathcal{X}$. We are guaranteed that the process is contained within the output region for the specified time horizon, i.e.,

$$\forall t \in [t, t+h). x_t \in \mathbf{A}(w_{[0,t]}^\varepsilon, h). \quad (3)$$

Second, the verifier checks whether the certificate B satisfies condition φ w.r.t. the process $\mathcal{F}$ for a given region $\mathcal{Y} \subseteq \mathcal{X}$, i.e.,

$$\forall \mathcal{Y} \subseteq \mathcal{X}. \mathbf{V}(\mathcal{Y}) = 1 \iff \varphi(\mathcal{F}, B, \mathcal{Y}) = 1 \quad (4)$$

In Section 4 we provide a concrete examples for both the abstraction function and the verifier. For now we treat both of them as black-boxes.

Local Verification. By combining the abstraction function and the verifier we construct a simple subroutine ensuring that the certificate condition is satisfied locally, i.e., the certificate remains valid within the immediate time horizon. Assume we are at time $t \in \mathbb{R}_+$, have observed the trace $w_{[0,t]}^\varepsilon$ derived from the actual trace $w_{[0,t]}$, and decided for a time horizon $h_t \in \mathbb{R}_+$. We invoke the abstraction function $\mathbf{A}$ with the observed trace $w_{[0,t]}^\varepsilon$ and time horizon h_t , and verify the generated region using $\mathbf{V}$ to guarantee the validity of the certificate up until time $t + h_t$, i.e., we compute

$$\mathbf{V}(\mathbf{A}(w_{[0,t]}^\varepsilon, h_t)). \quad (5)$$

If the verifier outputs 1 the validity for certificate is assured up until $t + h$, if the verifier outputs 0 the certificate condition is violated for a state in the predicted region. Hence, it may violated on $w_{[t,t+h)} \subseteq \mathbf{A}(w_{[0,t]}^\varepsilon, h_t)$. This gives a local overapproximation of the certificate condition value.

Lemma 1. *If the abstraction function satisfies Condition 3 and the verifier satisfies Condition 4, we are guaranteed that*

$$\mathbf{V}(\mathbf{A}(w_{[0,t]}^\varepsilon, h_t)) = 1 \implies \varphi(\mathcal{F}, B, w_{[t,t+h_t)}) = 1.$$

Proof. Condition 3 ensures that the over-approximation computed by the abstraction function contains the trace up to time $t + h_t$, i.e., $w_{[t,t+h_t)} \subseteq \mathcal{Z} := \mathbf{A}(w_{[0,t]}^\varepsilon, h_t)$. Condition 4 ensures that $\mathbf{V}(\mathcal{Y}) = 1 \implies \varphi(\mathcal{F}, B, w_{[t,t+h_t)}) = 1$.

Soundness. The subroutine described above guarantees that the certificate condition remains satisfied within the specified time horizon. However, verifying on the fly requires time and cannot be done continuously. Therefore, constructing a sound monitor requires us to get the timing right. Assume we can start the verification procedure after having observed the current state. If we are able to compute the local guarantee, i.e., $\mathbf{V}(\mathbf{A}(w_{[0,t]}^\varepsilon, h_t))$, for a time horizon of

$h_t := 2 \cdot \varepsilon + h$ in less than ε -time, then we can obtain a sound monitor defined in Algorithm 1. This guarantees that the monitor raises a warning *before* the system enters a region where the certificate is invalid, allowing the system to activate the fail-safe mechanism in time.

Theorem 1. *Given a state space $\mathcal{X}$, a system $\mathcal{F}$, and a certificate B . Let $\mathbf{A}: \mathcal{X}^* \times \mathbb{R}_+ \rightarrow \mathcal{P}(\mathcal{X})$ be an abstraction function (satisfying Eq. 3, $\mathbf{V}: \mathcal{P}(\mathcal{X}) \rightarrow \{0, 1\}$ be a verifier (satisfying Eq. 4), $h \in \mathbb{R}_+$ be a time horizon, $\varepsilon \in \mathbb{R}_+$ be a observation frequency parameter. If the sub-routine NEXT can be computed in less than ε -time, the monitor defined by Algorithm 1 is sound, i.e.,*

$$\mathbf{M}_h(w_{[0,t]}^\varepsilon) = 1 \implies \varphi(\mathcal{F}, B, w_{[0,t+h]}) = 1.$$

Proof. The sequence of observation points is $(k \cdot \varepsilon)_{k \in \mathbb{N}}$ we prove this claim by induction over the observation points. Let $k = 0$. We assume we have already verified the first $\varepsilon + h$ time interval, if the verdict $v = 0$ we have found a potential violation and we are done. Otherwise, we know from Lemma 1 that this implies that $\varphi(\mathcal{F}, B, w_{[0,\varepsilon+h]}) = 1$. This implies that every point $s \in [0, \varepsilon)$ is covered. The induction hypothesis is that at the beginning of every observation point $k \in \mathbb{N}_+$, $v_{k-1} = 1$ implies that $\varphi(\mathcal{F}, B, w_{[0,\varepsilon \cdot k+h]}) = 1$. Assume we are at observation point $k + 1$. If $v_k = 0$ we are done. If $v_k = 1$, then by the induction hypothesis we know that $\varphi(\mathcal{F}, B, w_{[0,\varepsilon \cdot (k+1)+h]}) = 1$. We start computing NEXT for the time horizon $2\varepsilon + h$, i.e. upon termination we know whether there exists a potential certificate violation on $[(k+1) \cdot \varepsilon, (k+3) \cdot \varepsilon + h)$ or not. By assumption we know that executing NEXT requires less than ε time. Hence, before time step $k + 2$ we have finished computing v_{k+1} , which if $v_{k+1}=1$ guarantees that $\varphi(\mathcal{F}, B, w_{[0,\varepsilon \cdot (k+2)+h]}) = 1$.

Implementation. Algorithm 1 describes the schematic dynamic verification routine. We initialize the monitor with INIT, which creates the initial empty trace w_0 and sets the initial verdict v_0 to 1. During runtime the monitor executes NEXT whenever a new input state $x \in \mathcal{X}$ is observed. At execution $k \in \mathbb{N}_+$, it appends the observed state x to the past trace creating w_k , which is used to perform the verification step over the uncertainty region provided by the abstraction function; checking whether there is a certificate violation in the future. The verdict of this step is stored in v_{buffer} . If the previous verdict v_{k-1} is 1, i.e., no certificate violation was detected in some previous iteration, the verdict at iteration v_k is set to v_{buffer} . Otherwise, the verdict remains 0, because once a violation is detected, the monitor will consider the certificate condition violated.

Remark 1. If the execution of NEXT requires more time, parallelization can be exploited to avoid this problem, i.e., we simply execute NEXT while accounting for the additional computation time on a new thread at the end of every control interval. If the computation time remains constant, we can obtain a similar guarantee. Moreover, the number of threads can be bounded based on that time.

Algorithm 1 Schematic Monitor M_h

```

1: Given: time horizon  $h \in \mathbb{R}_+$ , observation frequency parameter  $\varepsilon$ , abstraction function  $\mathbf{A}: \mathcal{X}^* \times \mathbb{R}_+ \rightarrow \mathcal{P}(\mathcal{X})$ , a verifier  $\mathbf{V}: \mathcal{P}(\mathcal{X}) \rightarrow \{0, 1\}$ .
2: function INIT()
3:    $w_0 \leftarrow x$ ;  $v_0 \leftarrow 1$ 
4: function NEXT( $x$ )
5:    $w_k \leftarrow w_{k-1} \cdot x$ 
6:    $v_{\text{buffer}} \leftarrow \mathbf{V}(\mathbf{A}(w_k, 2 \cdot \varepsilon + h))$ 
7:    $v_k \leftarrow v_{\text{buffer}}$  if  $v_{k-1} = 1$  else  $v_k \leftarrow 0$ 
8:   return  $v_k$ 
    
```

4 Online Verification of ReLU-based Control Barrier Functions (CBFs)

Our monitoring approach is designed to ensure runtime safety and verify the correctness of a neural certificate, without requiring full state space verification or access to future control inputs. In this section, we instantiate the general monitoring framework for ReLU-based control barrier functions.

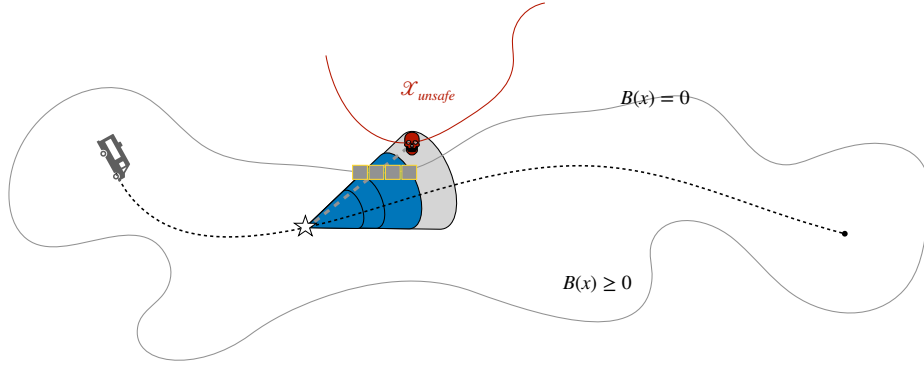


Fig. 2. Illustration of online verification with lookahead. At each time step, the monitor maintains a cone that over-approximates the reachable states within a fixed horizon. If this cone intersects the unsafe region, the monitor searches for a cube on the barrier boundary where $B(x) = 0$. The cone is then shrunk to contain this boundary, and the identified cube along with its neighbors are verified to assess certificate validity.

As illustrated in Figure 2, the monitor continuously observes the system's trajectory and incrementally builds a conservative over-approximation of reachable states within a fixed lookahead horizon. If no violations are detected within this cone region, the system is deemed safe for the near future. If a *safety violation* is detected, e.g., the cone intersects the unsafe region, the monitor performs a refinement step. It searches for a state on the certificate boundary ($B(x) = 0$)

and identifies the corresponding *activation region* (called a *cube*) in which this state lies. Because ReLU networks induce a partition of the state space into finitely many such cubes, and each cube corresponds to a fixed neuron activation pattern, the CBF condition from Def. 1 becomes a set of linear constraints within each cube. The monitor then verifies these conditions locally by checking whether the barrier certificate holds in the identified cube and its neighbors. If a violation is detected, the system triggers a fail-safe response (e.g., controller override). If all cubes are verified to be safe, the monitor continues observing and updating the cone over time.

Our approach operates over two state spaces: (1) the continuous, real-valued state space defined by the underlying dynamical system, and (2) the discrete space of activation patterns, induced by the architecture of the neural CBF. In the following, we refer to an element of (1) simply as a *state*, and to the subset of states corresponding to a particular activation pattern as a *cube*. Two states belong to the same cube if they produce identical neuron activation patterns in the CBF. We present the main monitoring loop in Algorithm 2.

Initialization. Our approach begins with a pre-trained neural CBF and an associated control policy. The control policy is treated as a black-box, meaning its internal structure is not accessible. However, we assume access to a set of known constraints that bound the range of control inputs generated by the policy, e.g., minimum and maximum allowable acceleration values. In addition, we define a fixed horizon h parameter, which determines how far into the future the monitor evaluates the system’s evolution to assess safety.

Algorithm 2 Main Monitor Loop for ReLU-based CBFs

```

1: Given: forward invariant set  $\mathcal{C}$ , unsafe set  $\mathcal{X}_u$ , lookahead horizon  $h$ 
2: function INIT()
3:    $v_0 \leftarrow 1$ ;  $\mathcal{X}_{\text{safe}} \leftarrow \mathcal{X} \setminus \mathcal{X}_u$ 
4: function NEXT( $x$ )
5:    $(\text{cone}, x_{\text{unsafe}}, i) \leftarrow \text{CONSTRUCTCONE}(\mathcal{X}_u, x, h)$  // Abstraction function call
6:   if  $x_{\text{unsafe}}$  is found then
7:      $v_{\text{buffer}} \leftarrow \text{VERIFYCUBESONBOUNDARY}(x, x_{\text{unsafe}}, \text{cone}, i, h)$  // Verifier call
8:      $v_k \leftarrow v_{\text{buffer}}$  if  $v_{k-1} = 1$  else  $v_k \leftarrow 0$ 
9:   return  $v_k$ 

```

During monitoring, the cone is constructed using an abstraction function. To detect potential safety violations, the monitor expands the cone using the CONSTRUCTCONE procedure (Algorithm 3). If no intersection with the unsafe set $\mathcal{X}_u$ is found within the horizon, the system continues unimpeded. However, if an unsafe state is detected, the monitor performs a binary search between the current state and the unsafe state to locate a cube on the barrier boundary (line 2, Algorithm 4). The online verification procedure begins by adding the identified cube to a queue, referred to as the *boundary*. Each cube in this queue is then iteratively verified. Cubes that do not intersect with the current lookahead

Algorithm 3 ConstructCone (The Abstraction Function)

```

1: function CONSTRUCTCONE( $\mathcal{X}_u, x, h$ )
2:    $cone \leftarrow \{x\}, i \leftarrow 0$  // Initialize cone
3:   while  $i < h$  do
4:      $slice \leftarrow \text{EXPAND}(cone) \setminus cone$ 
5:      $cone \leftarrow cone \cup slice$ 
6:     if  $slice \cap \mathcal{X}_u \neq \emptyset$  then
7:       return ( $cone, \text{PICK}(slice \cap \mathcal{X}_u), i$ )
8:      $i \leftarrow i + 1$ 
9:   return ( $cone, \perp, h$ )
    
```

Algorithm 4 VerifyCubesOnBoundary (The Verifier)

```

1: function VERIFYCUBESONBOUNDARY( $x, x_{\text{unsafe}}, cone, i, h$ )
2:    $cube \leftarrow \text{BINARYSEARCH}(x, x_{\text{unsafe}})$ 
3:    $boundary \leftarrow \{cube\}$ 
4:   while  $boundary \neq \emptyset$  do
5:      $queue \leftarrow \emptyset$ 
6:     for all  $c \in boundary$  do
7:       if  $c \cap \mathcal{X} \cap cone = \emptyset$  then continue
8:       else if  $\neg \text{VERIFYLINEAR}(c)$  then // Check conditions in Def. 2
9:          $\text{FAIL-SAFE}()$ 
10:      else
11:         $queue.\text{PUSH}(c)$ 
12:       $boundary \leftarrow \emptyset$ 
13:      while  $queue \neq \emptyset$  do
14:        for all  $c' \in \text{NEIGHBORHOOD}(queue.\text{pop}())$  do
15:          if  $\text{VERIFIED}(c')$  or  $c' \cap \mathcal{X} = \emptyset$  then continue
16:          else if  $c' \cap \mathcal{X} \cap cone = \emptyset$  then
17:             $boundary.\text{PUSH}(c')$ 
18:          else if  $\neg \text{VERIFYLINEAR}(c')$  then
19:             $\text{FAIL-SAFE}()$ 
20:          else
21:             $queue.\text{PUSH}(c')$ 
22:         $cone \leftarrow cone \cup \text{EXPAND}(cone)$ 
23:         $i \leftarrow i + 1$ 
24:        if  $i > h$  then break
    
```

cone are discarded, as they cannot influence the set of reachable states. If verification fails for any cube, this indicates a violation of the certificate conditions, prompting the system to enter a fail-safe mode or switch to a backup controller. Successfully verified cubes are marked accordingly and added to a queue for neighborhood expansion.

With this initial queue, we start a breadth-first search by examining the 1-bit Hamming neighbors of each verified cube, i.e., those whose ReLU activation patterns differ by exactly one neuron. A neighboring cube is discarded if it has already been verified, or does not intersect any state in $\mathcal{X}$. If it lies outside the current cone, we may verify it after expanding the cone at the next iteration. All intersection checks as well as VERIFYLINEAR in Algorithm 4 operate on a single cube, thus they can be solved efficiently as a simple set of linear inequalities.

5 Experiments

In this section, we evaluate the effectiveness of our monitoring method via a case study and compare it with static verification.

The benchmark we used is a satellite rendezvous example, adapted from [10,15]. In this scenario, a chaser satellite attempts to approach a target satellite while remaining within a designated safe region, corresponding to its line of sight. Both satellites are assumed to be in orbit around the Earth. The system state is represented by the 6-dimensional vector $[x, y, z, v_x, v_y, v_z]$, where $[x, y, z]$ denotes the relative position of the chaser with respect to the target, and $[v_x, v_y, v_z]$ its relative velocity. The control input is given by $u = [u_x, u_y, u_z]$, representing thrust applied along each axis. The control interval is fixed to be 0.1 second.

We evaluate both static formal verification and online monitoring across different neural CBF architectures. Static verification is performed using SEEV [33], a recent tool tailored for verifying ReLU-based certificates. For online monitoring, we employ a local abstraction function that is fixed-step unrolling with known input bounds and a verifier operating on the ReLU activation patterns as described in Section 5.

Table 1. Verification results with corresponding full verification times using SEEV [33].

No. Hidden Layers	No. Neurons per Layer	Verification Result	Full Time (s)
2	8	safe	50.21
4	8	safe	253.20
8	16	—	>2 hours
16	16	unsafe	6.03

Table 1 presents the results of static formal verification for neural CBFs of varying sizes. Notably, the verification of CBF with 8 layers and 16 neurons per layer timed out after two hours using the SEEV method. In contrast, our online monitoring approach was able to detect violations much faster (see Figure 4).

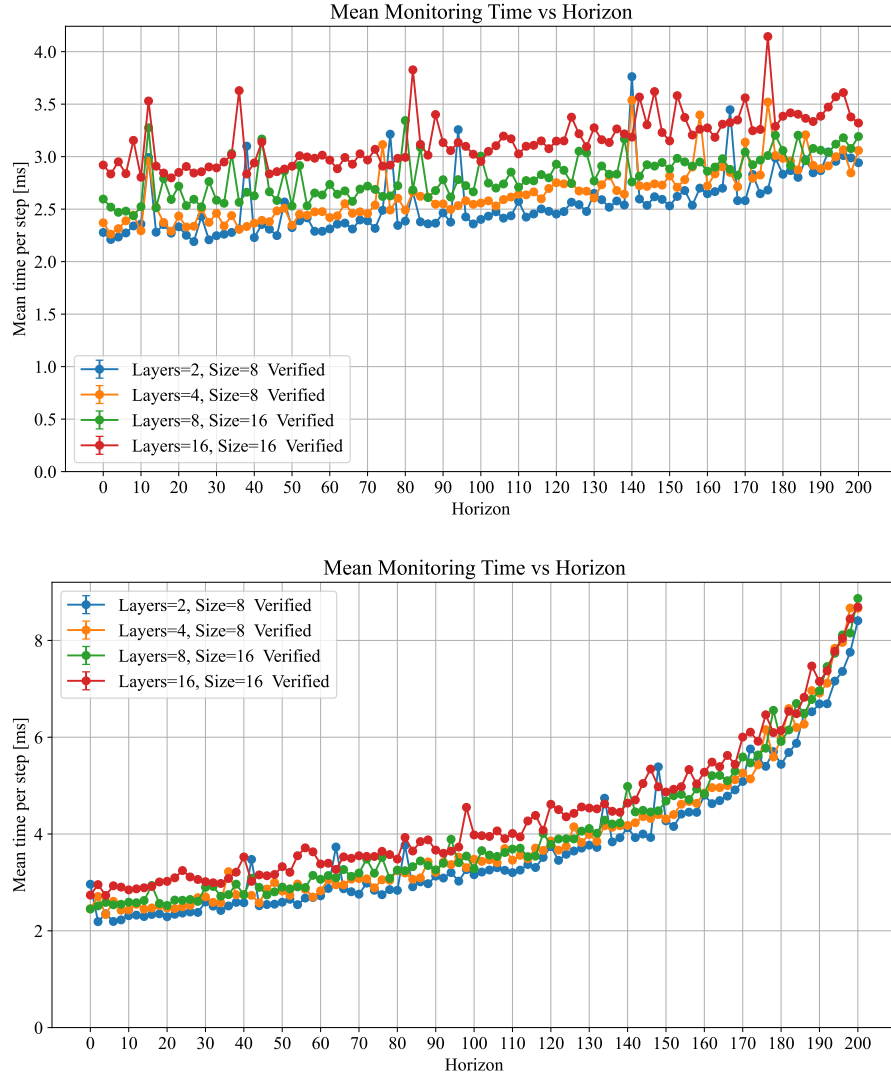


Fig. 3. Monitoring overhead measured for lookahead horizons up to 200 steps, across four network configurations with varying depth and width. All configurations were successfully verified through static analysis (see Table 1). The two plots correspond to different initial states. The monitor did not raise warnings during these runs.

We now evaluate how our monitoring framework performs. As shown in Figure 4, the monitor was able to detect violations with minimal overhead with a longer lookahead horizon (> 70 steps). With shorter horizons, no violations were detected, as the future cones remained relatively small. This also gives an insight: even incorrect certificates may suffice to ensure safety over limited parts of the system’s state space. As the lookahead horizon increases, the monitor explores a broader region and is more likely to encounter violations where the certificate conditions fail. We also observe an increase in monitoring overhead around 70 steps. This is because, as the horizon increases, the over-approximation increases in size, thus intersects more barrier cubes which need to be verified.

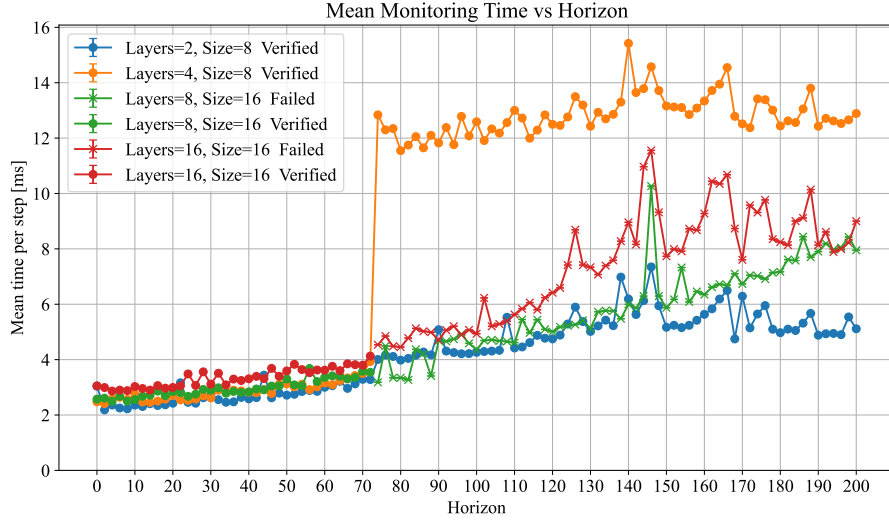


Fig. 4. Monitoring overhead measured for lookahead horizons up to 200 steps, across four network configurations with varying depth and width. Two CBFs failed static verification (one of them timed out), and our monitor successfully detected violations online in both cases, with a higher horizon.

There is a trade-off between monitoring overhead and lookahead horizon, which is as expected. However, we observe that the overhead remains well within practical limits. Figures 3 and 4 report average monitoring overheads for lookahead horizons up to 200 steps (20 seconds). Across all six network configurations, the average overhead remains below 16ms per step, which is substantially lower than the 0.1s control interval. This demonstrates the practicality of our approach for real-time safety monitoring.

In Table 2, we report the results of online monitoring across different network architectures with lookahead horizons. For each configuration, we summarize the number of traces evaluated and the average runtime per monitoring step. Each

Table 2. Online verification results for various lookahead horizons. The outcome column indicates whether verification failed or succeeded. We also display the number of traces evaluated ($\#traces$), with a total number of 323 traces.

Layers	Size	Outcome	Metric	Lookahead Horizon [steps]				
				40	80	120	160	200
2	8	✓	$\#traces$	323	323	323	323	323
			time [ms]	2.52	2.85	3.27	3.82	4.06
4	8	✓	$\#traces$	323	323	323	323	323
			time [ms]	2.72	3.64	4.90	6.69	8.21
8	16	✓	$\#traces$	318	292	255	199	142
			time [ms]	2.76	3.01	3.35	3.75	4.01
8	16	✗	$\#traces$	5	26	63	113	164
			time [ms]	6.18	5.09	5.18	8.64	7.84
16	16	✓	$\#traces$	318	292	255	199	142
			time [ms]	3.09	3.33	3.70	4.07	4.39
16	16	✗	$\#traces$	5	31	68	124	181
			time [ms]	6.49	5.70	6.47	10.37	8.93

configuration was tested on a total of 323 traces, generated by random initial states. As the lookahead horizon increases, our monitor successfully detects more violations in networks that failed verification, confirming the effectiveness of lookahead in revealing unsafe behaviors. The per-step verification overhead remains low across all configurations, even for larger networks.

In addition to deployment-time use, our monitor is also well-suited for identifying counterexamples *during testing*. This makes it a valuable tool for diagnosing safety and certificate violations early in the development cycle. Moreover, it can be integrated into a learner–monitor loop for iteratively repairing certificates, as proposed in [32]. This can be achieved by detecting violations online and adding counterexamples to the training data.

Discussion and limitations. We consider our online monitoring framework to be complementary to static verification for both testing and deployment settings. This can also be particularly valuable in the presence of environmental uncertainties or model inaccuracies, where exhaustive offline verification may not capture all relevant behaviors. While in our case, the monitor still uses the system model for prediction, it performs verification adaptively along the concrete execution path, avoiding exhaustive offline analysis over the entire state space. A potential limitation arises when the system operates near the boundary of the certificate region. In such cases, verifying large numbers of cubes may introduce additional computational overhead, which can affect real-time perfor-

mance. Moreover, if the system is already close to exiting the forward-invariant set, a fail-safe mechanism may not always react in time to prevent a safety violation. This is a fundamental challenge of runtime monitoring in general, where monitors are expected to signal warnings timely while minimizing false alarms. Finally, while our case study focuses on a linearized orbital rendezvous model and a corresponding verification procedure, the proposed framework is general. It applies to nonlinear control-affine systems, provided that suitable local abstractions and verifiers are available. The framework's capabilities scale with the performance of either component.

6 Related Work

Black-box simplex architecture. The Simplex architecture is a runtime assurance method that has been widely adopted in control applications [11,21,23]. It consists of an active controller responsible for primary decision-making and a backup controller that can override the active controller when necessary to ensure runtime safety. Recent research has extended this framework to black-box settings, where the internal architecture of the controllers is assumed to be unknown [17,19]. This is a similar setting to our work where we perform on-the-fly verification, however, they do not consider certificates.

Runtime enforcement using certificates. Shielding is a widely adopted runtime enforcement technique [5] where the shield overrides control inputs to ensure safety. Certificate functions, such as barrier functions, have been integrated with shielding, as demonstrated in [31], where the barrier functions are given in advance. However, in that setting the certificates are not formally verified thus there is no correctness guarantee. More recently, Corsi et al. [18] propose verification-guided shielding, which uses pre-computed verification to identify specific regions for shielding. Unlike these approaches, our method performs real-time verification over a lookahead horizon without requiring preprocessing.

Concolic Testing. Related to our work is also concolic testing [24,25,26], which combines concrete and symbolic execution to explore the local state space of software programs more effectively. Our work draws a parallel to this idea by applying a similar concolic approach in the context of certificate-based control.

Formal verification of certificates. To verify a neural certificate, we need to check whether the network satisfies certificate conditions w.r.t. the controlled system. Broadly speaking there are two approaches, a symbolic approach and a search approach. In the symbolic approach the system, the network, and the certificate condition are encoded as a logical formula and solved using a SMT solver [35,12,1], or they are encoded as a mixed integer linear program and solved with a solver such as Gurobi [36,13]. In the search approach the Lipschitz constant of a neural network is exploited to systematically search the state space, e.g., grid search, to detect potential certificate violations [7,30].

7 Conclusion

In this work, we proposed a general framework for online verification of neural certificates that integrates partial static verification over a finite lookahead horizon. Our method is lightweight, does not require access to future control inputs, and supports modular integration of local abstraction function and certificate verifiers. This enables runtime safety assurance even in settings where static verification is infeasible. We instantiated our framework with a verifier tailored to ReLU-based control barrier functions and demonstrated its practicality on a satellite rendezvous example. The monitor was able to effectively detect certificate violations as well as ensuring runtime safety with minimal overhead.

Our framework is general and readily applies to other types of certificate functions, such as Lyapunov functions and contraction metrics [29], and opens the door to further integration with runtime enforcement mechanisms such as shielding. As future work, we plan to investigate how this monitoring framework can be embedded into training loops to iteratively repair unsafe certificates. We also plan to extend our approach to probabilistic settings to account for environmental uncertainties.

Acknowledgement

This work is supported by the European Research Council under Grant No.: ERC-2020-AdG 101020093.

References

1. Abate, A., Ahmed, D., Edwards, A., Giacobbe, M., Peruffo, A.: Fossil: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In: *Proceedings of the 24th international conference on hybrid systems: computation and control*. pp. 1–11 (2021)
2. Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A., Arshad, H.: State-of-the-art in artificial neural network applications: A survey. *Heliyon* **4**(11) (2018)
3. Ames, A.D., Grizzle, J.W., Tabuada, P.: Control barrier function based quadratic programs with application to adaptive cruise control. In: *53rd IEEE conference on decision and control*. pp. 6271–6278. IEEE (2014)
4. Anwar, S.M., Majid, M., Qayyum, A., Awais, M., Alnowami, M., Khan, M.K.: Medical image analysis using convolutional neural networks: a review. *Journal of medical systems* **42**, 1–13 (2018)
5. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: Runtime enforcement for reactive systems. In: *International conference on tools and algorithms for the construction and analysis of systems*. pp. 533–548. Springer (2015)
6. Chang, Y.C., Roohi, N., Gao, S.: Neural lyapunov control. *Advances in neural information processing systems* **32** (2019)
7. Chatterjee, K., Henzinger, T.A., Lechner, M., Zikelic, D.: A learner-verifier framework for neural network controllers and certificates of stochastic systems. In: *TACAS (1). Lecture Notes in Computer Science*, vol. 13993, pp. 3–25. Springer (2023)

8. Crenshaw, T.L., Gunter, E., Robinson, C.L., Sha, L., Kumar, P.: The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures. In: 28th IEEE International Real-Time Systems Symposium (RTSS 2007). pp. 400–412. IEEE (2007)
9. Dawson, C., Gao, S., Fan, C.: Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Trans. Robotics* **39**(3), 1749–1767 (2023)
10. Dawson, C., Qin, Z., Gao, S., Fan, C.: Safe nonlinear control using robust neural lyapunov-barrier functions. In: Conference on Robot Learning. pp. 1724–1735. PMLR (2022)
11. Desai, A., Ghosh, S., Seshia, S.A., Shankar, N., Tiwari, A.: Soter: a runtime assurance framework for programming safe robotics systems. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 138–150. IEEE (2019)
12. Giacobbe, M., Kroening, D., Pal, A., Tautschnig, M.: Neural model checking. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems (2024)
13. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), <https://www.gurobi.com>
14. Henzinger, T.A., Kresse, F., Mallik, K., Yu, E., Zikelic, D.: Predictive monitoring of black-box dynamical systems. In: L4DC. Proceedings of Machine Learning Research, PMLR (2025)
15. Jewison, C., Erwin, R.S.: A spacecraft benchmark problem for hybrid control and estimation. In: 2016 IEEE 55th Conference on Decision and Control (CDC). pp. 3300–3305. Ieee (2016)
16. Kiumarsi, B., Vamvoudakis, K.G., Modares, H., Lewis, F.L.: Optimal and autonomous control using reinforcement learning: A survey. *IEEE transactions on neural networks and learning systems* **29**(6), 2042–2062 (2017)
17. Maderbacher, B., Schupp, S., Bartocci, E., Bloem, R., Ničković, D., Könighofer, B.: Provable correct and adaptive simplex architecture for bounded-liveness properties. In: International Symposium on Model Checking Software. pp. 141–160. Springer (2023)
18. Mandal, U., Amir, G., Wu, H., Daukantas, I., Newell, F.L., Ravaioli, U.J., Meng, B., Durling, M., Ganai, M., Shim, T., et al.: Formally verifying deep reinforcement learning controllers with lyapunov barrier certificates. In: # PLACEHOLDER_PARENT_METADATA_VALUE#. pp. 95–106. TU Wien Academic Press (2024)
19. Mehmood, U., Sheikhi, S., Bak, S., Smolka, S.A., Stoller, S.D.: The black-box simplex architecture for runtime assurance of autonomous cps. In: NASA formal methods symposium. pp. 231–250. Springer (2022)
20. Peruffo, A., Ahmed, D., Abate, A.: Automated and formal synthesis of neural barrier certificates for dynamical models. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 370–388. Springer (2021)
21. Phan, D., Yang, J., Grosu, R., Smolka, S.A., Stoller, S.D.: Collision avoidance for mobile robots with limited sensing and limited information about moving obstacles. *Formal Methods in System Design* **51**, 62–86 (2017)
22. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Trans. Autom. Control.* **52**(8), 1415–1428 (2007)

23. Schierman, J.D., DeVore, M.D., Richards, N.D., Gandhi, N., Cooper, J.K., Horne-
man, K.R., Stoller, S., Smolka, S.: Runtime assurance framework development
for highly adaptive flight control systems. Barron Associates, Inc. Charlottesville,
Tech. Rep (2015)
24. Sen, K.: Concolic testing. In: Proceedings of the 22nd IEEE/ACM international
conference on Automated software engineering. pp. 571–572 (2007)
25. Sen, K.: DART: directed automated random testing. In: Haifa Verification Confer-
ence. Lecture Notes in Computer Science, vol. 6405, p. 4. Springer (2009)
26. Sen, K., Marinov, D., Agha, G.: CUTE: a concolic unit testing engine for C. In:
ESEC/SIGSOFT FSE. pp. 263–272. ACM (2005)
27. Seto, D., Krogh, B., Sha, L., Chutinan, A.: The simplex architecture for safe online
control system upgrades. In: Proceedings of the 1998 American Control Conference.
ACC (IEEE Cat. No. 98CH36207). vol. 6, pp. 3504–3508. IEEE (1998)
28. Smith, M.C.: The general problem of the stability of motion : Translated and edited
by a. t. fuller. taylor and francis, 1992. Autom. **31**(2), 353–354 (1995)
29. Sun, D., Jha, S., Fan, C.: Learning certified control using contraction metric. In:
conference on Robot Learning. pp. 1519–1539. PMLR (2021)
30. Tayal, M., Zhang, H., Jagtap, P., Clark, A., Kolathaya, S.: Learning a for-
mally verified control barrier function in stochastic environment. arXiv preprint
arXiv:2403.19332 (2024)
31. Yin, J., Dawson, C., Fan, C., Tsiotras, P.: Shield model predictive path integral: A
computationally efficient robust mpc method using control barrier functions. IEEE
Robotics and Automation Letters **8**(11), 7106–7113 (2023)
32. Yu, E., Žikelić, Đ., Henzinger, T.A.: Neural control and certificate repair via run-
time monitoring. In: Proceedings of the AAAI Conference on Artificial Intelligence.
vol. 39, pp. 26409–26417 (2025)
33. Zhang, H., Qin, Z., Gao, S., Clark, A.: SEEV: synthesis with efficient exact verifi-
cation for relu neural barrier functions. In: NeurIPS (2024)
34. Zhang, H., Wu, J., Vorobeychik, Y., Clark, A.: Exact verification of relu neural
control barrier functions. Advances in neural information processing systems **36**,
5685–5705 (2023)
35. Zhao, H., Zeng, X., Chen, T., Liu, Z.: Synthesizing barrier certificates using neural
networks. In: Proceedings of the 23rd international conference on hybrid systems:
Computation and control. pp. 1–11 (2020)
36. Zhao, Q., Chen, X., Zhao, Z., Zhang, Y., Tang, E., Li, X.: Verifying neural net-
work controlled systems using neural networks. In: Proceedings of the 25th ACM
International Conference on Hybrid Systems: Computation and Control. pp. 1–11
(2022)