International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-021-00623-1

GENERAL

Special Issue: RV 2019

# Neural predictive monitoring and a comparison of frequentist and Bayesian approaches

Luca Bortolussi[1,2] · Francesca Cairoli[1] · Nicola Paoletti[3] · Scott A. Smolka[4] · Scott D. Stoller[4]

Accepted: 6 May 2021

**Abstract**
Neural state classification (NSC) is a recently proposed method for runtime predictive monitoring of hybrid automata (HA) using deep neural networks (DNNs). NSC trains a DNN as an approximate *reachability predictor* that labels an HA state $x$ as *positive* if an unsafe state is reachable from $x$ within a given time bound, and labels $x$ as *negative* otherwise. NSC predictors have very high accuracy, yet are prone to prediction errors that can negatively impact reliability. To overcome this limitation, we present *neural predictive monitoring* (NPM), a technique that complements NSC predictions with estimates of the predictive uncertainty. These measures yield principled criteria for the rejection of predictions likely to be incorrect, without knowing the true reachability values. We also present an active learning method that significantly reduces the NSC predictor's error rate and the percentage of rejected predictions. We develop two versions of NPM based, respectively, on the use of frequentist and Bayesian techniques to learn the predictor and the rejection rule. Both versions are highly efficient, with computation times on the order of milliseconds, and effective, managing in our experimental evaluation to successfully reject almost all incorrect predictions. In our experiments on a benchmark suite of six hybrid systems, we found that the frequentist approach consistently outperforms the Bayesian one. We also observed that the Bayesian approach is less practical, requiring a careful and problem-specific choice of hyperparameters.

**Keywords** Predictive monitoring · Runtime verification · Hybrid automata reachability · Neural networks · Conformal prediction · Bayesian inference

## 1 Introduction

Hybrid systems are a central model for many safety-critical, cyber-physical system applications [1]. Their verification typically amounts to solving a hybrid automata (HA) reachability checking problem [2]: given a model $\mathcal{M}$ of the system expressed as an HA, a set $I$ of initial states of $\mathcal{M}$, and a set $D$ of unsafe states, check whether $D$ is reached along any (time-bounded) path of $\mathcal{M}$ starting from a state in $I$. Due to its high computational cost, reachability checking is usually limited to design-time (offline) analysis.

Our focus is on the online analysis of hybrid systems and, in particular, on the *predictive monitoring* (PM) problem [3], i.e., the problem of predicting, *at runtime*, whether or not an unsafe state can be reached from the current system state within a given time bound. PM is at the core of architectures for runtime safety assurance such as Simplex [4], where the system switches to a certified-safe baseline controller whenever PM indicates the potential for an imminent safety violation.

In such approaches, PM is invoked periodically and frequently. Thus, reachability needs to be determined rapidly, from a single state (the current system state), and typically for short time horizons. This is in contrast with offline reachability checking, where long or unbounded time horizons and sizable regions of initial states are typically considered. PM also differs from traditional runtime verification [5] in that

✉ Nicola Paoletti
nclpltt@gmail.com

1 Department of Mathematics and Geosciences, Università di Trieste, Trieste, Italy

2 Modelling and Simulation Group, Saarland University, Saarbrücken, Germany

3 Department of Computer Science, Royal Holloway, University of London, London, UK

4 Department of Computer Science, Stony Brook University, New York, USA

Published online: 24 May 2021

PM is *preemptive*: it detects potential safety violations before they occur, not when or after they occur.

Any solution to the PM problem involves a trade-off between two main requirements: *accuracy* of the reachability prediction, and computational *efficiency*, as the analysis must execute within strict real-time constraints and typically with limited hardware resources.

In this paper, we present *neural predictive monitoring* (NPM), a machine-learning-based approach to PM that provides highly accurate predictions in a highly efficient manner. Moreover, NPM offers principled methods for detecting potential *prediction errors*, which significantly enhances the reliability of PM estimates.

NPM builds on *neural state classification* (NSC) [6], a recently proposed method for approximate HA reachability checking using deep neural networks (DNNs). NSC works by training a DNN as a state classifier using examples computed with an oracle (an HA model checker). For any state $x$ of the HA, such a classifier labels $x$ as *positive* if an unsafe state is reachable from $x$ within a given time bound; otherwise, $x$ is labeled as *negative*.

Executing a neural state classifier corresponds to computing the output of a DNN for a single input, and thus is extremely efficient. NSC has also demonstrated very high accuracy in reachability predictions, owing to the powerful approximation capabilities of DNNs. Some classification errors are, however, unavoidable, the most important being *false negatives*, in which positive states are misclassified as negative. Such errors may compromise system safety.

NPM overcomes this problem by extending NSC with rigorous methods for quantifying the uncertainty of the reachability estimates. NPM can consequently identify and reject predictions that are likely to produce classification errors. We investigate two alternative NPM methods: a *frequentist* approach that uses *Conformal Prediction* (CP) [7], a method that provides statistical guarantees on the predictions of machine-learning models with minimal assumptions on the data;[1] and a *Bayesian approach* that leverages uncertainty quantification via *Bayesian neural networks* (BNNs), rather than traditional (deterministic) DNNs. We consider two popular Bayesian inference methods: Hamiltonian Monte Carlo [8] and Variational Inference [9].

Figure 1 provides an overview of the NPM approach. We sample from a distribution of HA states to generate a training set $Z_t$ and a validation set $Z_v$. An HA reachability oracle (a model checker or, for deterministic systems, a simulator) is used to label sampled states as positive or negative. A neural state classifier $F$ (i.e., a DNN-based binary classifier) is derived from $Z_t$ via supervised learning, and is either a

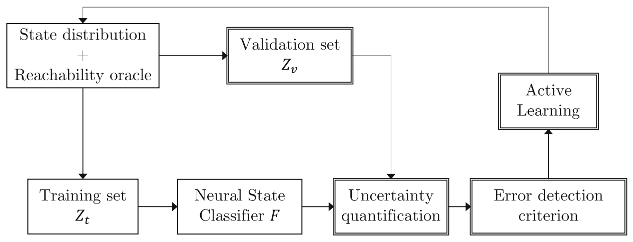

**Fig. 1** Overview of the NPM framework. Double-bordered components denote extensions to the method of [6]. Training of the neural state classifier $F$ and retraining via active learning are performed offline. The only components used at runtime are the classifier $F$ and the error detection criterion

deterministic neural network (in the frequentist approach) or a Bayesian neural network (in the Bayesian approach).

In the frequentist case, CP is used to estimate two statistically sound measures of prediction uncertainty: *confidence* and *credibility*. Informally, the confidence of a prediction is the probability that a reachability prediction for an HA state $x$ corresponds to the true reachability value of $x$. Credibility quantifies how likely a given state is to belong to the same distribution of the training data. In the Bayesian case, we use an empirical approximation of the BNN output distribution and, to be precise, statistics of said distribution to quantify the model uncertainty about a specific HA state.

The measures described above, henceforth referred to as *uncertainty measures*, are used to derive criteria for error detection, i.e., for rejecting NSC predictions that are likely to be erroneous. The rejection criterion is based on identifying, via supervised learning on the validation set $Z_v$, a decision rule that optimally separates incorrect and correct predictions. We again consider both a frequentist and a Bayesian approach, deriving a support vector classifier for the former, and a Gaussian process classifier for the latter.

Executing a neural predictive monitor corresponds to: (i) computing the output of a DNN, either deterministic or Bayesian, on a single input; (ii) computing the corresponding measure of uncertainty, whose computational cost depends on the size of the validation set for the frequentist case, and on the sample size used for the empirical distribution of the BNN for the Bayesian case; (iii) evaluate the rejection rule on the measure of uncertainty obtained via step (ii). For all of the approaches we consider, this whole process is very efficient, taking from 2 milliseconds to 0.3 seconds in our experiments. This makes our NPM method suitable for online analysis and PM.

Finally, our approach includes an active learning strategy to improve the reliability of the state classifier $F$. The idea is to employ the uncertainty-based rejection criterion to identify HA states for which $F$ yields uncertain predictions, and augment the training and validation sets with those states. We then train a new state classifier with the augmented dataset,

[1] The only assumption is exchangeability, a weaker version of the independent and identically distributed assumption.

thus ensuring improved accuracy on the HA states where $F$ performed poorly, and, in turn, a reduced rejection rate.

Compared to simple random sampling of the state distribution, our active learning strategy has the advantage of parsimony: by focusing on the states with uncertain predictions, it requires a significantly smaller number of additional retraining samples to achieve a given reduction in the rejection rate, and thus significantly reduces the cost of retraining. The active learning procedure can be iterated, as shown in Fig. 1. We stress that these retraining iterations are part of the training process, which is performed offline and hence does not affect runtime performance.

In summary, the main contributions of this work are the following:

- We develop neural predictive monitoring, a framework for runtime predictive monitoring of hybrid automata that extends neural state classification with quantification of prediction reliability.
- We instantiate the NPM framework in two variants, which, respectively, use frequentist and Bayesian learning techniques. For both approaches, we derive optimal criteria for rejecting unreliable NSC predictions by leveraging sound measures of prediction uncertainty.
- We develop an active learning method designed to reduce both prediction errors and the rejection rate.
- We evaluate our method on six hybrid automata case studies, demonstrating that our optimal rejection criteria successfully rejects almost all prediction errors, with 100% of the errors recognized in 5 out of 6 case studies for the frequentist method. Moreover, only a single active learning iteration is needed to significantly increase the prediction accuracy and reduce the rejection rate.
- With our analysis and experimental comparison of frequentist and Bayesian variants of NPM, we show that the frequentist approach empirically outperforms the Bayesian one on all relevant metrics. Furthermore, the frequentist techniques require minimal assumptions and tuning, which makes them more practical.
  In contrast, Bayesian inference requires tuning of a number of hyperparameters, including the prior distribution for the DNN weights. If the hyperparameters are not optimally tuned, the performance may be extremely poor.

This work is an extended version of [10], where we first introduced the NPM method, but only the frequentist version. Here, we introduce a fully Bayesian variant of NPM, and compare the two versions in a new experimental evaluation section. In this version of the paper, we also include a more extensive treatment of the problem formulation and the NPM method itself.

The paper is structured as follows. Section 2 presents a rigorous formulation of the problem. Section 3 provides background on neural state classification and on the methods used to estimate predictive uncertainty. The uncertainty-based error detection rules are presented in Sect. 4. Section 5 presents the active learning algorithm. Results of the experimental evaluation are given in Sect. 6. Related work is discussed in Sect. 7. Section 8 offers concluding remarks.

## 2 Problem formulation

We describe the predictive monitoring problem for hybrid automata reachability and the related problem of finding an optimal criterion for rejecting erroneous reachability predictions.

**Definition 1** (Hybrid automaton) A *hybrid automaton* (HA) is a tuple $\mathcal{M} = (Loc, Var, Init, Flow, Trans, Inv)$, where $Loc$ is a finite set of discrete *locations* (or modes); $Var = \{v_1, \dots, v_n\}$ is a set of continuous *variables*, evaluated over a *continuous domain* $V \subseteq \mathbb{R}^n$; $Init \subseteq S(\mathcal{M})$ is the set of *initial states*, where $S(\mathcal{M}) = Loc \times V$ is the *state space* of $\mathcal{M}$; $Flow : Loc \to (V \to V)$ is the *flow* function, defining the continuous dynamics at each location; $Trans$ is the *transition relation*, consisting of tuples of the form $(l, g, r, l')$, where $l, l' \in Loc$ are *source and target locations*, respectively, $g \subseteq V$ is the *guard*, and $r : V \to V$ is the *reset*; $Inv : Loc \to 2^V$ is the *invariant* at each location.

We also consider *parameterized* HA in which the flow, guard, reset and invariant may have parameters whose values are constant throughout an execution. We treat parameters as continuous variables with flow equal to zero and identity reset map. The behavior of an HA $\mathcal{M}$ can be described in terms of its trajectories. A trajectory may start from any state; it does not need to start from an initial state. For time bound $T \in \mathbb{R}^{\geq 0}$, we denote with $\mathbb{T} = [0, T] \subseteq \mathbb{R}^{\geq 0}$ the time domain.

**Definition 2** (Trajectory [11]) For HA $\mathcal{M} = (Loc, Var, Init, Flow, Trans, Inv)$, time domain $\mathbb{T} = [0, T]$, let $\rho : \mathbb{T} \to S(\mathcal{M})$ be a function mapping time instants into states of $\mathcal{M}$. For $t \in \mathbb{T}$, let $\rho(t) = (l(t), v(t))$ be the state at time $t$, with $l(t)$ being the location and $v(t)$ the vector of continuous variables. Let $(\xi_i)_{i=0,\dots,k} \in \mathbb{T}^{k+1}$ be the ordered sequence of time points where mode jumps happen, i.e., such that $\xi_0 = 0$, $\xi_k = T$, and for all $i = 0, \dots, k-1$ and for all $t \in [\xi_i, \xi_{i+1})$, $l(t) = l(\xi_i)$. Then, $\rho$ is a trajectory of $\mathcal{M}$ if it is consistent with the invariants: $\forall t \in \mathbb{T}.\ v(t) \in Inv(l(t))$; flows: $\forall t \in \mathbb{T}.\ \dot{v}(t) = Flow(l(t))(v(t))$; and transition relation: $\forall i < k.\ \exists (l(\xi_i), g, r, l(\xi_{i+1})) \in Trans.\ v(\xi_{i+1}^-) \in g \wedge v(\xi_{i+1}) = r(v(\xi_{i+1}^-))$.

***Example 1*** (Spiking neuron HA) This model describes the evolution of a neuron's action potential. It is a deterministic

HA with two continuous variables, one mode, one jump and nonlinear polynomial dynamics, defined by the ODE

$$\begin{cases} \dot{v_2} &= 0.04v_2^2 + 5v_2 + 140 - v_1 + I \\ \dot{v_1} &= a \cdot (b \cdot v_2 - v_1) \end{cases} \qquad (1)$$

The jump condition is $v_2 \geq 30$, and the associated reset is $v_2' := c \wedge v_1' := v_1 + d$, where, for any variable $x$, $x'$ denotes the value of $x$ after the reset. We consider the unsafe set $D$ defined by $v_2 \leq 68.5$, expressing that the neuron should not undershoot its resting potential. The parameter values are given in Appendix C.

Reachability checking of HA is concerned with establishing whether, given an initial HA state $x$ and a set of target states $D$ – typically a set of unsafe states to avoid – the HA admits a trajectory starting from $x$ that reaches $D$.

**Definition 3** (Time-bounded reachability) Given an HA $\mathcal{M}$ with state space $X$, a set of states $D \subseteq X$, state $x \in X$, and time bound $T$, decide whether there exists a trajectory $\rho$ of $\mathcal{M}$ starting from $x$ and $t \in [0, T]$ such that $\rho(t) \in D$, denoted $\mathcal{M} \models \mathsf{Reach}(D, x, T)$.

We aim to derive a predictive monitor for HA reachability, i.e., a function that can predict whether or not a state in $D$ (an unsafe state) can be reached from the current system state within time $T$. In solving this problem, we assume a distribution $\mathcal{X}$ of HA states and seek the monitor that predicts HA reachability with minimal error probability w.r.t. $\mathcal{X}$. The choice of $\mathcal{X}$ depends on the application at hand and can include a uniform distribution on a bounded state space or a distribution reflecting the density of visited states in some HA executions [6].

**Problem 1** (Predictive monitoring for HA reachability) Given an HA $\mathcal{M}$ with state space $X$, a distribution $\mathcal{X}$ over $X$, a time bound $T$ and set of unsafe states $D \subseteq X$, find a function $F^* : X \to \{0, 1\}$ that minimizes the probability

$$Pr_{x\sim\mathcal{X}}\left(F^*(x) \neq \mathbf{1}(\mathcal{M} \models \mathsf{Reach}(D, x, T))\right),$$

where $\mathbf{1}$ is the indicator function. A state $x \in X$ is called *positive* w.r.t. a predictor $F : X \to \{0, 1\}$ if $F(x) = 1$. Otherwise, $x$ is called *negative* (w.r.t. $F$).

Any practical solution to the above PM problem must also assume a space of functions within which to restrict the search for the optimal predictive monitor $F^*$. Following the neural state classification method of [6], in this work we consider functions described by deep neural networks (DNNs)[2]. Finding $F^*$, i.e., finding a function approximation with minimal error probability, is indeed a classical machine-learning problem, a *supervised classification* problem in particular, with $F^*$ being the *classifier*, i.e., the function mapping HA state inputs $x$ into one of two classes: 1 ($x$ is positive, can reach $D$) and 0 ($x$ is negative, cannot reach $D$).

[2] In [6], the PM problem is called "state classification problem," and its solution a "state classifier."

Machine-learning classifiers often admit an underlying *discriminant function*, which is used to determine the final classifier output. In neural networks, the discriminant is the function mapping inputs into the class likelihoods (i.e., the softmax probabilities of the last network layer), such that the predicted class is the one with the highest likelihood.

**Definition 4** (Discriminant function) Let $Y = \{y^1, \ldots, y^c\}$ be a set of classes. A function $f : X \to [0, 1]^c$ is a *discriminant* for a classifier $F : X \to Y$ iff for any input $x \in X$, $F(x) \in \arg\max_{y^i \in Y} f_i(x)$, where $f_i(x)$ is the $i$-th component of $f(x)$.

***Remark 1*** If $\arg\max_{y^i \in Y} f_i(x)$ contains more than one class, then the discriminant implies multiple possible predictions for $x$. To avoid this ambiguity, we will assume that $f$ is always *well defined*, meaning that $\arg\max_{y^i \in Y} f_i(x)$ is a singleton, and thus, only one prediction is possible for the classifier $F$. Any discriminant can be made well defined by, for instance, imposing a total ordering on the classes to select in such ambiguous cases and adequately adjusting the discriminant output[3].

In what follows, we will interchangeably use the term "(reachability) predictor" for the classifier $F$ and for its discriminant $f$. Likewise, we will also call $F$ a state classifier, and in particular, neural state classifier (NSC) when its discriminant $f$ is described by a deep neural network. Below we provide a definition of discriminant based on feed-forward neural networks.

**Definition 5** (Deep Neural Network discriminant) A DNN-based discriminant over $c$ classes can be defined as a function $f_w : X \to [0, 1]^c$ of the form:

$$f_w = \mathrm{f}_L \circ \mathrm{f}_{L-1} \circ \ldots \circ \mathrm{f}_1 \circ \mathrm{f}_0,$$

where $L$ is the number of hidden layers, $\circ$ is the function composition operator, $\mathrm{f}_0$ is the input standardization function and, for $i = 1, \ldots, L$, $\mathrm{f}_i$ is the function computed by the $i$-th layer. In particular, $\mathrm{f}_L$ is a function mapping the output of layer $L - 1$ to $[0, 1]^c$.

Let $n_i$ indicate the number of neurons in layer $i$ and let $\mathrm{o}_{i-1} \in \mathbb{R}^{n_{i-1}}$ be the output vector of layer $i - 1$. The output

[3] Any discriminant $f$ can be turned into a well-formed one by $f'(x) = \{f(x)$ if $|Y_{\max}| = 1$; $(f(x) + \mathbf{0}_{k\mapsto\epsilon})/(1+\epsilon)$ otherwise$\}$, where $Y_{\max} = \arg\max_{y^i \in Y} f_i(x)$, $\epsilon \in \mathbb{R}^+$, $k = \max_{y^i \in Y_{\max}} i$ (based on the ordering on $Y$) and $\mathbf{0}_{k\mapsto\epsilon} \in \mathbb{R}^c$ is a vector whose components are all zeros but the $k$-th component equals to $\epsilon$.

of layer $i$ results from applying function $\mathrm{f}_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ to the output of the previous layer:

$$\mathrm{f}_i\left(\mathrm{o}_{i-1}\right) = \mathrm{a}_i\left(\mathrm{w}_{i,i-1} \cdot \mathrm{o}_{i-1} + \mathrm{b}_i\right), \quad i = 1, \ldots, l \qquad (2)$$

where $\mathrm{w}_{i,i-1} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the *weight matrix* that connects $\mathrm{o}_{i-1}$ to the neurons of layer $i$, $\mathrm{b}_i \in \mathbb{R}^{n_i}$ is the *bias vector* of layer $i$, and $\mathrm{a}_i$ is the *activation function* of the neurons of layer $i$. Weights and biases are the parameters learned during training.

*Training set.* In supervised learning, one minimizes a measure of the empirical prediction error w.r.t. a *training set*. In our case, the training set $Z'$ is obtained from a finite sample $X'$ of $\mathcal{X}$ by labeling the training inputs $x \in X'$ using some reachability oracle, that is, a hybrid automata reachability checker like [12–15]. Hence, given a sample $X'$ of $\mathcal{X}$, the training set is defined by

$$Z' = \{(x, \mathbf{1}(\mathsf{Reach}(D, x, T)) \mid x \in X'\}.$$

*Prediction errors.* It is well known that neural networks are universal approximators, i.e., they are expressive enough to approximate arbitrarily well the output of any measurable mathematical function [16]. Even though arbitrarily high precision might not be achievable in practice, state-of-the-art optimization methods based on gradient descent via back-propagation [17] can effectively learn highly accurate neural network approximators. However, such methods cannot completely avoid prediction errors (no supervised learning method can). Therefore, we have to deal with predictive monitors $F$ that are prone to prediction errors, which are of two kinds:

- *false positives* (FPs), when, for a state $x \in X$, $F(x) = 1$ ($x$ is positive w.r.t. $F$) but $\mathcal{M} \not\models \mathsf{Reach}(D, x, T)$, and
- *false negatives* (FNs), when $F(x) = 0$ ($x$ is negative w.r.t. $F$) but $\mathcal{M} \models \mathsf{Reach}(D, x, T)$.

These errors are, respectively, denoted by predicates $\mathit{fn}(x)$ and $\mathit{fp}(x)$. In what follows, we consider general kinds of prediction error, described by predicate $\mathit{pe}(x)$, and defined by any arbitrary combination of $\mathit{fn}(x)$ and $\mathit{fp}(x)$.

***Remark 2*** (Feature space and state space). Neural networks only admit inputs from some vector space $\subseteq \mathbb{R}^n$, which is also called feature space. However, the state space $X$ of a hybrid automaton is not a vector space as it is given by $X = \mathit{Loc} \times V$, where $\mathit{Loc}$ is the finite set of HA locations and $V$ is the domain of the continuous HA variables (see Definition 1). To this purpose, for $x = (l, v) \in X$, we apply the (straightforward) vector embedding $(l, v) \mapsto [\#(l)\ v]^T$, where $\# : \mathit{Loc} \to \mathbb{R}$ is a suitable injection (a typical choice is assigning an ordinal value to each HA location). In what follows, we keep this vector embedding implicit and work directly with inputs over $X$.

## 2.1 Uncertainty-based error detection

A central objective of this work is to derive, given a predictor $f$, a rejection criterion $R_f$ able to identify states $x$ that are wrongly classified by $F$, i.e., FNs and FPs or any combination of the two, without knowing the true reachability value of $x$. Further, $R_f$ should be optimal, that is, it should ensure minimal probability of rejection errors w.r.t. the state distribution $\mathcal{X}$. For this purpose, we propose to utilize information about the reliability of reachability predictions, so as to detect and reject potentially erroneous (i.e., unreliable) predictions.

Our solution relies on enriching each prediction with a measure of predictive uncertainty: given $f$, we define a function $u_f : X \to U$ mapping an HA state $x \in X$ into some measure $u_f(x)$ of the uncertainty of $f$ about $x$. The set $U$ is called *uncertainty domain*. Defining $u_f$ and $U$ is a non-trivial task, details are provided in Sect. 3. The only requirements are that $u_f$ is point-specific and should not use any knowledge about the true reachability value. Once defined, $u_f$ can be used to build an optimal error detection criterion, as explained below.

**Problem 2** (Uncertainty-based error detection) Given a reachability predictor $f$, a distribution $\mathcal{X}$ over HA states $X$, a predictive uncertainty measure $u_f : X \to U$ over some uncertainty domain $U$, and a kind of error $\mathit{pe}$ find an optimal error detection rule $G^*_{f,pe} : U \to \{0, 1\}$, i.e., a function that minimizes the probability

$$Pr_{x\sim\mathcal{X}}(pe(x) \neq G^*_{f,pe}(u_f(x))).$$

Note that Problem 2 requires specifying the kind of prediction errors to reject. Indeed, depending on the application at hand, one might desire to reject only a specific kind of errors. For instance, in safety-critical applications, FNs are the most critical errors while FPs are less important.

As for Problem 1, we can obtain a sub-optimal solution $G_{f,pe}$ to Problem 2 by expressing the latter as a supervised learning problem, where the inputs are, once again, sampled according to $\mathcal{X}$ and labeled using a reachability oracle. We call *validation set* the set of labeled observations used to learn $G_{f,pe}$. These observation need to be independent from the above introduced training set $Z'$, i.e., those used to learn the reachability predictor $f$. In the simplest scenarios, learning $G_{f,pe}$ reduces to identifying an optimal threshold. However, the proposed supervised learning solution is capable of identifying complex and multi-dimensional decision boundaries in an automatic fashion, making it suitable also for complex scenarios.

For an error $pe$, the final rejection rule $R_{f,pe}$ for detecting HA states where the reachability prediction should not be trusted, and thus rejected, is readily obtained by the composition of the uncertainty measure and the error detection rule

$$R_{f,pe} = G_{f,pe} \circ u_f : X \to \{0,1\},$$

where $R_{f,pe}(x) = 1$ if the prediction on state $x$ is rejected; $R_{f,pe}(x) = 0$, otherwise. We remark that rejection rules for different kinds of errors could be combined together to express more sophisticated criteria.

## List of notation

| | |
|---|---|
| $F$ $(f)$ | reachability predictor (NSC) (and its discriminant) |
| $u_f$ | uncertainty function for $f$ |
| $G_f$ $(g_f)$ | error detection function for $f$ (and its discriminant) |
| $R_f$ | rejection function for $f$ |
| $X$ $(\mathcal{X})$ | HA state space (and its distribution) |
| $D$ | set of unsafe states |
| $Y$ | set of possible reachability values |
| $Z = X \times Y$ $(\mathcal{Z})$ | data domain (and its distribution) |
| $U$ $(U_{\mathcal{F}}, U_{\mathcal{B}})$ | uncertainty domain (frequentist, Bayesian versions) |
| $pe, fn, fp$ | type of prediction errors |
| $Z_t, Z_v, Z_c$ | training, validation, calibration datasets |
| $Z_t^a, Z_v^a, Z_c^a$ | augmented training, validation, calibration datasets in active learning |
| $U_v$ | uncertainty measures for $f$ over $Z_v$ |
| $E_v$ | error labels for $f$ over $Z_v$ |
| $W_v$ | training set for the error detection function $G_f$ |

## 3 Uncertainty quantification in neural predictive monitoring

We explore two approaches to quantify the uncertainty produced by a neural network-based reachability predictor $f$, i.e., to derive the uncertainty measures $u_f$ introduced in Section 2.1.

The first approach, referred to as the *frequentist approach*, is based on Conformal Prediction [7,18,19]. The second one, referred to as the *Bayesian approach*, relies on Bayesian learning and employs probability distributions to express and measure uncertainty. In particular, we leverage the theory of Bayesian neural networks [20], which combines neural networks and probabilistic modeling.

We present the two uncertainty quantification approaches for a generic classification problem, where we consider an input space $X$, a set of classes $Y = \{y^1, \ldots, y^c\}$, and a classifier $F : X \to Y$ with discriminant $f : X \to [0,1]^c$. For an input $x$, we will use the notation $\hat{y}$ as a shorthand for $F(x)$, the classifier prediction on $x$. We define the data domain by $Z = X \times Y$, and we denote with $\mathcal{Z}$ the distribution of the data over $Z$.

In the context of PM of HA reachability, $X$ is the HA state space, $Y = \{0,1\}$ $(c = 2)$ is the set of possible reachability values, $\mathcal{Z} = Pr_{x\sim\mathcal{X}}(x, \mathbf{1}(\mathsf{Reach}(D,x,T)))$, where $\mathcal{X}$ is the distribution of HA states and $\mathsf{Reach}(D,x,T)$ is the reachability specification, and $F$ is the reachability predictor, i.e., a (sub-)optimal solution of Problem 1.

### 3.1 Conformal prediction

Conformal Prediction (CP) is a technique that associates measures of reliability to any traditional supervised learning model. It is a very general approach that can be applied across all existing classification and regression methods [7,18,19]. CP produces *prediction regions* instead of single point predictions: given a significance level $\varepsilon \in (0,1)$ and a test point $x_*$, its prediction region, $\Gamma_*^\varepsilon \subseteq Y$, is a set of classes that are guaranteed to contain the true class $y_*$ with probability $1-\varepsilon$, i.e.,

$$Pr_{(x_*,y_*)\in\mathcal{Z}}(y_* \in \Gamma_*^\varepsilon) \geq 1 - \varepsilon.$$

The CP method requires defining a so-called *non-conformity function (NCF)* $h : Z \to \mathbb{R}$: given a predictor $f$ and an example $z = (x,y)$, $h(z)$ measures the "strangeness" of $z$, i.e., the deviation between the label $y$ and the corresponding prediction $f(x)$. The definition of a suitable NCF for our problem is discussed later. Now consider the distribution of NCF scores induced by the data distribution $\mathcal{Z}$ and the predictor $f$:

$$\mathcal{H} = Pr_{(x,y)\sim\mathcal{Z}}(h(x,y)).$$

Note that $\mathcal{H}$ precisely captures the distribution of the distances between true classes and corresponding predictions. The rationale behind CP is to construct the prediction region by "inverting" a suitable hypothesis test: given a test point $x_*$ and a tentative class $y^j \in Y$, we *exclude* $y^j$ from the prediction region only if it appears unlikely that the NCF score $h(x_*, y^j)$ is distributed according to $\mathcal{H}$, which implies that it is not likely that $y^j$ is the true class. In other words, for each $y^j \in Y$, we perform the following hypothesis test:

$$H_0 : h(x_*, y^j) \sim \mathcal{H} \text{ against } H_a : h(x_*, y^j) \nsim \mathcal{H},$$

and include in $\Gamma_*^\varepsilon$ all the $y^j$ values for which we fail to reject the null hypothesis $H_0$ at significance level $\varepsilon$ (this is what we mean by "inverting the test"). In particular, $H_0$ is rejected if

$$Pr_{\alpha \sim \mathcal{H}}(\alpha \geq h(x_*, y^j)) < \epsilon, \tag{3}$$

i.e., if the probability of observing a data point that is more "non-conforming" than $(x_*, y^j)$ is below $\epsilon$. This probability is called *p value* in statistical jargon.

Note that exact derivation of $\mathcal{H}$ is intractable, as it requires integration of $h$, which depends on the classifier and thus is typically nonlinear, over $\mathcal{Z}$, a distribution that is seldom given in explicit form (but only empirically, through a set of observations). Therefore, we use instead an empirical approximation of $\mathcal{H}$, derived by computing the NCF scores of a finite sample $Z_c$ of $\mathcal{Z}$ independent of the training set used to learn the predictor $f$. We call $Z_c$ the *calibration set*. This approach is called inductive CP [18]. We summarize it in the algorithm below.

*CP algorithm for classification.* Given a sample $Z'$ of $\mathcal{Z}$, a test input $x_* \in X$, and a significance level $\varepsilon \in (0, 1)$, CP computes a prediction region $\Gamma_*^\varepsilon$ for $x_*$ as follows.

1. Divide $Z'$ into a training set $Z_t$ and a calibration set $Z_c$.
2. Train a predictor $f$ using $Z_t$.
3. Define a NCF $h : Z \to \mathbb{R}$.
4. Apply $h(z)$ to each example $z$ in $Z_c$ and sort the resulting NCF scores $\{\alpha = f(z) \mid z \in Z_c\}$ in descending order: $\alpha_1 \geq \cdots \geq \alpha_{|Z_c|}$.
5. Compute the NCF scores $\alpha_*^j = h(x_*, y^j)$ for the test input $x_*$ and each possible class label $j \in \{1, \ldots, c\}$. Then, compute the smoothed p value

$$p_*^j = \frac{|\{z_i \in Z_c : \alpha_i > \alpha_*^j\}|}{|Z_c| + 1} + \theta \frac{|\{z_i \in Z_c : \alpha_i = \alpha_*^j\}| + 1}{|Z_c| + 1}, \tag{4}$$

   where $\theta \in \mathcal{U}[0, 1]$ is a tie-breaking random variable. Note that $p_*^j$ is the portion of calibration examples that are at least as non-conforming as the tentatively labeled test example $(x_*, y^j)$, i.e., an empirical approximation of the p value of Equation 3.
6. Return the prediction region

$$\Gamma_*^\varepsilon = \{y^j \in Y : p_*^j > \varepsilon\}. \tag{5}$$

   together with the vector $(p_*^1, \ldots, p_*^c)$ of p values, one for each class.

Note that steps 1–4 are performed only once, while steps 5–6 are performed for each test point $x_*$.

*Non-conformity function.* A NCF function is well defined if it assigns low scores to correct predictions and high scores to wrong predictions. A natural choice for $h$, based on the discriminant model $f$, is $h(z) = \Delta(f(x_i), y_i)$, where $\Delta$ is a suitable distance[4]. Recall that, for an input $x \in X$, the output of $f$ is a vector of class likelihoods, which we denote by $f(x) = [f_1(x), \ldots, f_c(x)]$. In classification, a common well-defined NCF function is given by setting

$$\Delta(f(x), y) = 1 - f_y(x), \tag{6}$$

where $f_y(x)$ is the likelihood of class $y$ when the predictor $f$ is applied on $x$. If $F$ correctly predicts $y$ for input $x$, the corresponding likelihood $f_y(x)$ is high (the highest among all classes, see Definition 4) and the resulting NCF score is low. The opposite holds when $F$ does not predict $y$. The NCF measure chosen for our experiments (Eq. 6) preserves the ordering of the class likelihoods predicted by $f$.

*Prediction uncertainty.* A CP-based prediction region provides a set of plausible predictions with statistical guarantees, and as such, also captures the uncertainty about the prediction. Indeed, if CP produces a region $\Gamma_*^\varepsilon$ with more than one class, then the prediction for $x_*$ is *ambiguous* (i.e., multiple predictions are plausible), and thus, potentially erroneous. Similarly, if $\Gamma_*^\varepsilon$ is empty, then there are no plausible predictions at all, and thus, none can be trusted. The only reliable prediction is the one where $\Gamma_*^\varepsilon$ contains only one class. In this case, $\Gamma_*^\varepsilon = \{\hat{y}_*\}$, i.e., the region only contains the predicted class. This is always true for our NCF function. The proof is trivial and given in Appendix B.

The size of the prediction region is determined by the chosen significance level $\varepsilon$ and by the p values derived via CP. Specifically, from Eq. 5 we can see that, for levels $\varepsilon_1 \geq \varepsilon_2$, the corresponding prediction regions are such that $\Gamma^{\varepsilon_1} \subseteq \Gamma^{\varepsilon_2}$. It follows that, given a test input $x_*$, if $\varepsilon$ is lower than all its p values, i.e., if $\varepsilon < \min_{j=1,\ldots,c} \ p_*^j$, then the region $\Gamma_*^\varepsilon$ contains all the classes, and $\Gamma_*^\varepsilon$ shrinks as $\varepsilon$ increases. In particular, $\Gamma_*^\varepsilon$ is empty when $\varepsilon \geq \max_{j=1,\ldots,c} \ p_*^j$.

We are now ready to introduce our frequentist uncertainty measures, called confidence and credibility, which we define in terms of two p values, independently of the significance level $\varepsilon$. The intuition is that these two p values identify the range of $\epsilon$ values for which the prediction is reliable, i.e., $|\Gamma_*^\varepsilon| = 1$.

**Definition 6** (Confidence and credibility) Given a predictor $F$, the *confidence* of a point $x_* \in X$, denoted by $1 - \gamma_*$, is defined as:

$$1 - \gamma_* = \sup\{1 - \varepsilon : |\Gamma_*^\varepsilon| = 1\}, \tag{7}$$

and the *credibility* of $x_*$, denoted by $c_*$, is defined as:

$$c_* = \inf\{\varepsilon : |\Gamma_*^\varepsilon| = 0\}. \tag{8}$$

[4] The choice of $\Delta$ is not very important, as long as it is symmetric.

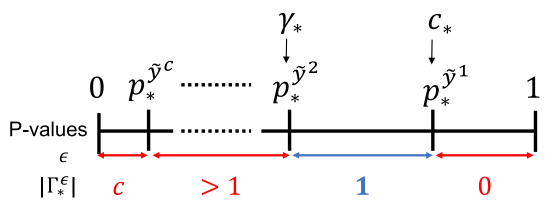

**Fig. 2** CP $p$ values over the [0, 1] interval and corresponding sizes of prediction interval. $\tilde{y}^i$ is the class with the $i$-th largest $p$ value, so $p_*^{\tilde{y}^1} = c_*$ and $p_*^{\tilde{y}^2} = \gamma_*$

Therefore, the so-called *confidence-credibility interval* $[\gamma_*, c_*)$ contains all the values of $\varepsilon$ such that $|\Gamma_*^\varepsilon| = 1$.

The confidence $1 - \gamma_*$ is the highest probability value for which the corresponding prediction region contains only $\hat{y}_*$, and thus it measures how likely (according to the calibration set) our prediction for $x_*$ is. In particular, $\gamma_*$ corresponds to the second largest p value. The credibility $c_*$ is the smallest level for which the prediction region is empty, i.e., no plausible prediction is found by CP. It corresponds to the highest $p$ value, i.e., the $p$ value of the predicted class. Figure 2 illustrates CP $p$ values and corresponding prediction region sizes. In binary classification problems, like our predictive monitoring problem, each point $x_*$ has only two $p$ values: $c_*$ ($p$ value of the predicted class) and $\gamma_*$ ($p$ value of the other class).

It follows that the higher $1 - \gamma_*$ and $c_*$ are, the more reliable the prediction $\hat{y}_*$ is, because we have an expanded range $[\gamma_*, c_*)$ of $\varepsilon$ values by which $|\Gamma_*^\varepsilon| = 1$. Indeed, in the degenerate case where $c_* = 1$ and $\gamma_* = 0$, then $|\Gamma_*^\varepsilon| = 1$ for any value of $\varepsilon < 1$. This is why, as we will explain in the next section, our uncertainty-based rejection criterion relies on excluding points with low values of $1 - \gamma_*$ and $c_*$. Hence, our frequentist uncertainty measure associates with each input its confidence and credibility values.

**Definition 7** (Frequentist uncertainty measure) Given a predictor $F$ with discriminant $f$, we define the frequentist uncertainty measure $u_f : X \to U_{\mathcal{F}} = [0,1]^2$ as the function mapping inputs $x$ into their corresponding confidence and credibility values, obtained as per Definition 6, i.e., $\forall x \in X, u_f(x) = (1-\gamma, c)$.

## 3.2 Bayesian neural networks

A neural network is a function $f_{\mathbf{w}} : X \to [0,1]^c$, which maps an input $x \in X$ into a vector of class likelihoods, $f_{\mathbf{w}}(x) = [f_{\mathbf{w}}^1(x), \ldots, f_{\mathbf{w}}^c(x)]$, depending on some parameters $\mathbf{w}$, namely weights and biases. A trained neural network is typically a complex but deterministic model. The core idea of Bayesian neural networks (BNNs) is to place a probability distribution over its parameters $\mathbf{w}$, thereby transforming the neural network into a stochastic model. The advantage of Bayesian methods, and BNNs in particular, is that they provide a distribution of predictions, called *predictive distribution*, rather than a single prediction like deterministic NNs. Such distribution captures both the aleatoric uncertainty, i.e., the noise inherent in the observations, and the epistemic uncertainty, i.e., model uncertainty about its prediction [21]. We will therefore leverage this predictive distribution (its mean and variance to be precise) to compute our Bayesian uncertainty measures.

The Bayesian learning process starts by defining a prior distribution for $\mathbf{w}$ that expresses our initial belief about the parameter values. As we observe data $Z' \sim \mathcal{Z}$, we update this prior to a posterior distribution using Bayes' rule:

$$p(\mathbf{w}|Z') = \frac{p(Z'|\mathbf{w})p(\mathbf{w})}{p(Z')}. \tag{9}$$

Note that placing a prior distribution over $\mathbf{w}$ is analogous to the random weight initialization required to train a traditional (deterministic) neural network. A common choice is to choose a zero-mean Gaussian prior.

Similarly, we assume that the conditional distribution $p(y|x)$ is a softmax likelihood

$$p(y = y^j|x, \mathbf{w}) = \frac{\exp(f_{\mathbf{w}}^j(x))}{\sum_{i=1}^c \exp(f_{\mathbf{w}}^i(x))}. \tag{10}$$

It follows that, given a set of i.i.d. observations $Z'$, the likelihood function can be expressed as

$$p(Z' \mid \mathbf{w}) = \prod_{(x_i,y_i)\in Z'} p(y_i|x_i, \mathbf{w}). \tag{11}$$

Note that, because of the nonlinearity introduced by the neural network function $f_{\mathbf{w}}(x)$ and by the softmax likelihood, the posterior $p(\mathbf{w}|Z')$ is non-Gaussian. Finally, in order to predict the value of the target for an unobserved input $x_*$, we marginalize the predictions with respect to the posterior distribution of the parameters, obtaining

$$p(\hat{y}_*|x_*, Z') = \int p(\hat{y}_*|x_*, \mathbf{w})p(\mathbf{w}|Z')d\mathbf{w}. \tag{12}$$

The latter is called *posterior predictive* distribution and it can be used to retrieve information about the uncertainty of a specific prediction $\hat{y}_*$. Unfortunately, the integration is analytically intractable due to the nonlinearity of the neural network function [20,22].

*Empirical approximation of the predictive distribution* . Assume, for the moment, that we are able to sample from the posterior distribution (9) and let $[w_1, \ldots, w_N]$ denote a vector of $N$ realizations of the random variable $\mathbf{w} \sim p(\mathbf{w}|Z')$. Each realization $w_i$ induces a deterministic function $f_{w_i}$ that can be evaluated at $x_*$, the unobserved input. The likelihood for a target $\hat{y}_*$ can be computed using (10). The empirical approximation of the predictive distribution (12) can be expressed as

$$p(\hat{y}_*|x_*, Z') \approx \sum_{i=1}^{N} \frac{1}{N} p(\hat{y}_*|x_*, w_i) = \sum_{i=1}^{N} \frac{1}{N} \left[ \frac{\exp(f^*_{w_i}(x_*))}{\sum_{j=1}^{c} \exp(f^j_{w_i}(x_*))} \right], \tag{13}$$

where $f^*_{w_i}$ denotes the component of $f_{w_i}$ corresponding to class $\hat{y}_*$. By the strong law of large numbers, the empirical approximation converges to the true distribution as $N \to \infty$ [23]. The sample size $N$ can be chosen, for instance, to ensure a given width of the confidence interval for a statistic of interest [24] or to bound the probability that the empirical distribution differs from the true one by at most some given constant [25].

*Bayesian inference techniques.* Since precise inference is infeasible, various approximate methods have been proposed to infer a BNN. We consider two approximate solution methods: Hamiltonian Monte Carlo and Variational Inference.

Let $w$ be a weight vector sampled from the posterior distribution $p(\mathbf{w}|Z')$, i.e., a realization of the random variable $\mathbf{w}$. We denote with $f_w(x)$ the corresponding deterministic neural network having weights fixed to $w$.

*Hamiltonian Monte Carlo* (HMC) [8] defines a Markov chain whose invariant distribution is exactly the posterior $p(\mathbf{w}|Z')$ . The Hamiltonian dynamics is used to speed up the space exploration. HMC does not make any assumption on the form of the posterior distribution, and is asymptotically correct. After convergence, HMC returns a trace of explored network weights $w_0, w_1, \dots, w_N$ that, all together, can be interpreted as an empirical approximation of the posterior $p(\mathbf{w}|Z')$. Controlling in a precise way the convergence rate and how well the chain explores the parameter space is, however, far from trivial.

*Variational Inference* (VI) [9] directly approximates the posterior distribution with a known parametric distribution $q(\mathbf{w}; \psi)$, typically a distribution easy to sample from. Its parameters $\psi$, called variational parameters, are learned by minimizing the Kullback–Leibler (KL) divergence between the proposed distribution and the posterior. The KL divergence between $q(\mathbf{w}; \psi)$ and $p(\mathbf{w}|Z')$ is defined as

$$KL(q(\mathbf{w}; \psi)||p(\mathbf{w}|Z')) = \int q(\mathbf{w}; \psi) \log \frac{q(\mathbf{w}; \psi)}{p(\mathbf{w}|Z')} d\mathbf{w}. \tag{14}$$

Since the posterior distribution is not known, a different objective function, called Evidence Lower Bound (ELBO), is introduced. It is defined as

$$ELBO_\psi = \mathbb{E}_{q(w;\psi)}\{\log p(Z'|\mathbf{w}) - KL(q(\mathbf{w}; \psi)||p(\mathbf{w}))\}. \tag{15}$$

ELBO [26]. In VI, the variational objective, i.e., the negative ELBO, becomes the loss function used to train of a Bayesian neural network [26]. A common choice for $q(\mathbf{w}; \psi)$ is the Gaussian distribution (where $\psi$ are its mean and variance).

The predictive distribution is a nonlinear combination of Gaussian distributions, and thus it is not Gaussian. However, samples can be easily extracted from $q(\mathbf{w}; \psi)$, which allows us to obtain an empirical approximation of the predictive distribution.

*Prediction uncertainty.* Having showed how to derive empirical approximations of the predictive distribution (either with HMC or with VI) we can now extract statistics to characterize the latter. We stress that the predictive distribution, and hence its statistics, effectively captures prediction uncertainty. Our Bayesian uncertainty measure is, therefore, given by the empirical mean and variance of the predictive distribution.

**Definition 8** (Bayesian uncertainty measure) Given observations $Z' \sim \mathcal{Z}$ and a Bayesian neural network $f_\mathbf{w}$ with $\mathbf{w} \sim p(\mathbf{w}|Z')$, we define the Bayesian uncertainty measure $u_f : X \to U_\mathcal{B} \subseteq \mathbb{R}^2$ as the function mapping inputs $x$ into the empirical mean and variance of the predictive distribution $p(\hat{y} \mid x, Z')$ (12). Formally, $\forall x \in X, u_f(x) = (\mu, \sigma^2)$.

***Remark 3*** (Softmax probabilities as uncertainty measures). A popular method to assess the quality of a prediction is to use the probabilities outputted by the softmax layer of the DNN (i.e., the output of the discriminant function). However, previous works have shown that such probabilities are not well calibrated, meaning that the probability associated with the predicted class label does not reflect its ground-truth correctness likelihood [27,28]. For example, in a binary classification problem, one can consider the difference between the probability of the two classes as a measure of uncertainty, where small differences indicate uncertain predictions. Previous experiments in [29] have shown that such a measure yields poor error detection in NPM, because the measure is overconfident in predictions that turn out being erroneous.

Such observation supports our claim that more principled and calibrated methods to measure uncertainty are needed. On one hand, our uncertainty measure based on Conformal Prediction overcomes this limitation, as it makes predictions with statistical evidence, rather than probabilistic evidence. On the other hand, our Bayesian measure of uncertainty remains consistent also in regions where no data has been observed and, in particular, where the deterministic DNN will behave almost randomly.

## 4 Uncertainty-based rejection criteria

In this section, we show how to leverage the measures of uncertainty introduced in Sect. 3 to learn an optimal, uncertainty-based error detection rule for reachability predictions, thereby solving Problem 2.

The rationale is that an unseen input $x$ must have sufficiently low uncertainty values in order for the prediction to be accepted. However, manually determining such decision boundaries on the uncertainty domain $U$ is a non-trivial task. As discussed in Sect. 2, optimal error detection thresholds can be automatically identified by solving an additional supervised learning problem.

For a type of error $e \in \{pe, fp, fn\}$, given a set of validation inputs $X_v$, sampled from $\mathcal{X}$, and an uncertainty measure $u_f$, we build a *validation set* $W_v^e$, defined as

$$W_v^e = \{(u_f(x), \mathbf{1}(e(x))) \mid x \in X_v\}. \tag{16}$$

The inputs of $W_v^e$, referred to as $U_v$, are the uncertainty measures evaluated on $X_v$:

$$U_v = \{u_f(x) \mid x \in X_v\}. \tag{17}$$

Similarly, each input $u_f(x) \in U_v$ is then labeled, respectively, with 1 or 0 depending, respectively, on whether or not the classifier $f$ makes an error of type $e$ on $x$:

$$E_v^e = \{\mathbf{1}(e(x)) \mid x \in X_v\}. \tag{18}$$

For ease of notation, in the following we omit the type of error considered. However, it is important to keep it in mind when addressing a specific application.

We seek to find those uncertainty values that optimally separate the points in $U_v$ in relation to their classes, that is, separate points yielding errors from those that do not. Finding such a separation corresponds to finding a (sub-)optimal solution to Problem 2.

As for the uncertainty measures of the previous section, we present two alternative solutions to this problem. The first one leverages support vector classification (SVC) and applies to the frequentist case, based on CP, where the uncertainty values are given by confidence and credibility. The second solution leverages Gaussian process classification (GPC) and applies to the Bayesian case, based on BNNs, where uncertainty values are given by mean and standard deviation of the predictive distribution.

***Remark 4*** (Highly unbalanced dataset). In predictive monitoring, the neural network-based reachability predictors that we use have typically very high accuracy (see results in Sect. 6). Therefore, the dataset $W_v$ is typically highly unbalanced, as it contains more examples of correct classifications (class 0) than of classification errors (class 1). In binary classification problems, in particular in both SVC and GPC, accuracy can be a misleading measure when dealing with imbalanced datasets. Indeed, for instance, a constant function mapping any input into the most frequent class will have high accuracy. Therefore, a model trained on accuracy maximization tends to misinterpret the behavior of the observations belonging to the minority class, causing misclassification. However, in our method, the less frequent class (i.e., the prediction errors) is actually the most interesting one, which we want to classify correctly.

### 4.1 Frequentist error detection via support vector classification (SVC)

For data parsimony, since calibration points were not used to train the reachability predictor, we build the validation set $W_v$ from the calibration set $Z_c$. A cross-validation strategy is used to compute values of confidence and credibility for points in $Z_c$. The cross-validation strategy consists of removing the $j$-th score, $\alpha_j$, in order to compute $\gamma_j$ and $c_j$, i.e., the p values at $x_j \in X_c$, where $X_c = \{x \mid \exists y \in Y : (x, y) \in Z_c\}$. In this way, we can compute our frequentist uncertainty measure given by confidence $1 - \gamma$ and credibility $c$ for every point in the calibration set. The support vector classifier (SVC) is then trained on pairs $((1 - \gamma, c), e)$, where $e$ indicates whether or not a prediction error is observed.

In a nutshell, SVC is a kernel-based method that maps the original data into a new space, called feature space, via a feature map $\phi$. By doing so, patterns that are not linearly separable in the original data can be converted to be linearly separable in the feature space [20]. The linear decision boundary for a binary SVC is defined as

$$d(x) = a \cdot \phi(x) + b = 0, \tag{19}$$

for $x$ in the original space. Training a SVC reduces to find the values of $a$ and $b$ that maximize the margin around the separating hyperplane and the decision function $d(x)$. In the dual formulation of the SVC problem, whose details are out of the scope of this paper, the optimization is performed using kernel functions rather than feature maps. Recall that a kernel can be defined as $k(u, u') = \phi(u)^T \cdot \phi(u')$. In practice, given a test point $x_*$ with predicted label $\hat{y}_*$ and uncertainty measure $u_f(x_*) = (1 - \gamma_*, c_*)$, error detection at $x_*$ boils down to evaluating the learned SVC, i.e., its decision function $d$, at $u_f(x_*)$. If $d(u_f(x_*)) > 0$ the point $x_*$ is classified as potentially erroneous, class 1, and 0 otherwise.
*Tuning of SVC hyperparameters.* A simple method to handle imbalanced classes in SVC is via cost-sensitive learning [30]. The aim is to find the classifier that minimizes the mean predictive error on the training set. Each misclassified example by a hypothetical classifier contributes differently to the

error function. One way to incorporate such costs is the use of a penalty matrix, which specifies the misclassification costs in a class dependent manner [31]. We design an empirical penalty matrix $\mathcal{P}$, as follows: the $(i, j)$-th entry of $\mathcal{P}$ gives the penalty for classifying an instance of class $i$ as class $j$. Of course, when $i = j$, the penalty is null. The penalty matrix for dataset $W_v$ is defined as

$$\mathcal{P} = \begin{bmatrix} 0 & \frac{q}{2r_e(q-n_e)} \\ \frac{r_e q}{2n_e} & 0 \end{bmatrix}, \tag{20}$$

where $n_e$ is the number of points belonging to class 1 in $W_v$, $r_e$ is a parameter influencing how many errors we are willing to accept and $q = |W_v|$. The term $\frac{r_e q}{2n_e}$, which represents the penalty for wrongly classifying an error as correct, increases as $n_e$ decreases. Note that, when $r_e = 1$ and the dataset is perfectly balanced ($q = 2n_e$), the penalties are equal: $\frac{r_e q}{2n_e} = \frac{q}{2r_e(q-n_e)} = 1$. Further, if $r_e > 1$, the penalty term increases, leading to more strict rejection thresholds and higher overall rejection rates. On the contrary, if $r_e < 1$, the penalty decreases, leading to possibly missing some errors.

**Definition 9** (Frequentist error detection criterion) Given a state $x_* \in X$, a reachability predictor $f$ and an uncertainty measure $u_f(x_*) = (1 - \gamma_*, c_*)$ as per Definition 7, the frequentist error detection function $G_f : U_{\mathcal{F}} \to \{0, 1\}$ rejects a reachability estimate $F(x_*)$, i.e., $G_f(1 - \gamma_*, c_*) = 1$, if and only if

$$d(1 - \gamma_*, c_*) > 0,$$

where $d$ is the binary SVC of (19) trained on $W_v$ (16).

## 4.2 Bayesian error detection via Gaussian process classification (GPC)

Recall that we define our Bayesian uncertainty measure as the mean and variance of the empirical approximation of the BNN predictive distribution. Therefore, the prediction $\hat{y}_*$ made on an unseen input $x_*$ is associated with a vector of uncertainty $(\mu_*, \sigma_*^2) \in U_{\mathcal{B}} \subset \mathbb{R}^2$. To keep the approach fully Bayesian, we propose a probabilistic solution to the error detection problem based on Gaussian Processes [32].

Formally, a *Gaussian Process (GP)* is a stochastic process, i.e., a collection of random variables indexed by some input variable, in our case $u \in U$, such that every finite linear combination of them is normally distributed. In practice, a GP defines a distribution over real-valued functions of the form $\ell : U_{\mathcal{B}} \to \mathbb{R}$ and such distribution is uniquely identified by its mean and covariance functions, respectively, denoted by $m(u) = \mathbb{E}[\ell(u)]$ and $k(u, u')$. The GP can thus be denoted as $\mathcal{GP}(m(u), k(u, u'))$. This means that the function value at any point $u$, $\ell(u)$, is a Gaussian random variable with mean $m(u)$ and variance $k(u, u)$. Typically, the covariance function $k(\cdot, \cdot)$ depends on some hyperparameter $\gamma$.

As mentioned before, GPs can be used to perform probabilistic binary classification, i.e., learning a Bayesian classifier $G_f : U_{\mathcal{B}} \to \{0, 1\}$ from a set of observations. GPs model the posterior probabilities by defining latent functions $\ell : U_{\mathcal{B}} \to \mathbb{R}$, whose output values are then mapped into the $[0, 1]$ interval by means of a so-called link function $\Phi$. Typically, in binary classification problem, the logit or the probit function are used as $\Phi$.

Given an input $u_i$, let $\ell_i = \ell(u_i)$ denote its latent variable, i.e., the latent function $\ell$ evaluated at $u_i$. Also denote $l_v = [\ell(u_i) \mid u_i \in U_v]$, where $U_v$ is a set of input points defined as per 17. From $U_v$, it is possible to compute the mean vector $m_v$ of the GP, by evaluating the mean function $m(\cdot)$ at every point in the set, and the covariance matrix $K_v^\gamma$, by evaluating the covariance function on every pair of points in the set: $m_v = [m(u_i) | u_i \in U_v]$ and $K_v^\gamma = [k_\gamma(u_i, u_j) | u_i, u_j \in U_v]$.

The first step of a GPC algorithm is to place a GP prior over the latent function $\ell$, defined by

$$p(\ell | U_v) = \mathcal{N}(\ell | m_v, K_v^\gamma).$$

Let now consider a test input $u_*$ with latent variable $\ell_*$. In order to do inference, that is, predict its label $e_*$, we have to compute

$$p(e_* = 1 | u_*, U_v, E_v) = \int \Phi(\ell_*) p(\ell_* | u_*, U_v, E_v) d\ell_*, \tag{21}$$

where $E_v$, see Eq. 18, denotes the set of labels corresponding to points in $U_v$, see Eq. 17. The discriminant function $g_f : U_{\mathcal{B}} \to [0, 1]^2$ of the error detection classifier $G_f$ is defined as

$$g_f(u_*) = [1 - p(e_* = 1 | u_*, U_v, E_v), p(e_* = 1 | u_*, U_v, E_v)]. \tag{22}$$

To compute Eq. (21), we have to marginalize the posterior over the latent Gaussian variables:

$$p(\ell_* | u_*, U_v, E_v) = \int p(\ell_* | u_*, U_v, \ell_v) p(\ell_v | U_v, E_v) d\ell_v, \tag{23}$$

where the posterior $p(\ell_v | U_v, E_v)$ can be obtained using the standard Bayes rule

$$p(\ell_v | U_v, E_v) = \frac{p(E_v | \ell_v, U_v) p(\ell_v | U_v)}{p(E_v | U_v)}.$$

Therefore, performing inference reduces to solving two integrals (Eqs. 21 and 23). In classification, the first integral is not available in closed form since it is the convolution of a Gaussian distribution, $p(\ell_v)$, and a non-Gaussian one, $p(E_v|\ell, U_v)$. Hence, we have to rely on approximations in order to compute and integrate over the posterior $p(\ell_v|E_v)$. In our experiments, we use the Laplace method, which provides a Gaussian approximation $q(\ell_v|E_v)$ of the posterior $p(\ell_v|E_v)$, which can then be easily computed and integrated over.

*Tuning of GPC hyperparameters.* In the prior distribution, the covariance for the latent variables depends on some hyperparameters $\gamma$. A classical strategy to select the optimal values for such parameters is to find the values of $\gamma$ that maximize the marginal likelihood, which, intuitively, measures how likely the data are, given a certain value of $\gamma$. However, as mentioned in Remark 4, the marginal likelihood may be a poor choice because class 1 is very little represented. An alternative solution is to compute, for different values of $\gamma$, the confusion matrix of the GPC on the training set $W_v$. The entries of such matrix can be used to define more clever measures of performance that apply to binary classification. In our experiments, we use the true positive rate (TPR), as it is well-suited for datasets presenting a strong disproportion. TPR measures the fraction of points in class 1 that have been correctly classified:

$$TPR := \frac{TP}{TP+FN}, \tag{24}$$

where $TP$ indicates the number of true positives and $FN$ indicates the number of false negatives. Alternative measures, such as the Matthews correlation coefficient (MCC) [33], may apply. Note that, during the training phase, the GPC assigns to each point the class with highest likelihood.

Another key step is the following. The discriminant function, $g_f$, returns a vector containing the probability of belonging to each of the two classes. However, such probabilities might not separate well in cases of highly unbalanced datasets. Therefore, choosing the class with the highest probability, as per Definition 4, may lead to bad performance. Therefore, after the GPC has been trained with an optimal value for $\gamma$, it may be useful to find the decision threshold that maximizes the GPC accuracy on the training set $W_v$. This can be done, for instance, using the ROC curve. In other words, we classify as correct (class 0) only those points that have an extremely high probability of belonging to that class. We do so by searching for the threshold $\tau$ that maximizes the quantity $TPR - FPR$, i.e., the proportion of recognized errors minus the proportion of points wrongly rejected. Below we provide the formal definition of the Bayesian error detection function for a generic threshold $\tau$.

**Definition 10** (Bayesian error detection criterion) Given a state $x_* \in X$, a reachability predictor $f$, a Bayesian uncertainty measure $u_f(x_*) = (\mu_*, \sigma_*^2) = u_*$ as per Definition 8, and a decision threshold $\tau \in [0, 1)$, the error detection function $G_f : U_\mathcal{B} \to \{0, 1\}$ rejects a reachability estimates $F(x_*)$ if and only if

$$p(e_* = 1 \mid u_*, U_v, E_v) > \tau,$$

where $U_v$, $E_v$ are the inputs and outputs of the validation set, see (17) and (18).

## 5 Active learning

Recall that we are dealing with two related learning problems: learning a prediction rule (i.e., a reachability predictor) using the training set $Z_t$, and learning a rejection rule using the validation set $W_v$ (via learning an adequate uncertainty measure first).

As the accuracy of a classifier increases with the quality and the quantity of observed data, adding samples to $Z_t$ will generate a more accurate predictor, and similarly, adding samples to $W_v$ will lead to more precise error detection. Ideally, one wants to maximize accuracy while using the least possible amount of additional samples, because obtaining labeled data is expensive (in NPM, labeling each sample entails solving a reachability checking problem), and the size of the datasets affects the complexity and the dimension of the problem. Therefore, to improve the accuracy of our learning models efficiently, we need a strategy to identify the most “informative” additional samples.

For this purpose, we propose an *uncertainty-aware active learning* solution, where the retraining points are derived by first sampling a large pool of unlabeled data, and then considering only those points where the current predictor $f$ is still uncertain. The criterion used to decide whether a point is uncertain enough to be considered informative is indeed our rejection rule $R_f$. In particular, recall that the uncertainty function $u_f : X \to U$ maps input states to their level of uncertainty, and the error detection function $G_f : U \to \{0, 1\}$ maps uncertainty values to a binary class interpreted as accepting/rejecting a prediction. The rejection rule $R_f : X \to \{0, 1\}$, introduced in Sect. 2, is defined as the combination of these two functions, $R_f = G_f \circ u_f$. Therefore, such rejection rule provides an effective uncertainty-based query strategy. Points rejected by $R_f$, i.e., points whose predictions are expected to be erroneous, are indeed the most uncertain ones.

The proposed active learning method should reduce the overall number of erroneous predictions, because it improves the predictor on the inputs where it is most uncertain, and, as a consequence, also reduces the overall rejection rate.

However, it cannot be excluded in general that the retraining process introduces new prediction errors. We stress that, with our method, these potential new errors can be effectively detected.

## 5.1 General active learning algorithm

Our active learning algorithm works as follows. The rejection rule $R_f$ is used as a query strategy to identify, from a batch of randomly selected unlabeled points, those with a high degree of uncertainty. We then query the oracle, i.e., the HA reachability checker, to label such uncertain points, and finally we divide them into two groups: one group is added to the training set $Z_t \subseteq X \times Y$, producing the augmented dataset $Z_t^a$; the other is added to the validation set $Z_v$, producing $Z_v^a$. The set of uncertain points must be divided according to the splitting probability used to originally divide $Z'$ into $Z_t$ and $Z_v$. The first step consists in retraining the reachability predictor on $Z_t^a$. Let $f_a$ denote the new predictor. The second step requires extracting the augmented validation dataset $W_v^a$ from $Z_v^a$. To do so, we must first train a new uncertainty measure $u_{f_a}$ so as to reflect the new predictor $f_a$. Then, the new error detection rule $G_{f_a}$ is trained on

$$W_v^a = \{(u_{f_a}(x), \mathbf{1}(e_{f_a}(x)) \mid x \in X_v^a\},$$

where $e_{f_a}$ is the error predicate introduced in Sect. 2 (the $f_a$ index is added to stress dependency on the updated predictor). In conclusion, this process leads to an updated rejection rule $R_{f_a} = G_{f_a} \circ u_{f_a}$, which is expected to have a reduced rate of incorrect rejections.

We now describe in detail our uncertainty-aware active learning algorithm, which given an initial training set $Z_t$, a predictor with discriminant $f$ trained on $Z_t$, an initial validation set $Z_v$, and a rejection rule $R_f$ trained on $Z_v$, computes an enhanced predictor $f_a$ and enhanced rejection rule $R_{f_a}$ as follows.

**Algorithm 1** Active Learning algorithm

```
Inputs: training set $Z_t$, validation set $Z_v$, predictor $f$, uncertainty function $u_f$, rejection rule $R_f$, maximum iterations $n_{it}$.
Outputs: enhanced predictor $f_a$, enhanced rejection rule $R_f^a$.

repeat $n_{it}$ times:
  // Select retraining inputs
  1. Randomly sample a set of input points.
  2. Identify the subset $A$ of points rejected based on $R_f$.
  // Derive augmented datasets
  3. Invoke the reachability oracle to label the points in $A$.
  4. Divide the data into two groups and add them, respectively, to $Z_t$
     and $Z_v$, obtaining an augmented training set, $Z_t^a$, and an augmented
     validation set, $Z_v^a$.
  5. Train a new predictor $f_a$ from $Z_t^a$.
  6. Build the training set $W_v^{f_a}$ using $Z_v^a$ and $f_a$.
  7. Train a new error detection rule $G_{f_a}$, using $u_f$ and the method of
     Section 4, and obtain the enhanced rejection rule $R_{f_a}$.
  8. $Z_t \leftarrow Z_t^a$, $Z_v \leftarrow Z_v^a$, $f \leftarrow f_a$, $R_f \leftarrow R_f^a$.
end
```

The above algorithm is used as-is in the Bayesian framework. For the frequentist case, we present a refined version of the algorithm that overcomes issues with the sensitivity of CP-based measures.

*Sensitivity of CP-based uncertainty measures.* The distribution of calibration scores depends both on the case study at hand and on the trained classifier. If such a classifier $F$ has high accuracy, then most of the calibration scores $\alpha_1, \ldots, \alpha_q$ will be close to zero. Each p value $p_*^j$ of an unseen test point $x_*$ counts the number of calibration scores greater than $\alpha_*^j$, the non-conformity score for label $j$ at $x_*$. Credibility, which is the p value associated with the class predicted by $F$, is expected to have a small score and therefore a high p value. On the contrary, $\gamma$, which is the p value associated with the other (non-predicted) class, is expected to have a larger score. However, given the high accuracy of $F$, the number of calibration scores significantly greater than zero is very small. Therefore, the fraction of calibration scores determining $\gamma$ is not very sensitive to changes in the value of $\alpha_*$, which is determined by $f(x_*)$. On the contrary, credibility is extremely sensitive to small changes in $\alpha_*$. In general, the sensitivity of confidence with respect to $\alpha_*$ increases as the accuracy of $f$ decreases, and vice versa for credibility. Figure 3 shows the credibility landscapes for two different training instances of model $f$ on the same training set for a concrete case study. We observe that even if regions where misclassifications take place are always assigned low credibility values, outside those regions credibility values are subject to high variance.

This sensitivity results in an over-conservative rejection criterion, leading to a high rejection rate and in turn, to an inefficient query strategy. However, if we enrich the calibration set using additional samples with nonzero $\alpha$-scores, we can reduce such sensitivity, thereby making credibility more robust with respect to retraining. This process is illustrated in Figure 3, where the additional nonzero $\alpha$-scores (right) lead to a more robust credibility landscape, where low-credibility regions are now more tightly centered around areas of misclassification.

Observing that samples with uncertain predictions will have nonzero $\alpha$-scores[5], we will use the original rejection rule to enrich the calibration set, thereby deriving a refined rejection rule and in turn, a refined and more effective query strategy for active learning.

[5] The $\alpha$-score of a sample $(x_i, y_i)$ is zero only if $f_{y_i}(x_i) = 1$.

## 5.2 Frequentist active learning algorithm

Provided that the credibility measure is extremely sensitive in our application, we found that dividing the frequentist active learning algorithm in two phases dramatically improves performances. In the first phase, we refine the query strategy: we use the current rejection rule $R_f$ to identify a batch of uncertain points, temporarily add these points to the calibration set, thereby obtaining an updated, more robust, rejection rule that we use as a query strategy. In the second phase, we simply perform an active learning iteration, i.e., steps 2–5 above) but using the refined query strategy to identify the retraining inputs.

We now describe the details of this variant of the active learning algorithm designed for the frequentist framework. Given an initial training set $Z_t$, a prediction rule $f$ trained on $Z_t$, an initial calibration set $Z_c$, a rejection rule $R_f$ trained on $Z_c$ and a rejection ratio $r_e$, we proceed as follows.

**Algorithm 2** Frequentist AL algorithm

*Inputs*: training set $Z_t$, calibration set $Z_c$, predictor $f$, uncertainty function $u_f$, rejection rule $R_f$, maximum iterations $n_{it}$.
*Outputs*: enhanced predictor $f_a$, enhanced rejection rule $R_f^a$.

```
repeat $n_{it}$ times:
  // Refine the query strategy
  1. Randomly sample a set of input points.
  2. Identify the subset $Q$ of points rejected by $R_f$.
  3. Identify the subset $A$ of points rejected based on $R_f$.
  4. Invoke the reachability oracle to label the points in $Q$.
  5. Define a query set $Z_Q$ by adding these points to $Z_c$.
  6. Obtain an updated rejection rule $R_f^Q$ from $Z_Q$ using the method of
     Section 4.
  // Active phase
  7. Randomly sample a set of input points.
  8. Identify the subset $A$ of points rejected by $R_f^Q$.
  9. Invoke the reachability oracle to label the points in $A$.
  10. Divide the labeled data into two groups and add them, respectively,
      to $Z_t$ and $Z_c$, obtaining an augmented training set, $Z_t^a$, and an
      augmented calibration set, $Z_c^a$.
  11. Train a new predictor $f_a$ from $Z_t^a$.
  12. $Z_t \leftarrow Z_t^a$, $Z_c \leftarrow Z_c^a$, $f \leftarrow f_a$, $R_f \leftarrow R_f^a$.
end
```

It is important to observe that, in order for the active learning algorithm to preserve the statistical soundness of conformal prediction, the augmented training and calibration sets $Z_t^a$ and $Z_c^a$ must be sampled from the same distribution. This is guaranteed by the fact that, in the active learning phase, we add new points to both the training and the calibration dataset, and these points are sampled from the same distribution (in particular, we apply the same random sampling method and same rejection criterion). The only caveat is ensuring that the ratio between the number of samples in $Z_c$ and $Z_t$ is preserved on the augmented datasets.

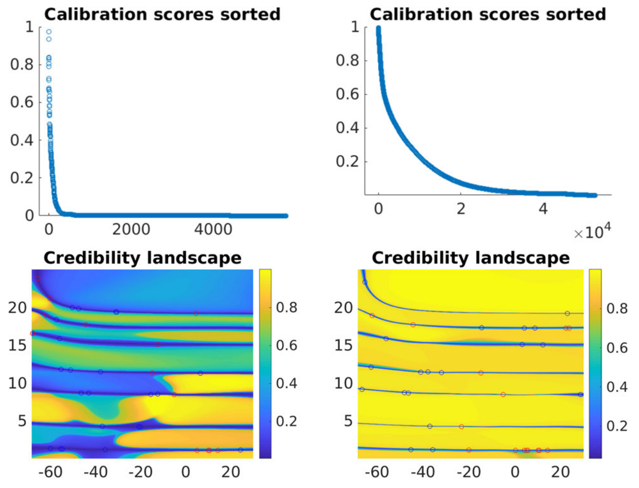

**Fig. 3** Credibility values in the spiking neuron case study. Calibration scores (first row) and credibility landscapes using the initial calibration set $Z_c$ (left column) versus the query set $Z_Q$ (right column). The landscapes are obtained for different instances of the predictor $f$, trained on the same dataset $Z_t$

## 6 Experimental results

We experimentally evaluate the proposed method and compare the frequentist and Bayesian approaches on a benchmark of six hybrid system models with varying degrees of complexity. We consider four deterministic case studies: the model of the spiking neuron (SN) action potential [6] introduced in Sect. 2 and the classic inverted pendulum (IP) on a cart, which are two-dimensional models with nonlinear dynamics, the artificial pancreas (AP) [34], which is a six-dimensional nonlinear model, and the helicopter model (HC) [6], which is a linear model with 29 state variables. In addition, we analyze two non-deterministic models with nonlinear dynamics: a cruise controller (CC) [6], whose input space has four dimensions, and a triple water tank (TWT)[6], which is a three-dimensional model. Details of the case studies are available in Appendix C.

### 6.1 Experimental settings

The experiments were performed on a computer with a CPU Intel x86, 24 cores and a 128GB RAM.

Table 1 compares the performances of DNN and BNN against different types, respectively, deterministic and Bayesian, of classifiers. In particular, in the deterministic case we compare: a sigmoid DNN (**DNN-S**) with 3 hidden

[6] http://dreal.github.io/benchmarks/networks/water/

**Table 1** Empirical accuracy of the state classifiers for each case study. Values are in percentage.

| | IP | AP | CC | TWT | HC | SN |
|---|---|---|---|---|---|---|
| **DNN-S** | **99.84** | **99.56** | **99.92** | **99.91** | 98.33 | **99.78** |
| **SNN** | 99.74 | 99.49 | 99.91 | 99.81 | **98.96** | 99.51 |
| **DNN-R** | 99.61 | 99.41 | 99.91 | 99.82 | 98.75 | 97.59 |
| **SVM** | 98.85 | 99.17 | 99.50 | 99.32 | 96.54 | 67.94 |
| **RF** | 99.66 | 96.61 | 99.19 | 99.24 | 91.67 | 99.51 |
| **NBOR** | 99.66 | 96.61 | 99.19 | 99.24 | 91.67 | 98.43 |
| **BNN-VI** | 99.47 | 99.29 | 99.49 | 99.56 | **99.32** | **99.45** |
| **BNN-HMC** | 99.12 | **99.63** | **99.88** | **99.77** | 97.79 | 98.87 |
| **GP** | **99.80** | 99.61 | 99.86 | 99.76 | 96.16 | 98.43 |
| **BLR** | 57.85 | 97.68 | 97.96 | 87.16 | 90.14 | 56.72 |

For each model, the best result for deterministic classifiers and the best result for Bayesian classifiers are highlighted in bold

layers of 10 neurons each, tanh activations for the hidden layers and Sigmoid function for the output layer; a shallow NN (**SNN**) of 20 neurons; a ReLU DNN (**DNN-R**), with 3 hidden layers of 10 neurons each and rectified linear unit (ReLU) activations for all layers; a support vector machine with radial kernel (**SVM**); a random forest classifier (**RF**); and a k-nearest neighbors classifier (**NBOR**). For the Bayesian case, we compare: a Bayesian NN (**BNN**) with 3 hidden layers of 10 neurons each, standard Gaussian priors, trained with both variational inference (**BNN-VI**) and Hamiltonian Monte Carlo (**BNN-HMC**) methods; a Gaussian Process (**GP**); and a Bayesian Logistic Regression model (**BLR**). In the deterministic framework, differences in accuracy values are relatively small, even though the DNNs outperform the other classifiers in all case studies but HC.

In the Bayesian scenario, we observe that GP and BNN have comparable performances on the simplest models. However, BNN works better as soon as the dimension of the system increases. Furthermore, BNNs offer better scalability than GPs. Indeed, the scalability of GP inference depends heavily on the size of the dataset $n$, with a time complexity of $O(n^3)$, whereas for BNNs with VI this is $O(n \cdot m)$, where $m$ is the number of epochs, and for BNN with HMC the complexity is $O(n \cdot k)$, where $k$ is the number of steps of the Markov chain. In our experiments $m \ll k$, and $k$ and $n$ have same order of magnitude. On the other hand, BLR shows limited performances for systems whose dynamics is intrinsically nonlinear. In general, despite the overall difference in performance may seem small, we would like to stress that we target safety-critical applications, for which we seek accuracies as close as possible to 100% and even small improvements become important.

Motivated by the results presented in Table 1, we choose the sigmoid DNN architecture described above for our reachability predictions. In particular, the output of the DNN with parameters $w$ in a state $x \in X$, denoted by $f_w(x)$, is the likelihood of class 1, i.e., the likelihood that the hybrid automaton state is positive. Therefore, the discriminant function $f$ evaluated at $x$ returns a vector of probabilities $f(x) = [1 - f_w(x), f_w(x)]$. To avoid overfitting, we did not tune the architecture (i.e., number of neurons and hidden layers) to optimize the performance for our data and, for the sake of simplicity, we choose the same architecture for all the case studies, as we found no specific DNN architecture with consistently better performances. See Appendix D for a detailed performance analysis for different choices of the DNN architecture. In particular, we use the same architecture for deterministic DNNs and their Bayesian counterpart.

The entire pipeline is implemented in Python, and the neural networks are trained with TensorFlow [35] and PyTorch [36]. More precisely, Keras [37], a Python deep learning library, is used to train the deterministic DNN, Edward [38], a Python library for probabilistic modeling built on TensorFlow, is used to train the BNN with HMC inference, and Pyro [39], a probabilistic programming library built on PyTorch, is used to train the BNN with VI. The source code for all the experiments can be found at the following link: https://github.com/francescacairoli/NPM.

For every model, we generate an initial dataset $Z'$ of 20K samples and a test set $Z_{test}$ of 10K samples. The helicopter model is the only exception, where, due to the higher dimensionality, a set $Z'$ of 100K samples is used. Both $Z'$ and $Z_{test}$ are drawn from the same distribution $\mathcal{Z}$; see [6] and Appendix E for more details on how data are labeled and on the distributions for each case study. The training and validation sets are two subsets of $Z'$ extracted as follows: a sample $z \in Z'$ has probability $s$ of falling into $Z_t$ and probability $1 - s$ of falling into $Z_v$, where $s = 0.7$ is the splitting ratio. Recall that the calibration set $Z_c$ of the frequentist approach coincides with the validation set $Z_v$ and that we use the same splitting rate $s$ when augmenting the datasets during active learning. The dReal solver [12] is used as a reachability oracle to label the datasets for the non-deterministic case studies. For the deterministic case studies, we used an HA simulator implemented in MATLAB.

*Kernel choice.* Both error detection rules, based on SVC for the frequentist approach and GPC for the Bayesian approach, are kernel-based methods. The radial basis function (RBF) kernel has been chosen in both cases, as it outperforms the polynomial and linear kernels. The RBF is defined as $k_\gamma(u, u') = \exp(\gamma \|u - u'\|^2)$, where $\|u - u'\|^2$ is the squared Euclidean distance between the two input vectors.

*Error type selection.* Below we focus on detecting all kinds of prediction error, including false positives and false negatives. However, it is possible—and this is a very useful feature of our approach—to focus on a specific type of error. For example, in safety-critical applications, one could focus on detecting false negatives, which are the most critical kind of errors. An alternative solution, which we explored in [10], is to learn two distinct rejection rules, one for false positives and one for false negatives, and combine them into a global rejection rule that suits the case study at best.

In addition, in the frequentist case, one can tune the SVC penalty matrix $\mathcal{P}$ (see equation (20)) to penalize specific kinds of errors. For instance, setting $r_{fn} > 1$ will result in a detection rule that is stricter on recognizing false negatives. In the Bayesian case, the GPC decision threshold can be tuned by maximizing scores other than $TPR - FPR$.

*Tuning of the Bayesian approach.* Training the deterministic DNNs was straightforward in our experiments. All models share the same initialization settings and all reach an extremely high accuracy (always higher than 99%, see Table 3). On the contrary, training a Bayesian neural network requires a careful tuning of the inference hyperparameters, e.g., the choice of prior distributions and the sample size used to empirically approximate the predictive distribution. Furthermore, in the HMC framework, the parameters governing the Hamiltonian dynamics affect the capabilities of the Monte Carlo algorithm to explore and to eventually converge. In the VI framework, the hyperparameters by which we maximize the ELBO (15) may change from one model to the other. The main drawback is that this may limit the effectiveness of the active learning framework, as explained later in this section.

## 6.2 Performance measures

We want our method to be capable of working at runtime, which means it must be extremely fast in making predictions and deciding whether to trust them. We emphasize that the time required to train the reachability predictor and the error detection rule does not affect its runtime efficiency, as it is performed in advance (offline) only once. Also, we do not want an over-conservative rejection rule, as unnecessary rejections would reduce effectiveness of our predictive monitor[7].

Keeping that in mind, the relevant performance metrics for NPM are the *accuracy of the reachability predictor* $F$, the *error detection rate* (or *recognition rate*) and the overall *rejection rate* of the rejection rule $R_f$. The error detection rate measures the proportion of errors made on the test set by $F$ that are actually recognized by $R_f$, whereas, the rejection rate measures the overall proportion of test points rejected by $R_f$. Clearly, we want our method to be reliable and thus, detect the majority of prediction errors (high detection rate) without being overly conservative, i.e., keeping a low rejection rate.

Another important remark is about the interaction of the two classifiers, $F$ and $G_f$: as the accuracy of $F$ increases it commits fewer errors, which makes it harder for the detection rule $G_f$ to learn how to capture them because the validation set $W_v$ for training $G_f$ will contain few examples of prediction errors. The opposite holds as well: if $F$ performs poorly, it produces a less unbalanced validation set $W_v$, which may result in a more accurate rejection rule $R_f$. These two behaviors are balanced against one another, as discussed above, by tuning the training of the rejection rules.

## 6.3 Computational performance

*Offline cost.* Training an NPM requires the following steps: (i) training the state classifier, (ii) generating the datasets $W_v$, which requires computing the uncertainty values for each point in $Z_v$, and (iii) training the error rejection rule. All these steps are performed offline. Executing the entire pipeline, i.e., learning a working NPM, when $|Z'| = 20\text{K}$, takes around 3 minutes in the frequentist case and around 11 minutes in the Bayesian case. When $|Z'| = 100\text{K}$, it takes around 6.5 (120–190) minutes in the frequentist (Bayesian) case. The time required to execute 20K VI epochs is comparable with the time required to perform 2K HMC steps (see Table 2, bottom-left frame).

*Online cost.* Given a test input $x_*$, it takes from 1.4 up to 31 milliseconds to evaluate the NPM, i.e., to make a prediction and choose whether to accept it or not (see Table 2, top frame). Importantly, this time does not depend on the dimension or dynamics of the hybrid system. However, in the Bayesian approach it depends on the number of observations used to empirically approximate the predictive distribution, which is a fixed cost, whereas, in the frequentist approach the evaluation time is affected by the size of the calibration set $Z_c$, which may increase as we add observations[8]. On this aspect, the query strategy refinement we propose for active learning ensures that the augmented calibration set is as small as possible, which translates into runtime efficiency of our method.

*Active learning overhead (offline).* Active learning carries two additional training costs: the time needed to compute uncertainty values for a large pool of data, and the time the oracle needs to compute labels for the most uncertain points. The latter dominates, especially for non-deterministic systems, since they require fully fledged reachability checking, which is more expensive than simulation of a deterministic system. Therefore, if the rejection rate is relatively high and we consider a large pool of randomly selected points,

[7] Defining countermeasures to a rejected prediction is out of the scope of our method, but these may include switching to a fail-safe mode of the system or querying the HA model checker for the true reachability value. Both cases consistently affect the runtime efficiency of our monitor.

[8] The size of $Z_c$ affects only the computation of the uncertainty measures, which reduces to computing two p values (confidence and credibility). Each p-value is derived by computing a non-conformity score, which has same cost as evaluating the state classifier, and one search over the array of calibration scores.

**Table 2** **(Top)**Online computational costs: time to evaluate the NPM, i.e., time to obtain a reachability prediction and decide whether to trust it, on a single state

| **Online costs** (ms) | | | | **Offline costs** (min) | | | | **AL overhead** (min) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **CP** | **VI** | **HMC** | **Model** | **CP** | **VI** | **HMC** | **CP** | **VI** | **HMC** |
| IP | 1.4 | 13.0 | 8.1 | IP | 2.0 | 8.9 | 14.5 | 9.6 | 32.4 | 27.8 |
| AP | 2.1 | 15.0 | 7.9 | AP | 2.5 | 11.6 | 9.3 | 10.0 | 30.4 | 16.4 |
| CC | 2.0 | 14.0 | 8.2 | CC | 2.9 | 12.2 | 14.1 | 57.9 | 95.0 | 135.5 |
| TWT | 2.1 | 14.0 | 8.1 | TWT | 3.1 | 12.8 | 11.0 | 21.2 | 74.4 | 207.6 |
| HC | 4.0 | 31.0 | 14.5 | HC | 6.5 | 194.0 | 121.0 | 26.0 | 593.0 | 917.1 |
| SN | 1.6 | 15.0 | 8.3 | SN | 4.5 | 9.9 | 15.2 | 11.4 | 29.9 | 57.5 |

Time is measured in milliseconds (ms). **(Bottom)** Offline computational costs: time required to initially train the NPM. AL overhead: time to complete an active learning iteration (on average). Time is measured in minutes

the procedure may take long. The pool of new inputs has indeed to be large in order to have good exploration and find significant instances. As we will show experimentally, our uncertainty-aware active learning approach results in a more precise rejection rule with a lower rejection rate. Therefore, the time spent for offline retraining pays off in improving the online performance of the NPM.

In our study, the pool used to refine the query strategy (required only with CP) contains 50K samples, whereas the pool used for the active learning phase contains 100K samples. In particular, one iteration of the active learning procedure takes, for the simplest deterministic models, around 10 minutes in the frequentist scenario, and around 20 minutes in the Bayesian scenario (both VI and HMC approaches). The helicopter model needs longer time, as it is trained for a higher number of epochs: it takes around 1 hour in the frequentist case against the 10 hours of the Bayesian case. For the non-deterministic models (triple water tank and cruise controller), an active learning iteration takes approximatively the same time of a simple deterministic model, except for the overhead introduced in labeling new points (see Table 2: bottom-right frame). dReal, the non-deterministic reachability checker, takes around 1.5 mins to label 100 observations of the CC model and around 4 mins to label the same amount of points of the TWT model. In general, the time required for a single active learning iteration is expected to decrease for subsequent iterations, as the rejection rate will be lower (leading to fewer retraining samples). Note that retraining is performed offline and does not affect runtime performance of our approach.

## 6.4 Experiments

We evaluate our approach on three configurations: the *initial* configuration, where the predictor $F$ and error detection rule $G_f$ are derived via supervised learning; the *active* configuration, where the initial models are retrained via our uncertainty-aware active learning; and the *passive* configuration, where the initial models are retrained using a uniform sampling strategy to augment the dataset and with the same number of observations of the active configuration. In this way, we can evaluate the benefits of employing an uncertainty-based criterion to retrain our models.

**Table 3** NSC accuracy, error detection rate and rejection rate in initial configuration

| **Model** | | **NSC Acc.** | **♯ Err.** | **Det. rate** | **Rej. rate** |
|---|---|---|---|---|---|
| **IP** | CP | 99.55 | 45 | 100.00 | 5.37 |
| | VI | 99.72 | 28 | 83.81 | 1.46 |
| | HMC | 99.03 | 97 | 96.70 | 9.68 |
| **AP** | CP | 99.64 | 36 | 100.00 | 4.97 |
| | VI | 99.30 | 70 | 97.17 | 5.36 |
| | HMC | 98.35 | 165 | 96.73 | 9.34 |
| **CC** | CP | 99.88 | 12 | 100.00 | 3.63 |
| | VI | 99.25 | 75 | 91.15 | 3.67 |
| | HMC | 98.16 | 184 | 98.04 | 7.75 |
| **TWT** | CP | 99.81 | 19 | 100.00 | 4.40 |
| | VI | 99.58 | 42 | 74.92 | 1.06 |
| | HMC | 98.13 | 187 | 93.30 | 6.56 |
| **HC** | CP | 99.43 | 57 | 97.21 | 6.12 |
| | VI | 97.37 | 263 | 84.24 | 13.55 |
| | HMC | 97.66 | 234 | 91.60 | 16.03 |
| **SN** | CP | 99.79 | 21 | 100.00 | 3.95 |
| | VI | 98.47 | 153 | 85.76 | 6.04 |
| | HMC | 98.69 | 131 | 99.24 | 21.84 |

Each block denotes a different case study. CP indicates the frequentist approach; VI and HMC indicate the two inference techniques used in the Bayesian approach

Table 3 presents the experimental performances (on the test set $Z_{test}$) in the initial configuration. The results are averaged over five runs, where in each run, we resample $Z_t$ and $Z_c$ from $Z'$ and retrain $F$. Table 4 compares the performances of the three configurations, only for one run in this case.

*Frequentist approach.* The average NSC accuracy over the six case studies is 99.68%. The rejection criterion recognizes well almost all the errors, with an average error detection rate of 99.53%, but the overall rejection rate in the initial configu-

**Table 4** Comparison of initial, active and passive approaches

| | | NSC Acc. | ♯ Err. | Det. rate | Rej. rate |
|---|---|---|---|---|---|
| *Inverted Pendulum (IP)* | | | | | |
| CP | Initial | 99.79 | 21 | 100.00 | 5.24 |
| | Active | 99.87 | 13 | 100.00 | 3.12 |
| | Passive | 99.79 | 21 | 100.00 | 6.07 |
| VI | Initial | 99.58 | 42 | 80.95 | 1.68 |
| | Active | 99.58 | 42 | 85.71 | 1.31 |
| | Passive | 99.71 | 29 | 75.81 | 1.13 |
| HMC | Initial | 87.45 | 1255 | 100.00 | 24.72 |
| | Active | 99.15 | 85 | 87.06 | 4.17 |
| | Passive | 98.49 | 151 | 100.00 | 13.02 |
| *Artificial Pancreas (AP)* | | | | | |
| CP | Initial | 99.61 | 39 | 100.00 | 4.17 |
| | Active | 99.83 | 17 | 100.00 | 1.65 |
| | Passive | 99.62 | 38 | 100.00 | 5.48 |
| VI | Initial | 99.29 | 71 | 95.77 | 5.17 |
| | Active | 99.71 | 29 | 96.55 | 1.75 |
| | Passive | 99.47 | 53 | 100.00 | 3.29 |
| HMC | Initial | 98.95 | 105 | 98.09 | 8.32 |
| | Active | 99.38 | 62 | 90.32 | 4.17 |
| | Passive | 95.12 | 488 | 100.00 | 26.79 |
| *Cruise Controller (CC)* | | | | | |
| CP | Initial | 99.85 | 15 | 100.00 | 3.46 |
| | Active | 99.96 | 4 | 100.00 | 0.51 |
| | Passive | 99.88 | 12 | 100.00 | 5.15 |
| VI | Initial | 99.01 | 99 | 97.98 | 5.53 |
| | Active | 99.84 | 16 | 100.00 | 1.25 |
| | Passive | 99.74 | 26 | 100.00 | 1.46 |
| HMC | Initial | 97.22 | 278 | 99.64 | 8.41 |
| | Active | 99.47 | 53 | 94.34 | 3.14 |
| | Passive | 95.75 | 425 | 97.03 | 8.42 |
| *Triple Water Tank (TWT)* | | | | | |
| CP | Initial | 99.82 | 18 | 100.00 | 5.87 |
| | Active | 99.96 | 4 | 100.00 | 0.70 |
| | Passive | 99.81 | 19 | 100.00 | 4.43 |
| VI | Initial | 99.60 | 40 | 77.50 | 1.11 |
| | Active | 99.67 | 33 | 84.85 | 1.61 |
| | Passive | 99.50 | 50 | 75.40 | 20.32 |
| HMC | Initial | 97.50 | 250 | 96.80 | 5.04 |
| | Active | 99.20 | 80 | 95.00 | 3.70 |
| | Passive | 91.86 | 814 | 52.58 | 11.31 |
| *Helicopter (HC)* | | | | | |
| CP | Initial | 99.21 | 79 | 95.95 | 6.75 |
| | Active | 99.49 | 51 | 94.12 | 4.52 |
| | Passive | 99.40 | 60 | 95.00 | 5.92 |
| VI | Initial | 98.14 | 186 | 88.71 | 13.64 |
| | Active | 98.90 | 110 | 92.86 | 1.98 |
| | Passive | 98.66 | 134 | 87.54 | 9.94 |
| HMC | Initial | 97.74 | 226 | 89.82 | 14.71 |
| | Active | 97.74 | 223 | 73.01 | 7.06 |
| | Passive | 97.77 | 223 | 65.47 | 6.40 |
| *Spiking Neuron (SN)* | | | | | |
| CP | Initial | 99.61 | 39 | 100.00 | 4.17 |
| | Active | 99.83 | 17 | 100.00 | 1.65 |
| | Passive | 99.62 | 28 | 100.00 | 5.48 |
| VI | Initial | 98.18 | 182 | 91.21 | 7.91 |
| | Active | 98.20 | 180 | 91.11 | 6.26 |
| | Passive | 98.20 | 180 | 98.52 | 14.57 |
| HMC | Initial | 98.32 | 168 | 74.40 | 9.89 |
| | Active | 98.89 | 111 | 87.38 | 5.91 |
| | Passive | 98.21 | 179 | 74.86 | 14.85 |

Results are over a single run. Legend is as in Table 3

ration is around 5%, a non-negligible amount. Table 4 shows that the passive learning approach provides little improvement: the NSC accuracy is similar to the initial one and the rejection rate is still relatively large. However, the active approach provides a significant improvement: the overall rejection rate and the number of errors made by the NSC falls dramatically, while preserving the ability of detecting almost all errors (with an error detection rate of 100%, except for the helicopter). In particular, rejection rates span from 3.46% to 6.75% with the initial rejection rule, but drop to between 0.51% and 4.52% after active learning, and the average NSC accuracy increases from 99.68% (initial) to 99.82% (active).

*Bayesian approach.* The predictive distribution is approximated by samples of 100 observations. The BNN priors are chosen to be standard normal distributions. In the initial configuration, the NSC accuracy, averaged over all the case studies, is 98.95% with VI and 98.27 with HMC. The rejection criterion recognizes on average 86.18% of errors with VI and 95.27% with HMC. The overall rejection rate is approx. 5.19% with VI, spanning from 1.06% to 13.55%, and approx. 9.87% with HMC, spanning from 6.56% to 16.03%.

Table 4 shows that, in the HMC framework, the passive learning approach happens to produce results that are even worse than the initial configuration. The reason might be that that once the training sets are extended with data that may come from a distribution different from $\mathcal{X}$, the set of hyperparameters chosen to optimally solve the initial problem may become sub-optimal. Indeed, when the hyperparameters were tuned again, specifically for the passive learning dataset, high performances were reached. For instance, in the TWT model, the initial HMC performance rates (obtained with proper hyperparameter tuning) are: 97.5% accuracy, 96.8% recognition and 5.04% rejection. Passive learning (without re-tuning the hyperparameters) causes a significant drop in performance: 91.86% accuracy, 52.58% recognition and 11.31% rejection. However, once the HMC hyperparameters are tuned specifically for the passive learning dataset, the level of performance gets back to the initial one: 98.59% accuracy, 96.45% recognition and 3.63% rejection. On the other hand, active learning still yields improvements: the NSC accuracy rises from 98.27% to 99.30%, and the rejection rates, initially very high, are significantly reduced, which unfortunately causes a slight decrease in the error detection accuracy. The recognition rate falls from 95.27% to 91.68%.

We finally observe that VI outperforms inference via HMC, even though VI is not able to reach recognition rates as high as in the frequentist approach. In particular, on average, the initial VI approach yield an NSC accuracy of 98.95%, a rejection rate of 5.19% and a recognition rate of 86.18%. The passive results, as before, introduce only minor improvements, whereas active learning yields a significant reduction in the rejection rate (from 5.19% to 2.36%), an increase in the NSC accuracy (from 98.95% to 99.32%) and an increase in the overall recognition rate (from 86.175% to 91.85%).
*Discussion.* A likely reason why the Bayesian approach falls behind the frequentist one is that the former introduces several levels of approximation. Indeed the BNN is trained using either VI or HMC, two approximate inference techniques, resulting in an approximation of the true posterior distribution. Moreover, the resulting uncertainty measures are defined by statistics of said distribution (mean and variance in our case), which introduces an additional error as these measures do not retain full information about the BNN posterior. The latter error propagates as we apply GP classification for error detection, which produces another approximate solution.

However, it is not to say that the Bayesian approach does not work well overall. Indeed, Bayesian NPM is capable of recognizing always at least the 85% of the prediction errors and the accuracy of the predictive monitor is always well above 98%. Moreover, we expect the Bayesian solution to work better in settings with noise and partial observability, settings where the Bayesian approach should provide a more robust performance than the frequentist one.

Another interesting aspect is that active learning seems to enhance the classifier confidence in its predictions, as demonstrated by an improved detection rate and a sensibly reduced rejection rate. This is the main advantage of active learning, besides providing a higher state classifier accuracy.

In summary, the main conclusions from our experimental analysis are:

- Our reachability predictors attain high accuracies, consistently above 97.37% (above 99.43% for the frequentist case).
- The frequentist approach overall outperforms the Bayesian ones in all metrics and configurations, followed by VI.
- Error detection rates stay approximately constant after retraining (active or passive). The frequentist approach achieves staggering performance on this metric.
- The benefits of active learning are visible from an overall reduction of the rejection rate and an overall increase in the NSC accuracy.

### 6.5 CP over the error detection rule

Recall that our frequentist error detection rule $G_f$ builds on CP to quantify the reliability of the NSC predictions. In principle, CP can be applied to derive prediction regions with statistical guarantees to any supervised learning model. The SVC $G_f$ for error detection is no exception.

In this experiment, we show that we can apply CP to produce prediction regions $\Gamma^\varepsilon$ for $G_f$, which, by definition, contain the correct rejection decision with probability $1 - \varepsilon$. For this purpose, we apply CP to the SVC $G_f^a$ obtained after one active learning iteration. In particular, we derive from $\Gamma^\varepsilon$ a so-called *risky* rejection strategy, aimed at reducing the rejection rate: we reject only those points by which the prediction region for $G_f^a$ contains only class 1, that is, when rejecting is the only plausible decision (according to CP). We report the results for two case studies, the helicopter and the artificial pancreas, to show how the performances of the risky rejection strategy compare to the ones obtained via active learning. These two models are representative of cases where active learning could sensibly reduce the rejection rate (artificial pancreas) and where instead it could not (helicopter).

Note that applying CP to $G_f^a$ requires a new calibration set, i.e., a set of points from $W_v^a$ not used to train $G_f^a$ but rather to calibrate its predictions. The CP framework needs a few further adjustments, discussed in detail in Appendix A.

Choosing an optimal value for $\varepsilon$, i.e., one that yields high detection rates and low rejection rates, is non-trivial and requires problem-specific tuning. In Fig. 4, we compare the above introduced risky strategy against the initial one at different $\varepsilon$ levels and after one active learning iteration (results are reported only for the HC and AP case studies). We observe that, with a properly tuned $\varepsilon$, we can achieve the same detection rate of the initial approach (this occurs for $\varepsilon \in [0.057, 0.1]$ in the AP model, and $\varepsilon \in [0.03, 0.11]$ in the HC model), but at the same time a lower rejection rate. For instance, a sweet spot that most reduces the rejection rate without sacrificing detection is $\varepsilon = 0.03$ for the HC and $\varepsilon = 0.057$ for the AP.

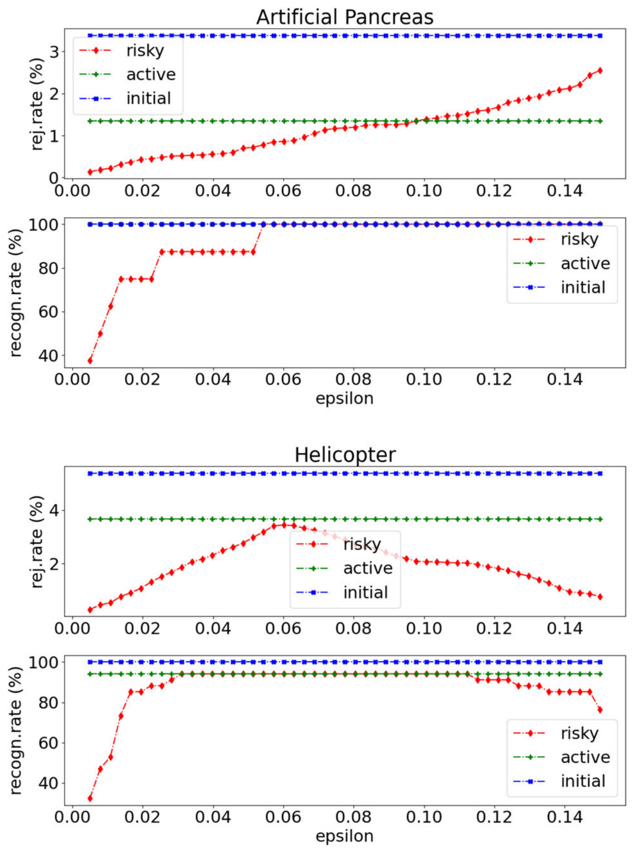

**Fig. 4** Rejection rate and recognition rate of initial rejection rule (initial), the rule obtained after one active learning iteration (active), and the "risky" rule obtained by applying CP to the latter (risk)

## 7 Related work

A number of methods have been proposed for online reachability analysis that rely on separating the reachability computation into distinct offline and online phases. However, these methods are limited to restricted classes of models [3,40], or require handcrafted optimization of the HA's derivatives [41], or are efficient only for low-dimensional systems and simple dynamics [42].

In contrast, NSC [6] is based on learning DNN-based classifiers, is fully automated and has negligible computational cost at runtime. In [43,44], similar techniques are introduced for neural approximation of Hamilton–Jacobi (HJ) reachability. Our methods for prediction rejection and active learning are independent of the class of systems and the machine-learning approximation of reachability, and thus can also be applied to neural approximations of HJ reachability.

In [45], Yel and others present a runtime monitoring framework that has similarities with our NPM approach, in that they also learn neural network-based reachability monitors (for UAV planning applications), but instead of using, like we do, uncertainty measures to pin down potentially erroneous predictions, they apply NN verification techniques [46] to identify input regions that might produce false negatives. Thus, their approach is complementary to our uncertainty-based error detection, but, due to the limitations of the underlying verification algorithms, they can only support deterministic neural networks with sigmoid activations. On the contrary, our techniques support any kind of ML-based monitors, including probabilistic ones.

The work of [47,48] addresses the predictive monitoring problem for stochastic black-box systems, where a Markov model is inferred offline from observed traces and used to construct a predictive runtime monitor for probabilistic reachability checking. In contrast to NSC, this method focuses on discrete-space models, which allows the predictor to be represented as a look-up table, as opposed to a neural network.

In [49], a method is presented for predictive monitoring of STL specifications with probabilistic guarantees. These guarantees derive from computing prediction intervals of ARMA/ARIMA models learned from observed traces. Similarly, we use CP which also can derive prediction intervals with probabilistic guarantees, with the difference that CP supports any class of prediction models (including auto-regressive ones). In [50], model predictions are used to forecast future robustness values of MTL specifications for runtime monitoring. However, no guarantee, statistical or otherwise, is provided for the predicted robustness. Deshmukh and others [51] have proposed an interval semantics for STL over partial traces, where such intervals are guaranteed to include the true STL robustness value for any bounded continuation of the trace. This approach can be used in the context of predictive monitoring but tends to produce over-conservative intervals.

A related approach to NSC is smoothed model checking [52], where Gaussian processes [32] are used to approximate the satisfaction function of stochastic models, i.e., mapping model parameters into the satisfaction probability of a specification. Smoothed model checking leverages Bayesian statistics to quantify prediction uncertainty, but faces scalability issues as the dimension of the system increases. In contrast, computing our measure of prediction reliability is very efficient, because it is nearly equivalent to executing the underlying predictor.

In the field of computer security, a machine learning-based method for malware detection [27] is conceptually very similar to our NPM method. The authors develop a tool for assessing the performance of a classifier using the statistical guarantees of conformal predictions with a self-trained mechanism to filter out unreliable classification decisions.

Literature on uncertainty-based active learning in deep-learning models is small and sparse, mainly because deep learning methods rarely represent model uncertainty, and state-of-the-art deep learning techniques require large amounts of data, which makes active learning impractical. Several uncertainty-based acquisition functions (i.e., functions used to rank the informativeness of new observations for active learning) are reviewed in [21]. In [53], a Deep Ensemble active learning method is proposed, where uncertainty is estimated from a stochastic ensemble of BNN models (obtained via MC-Dropout, another approximate Bayesian inference technique). Some applications of BNN in active learning are presented in [54,55]. In these works, however, the decision threshold used to identify informative samples is always chosen empirically. An important contribution of our work is the automatic tuning of the decision rule.

A basic application of conformal predictors in active learning is presented in [56]. Our approach introduces three important improvements: a more flexible and meaningful combination of confidence and credibility values, automated learning of rejection thresholds (which are instead fixed in [56]), and refinement of the query strategy.

In [29], we presented a preliminary version of the frequentist approach. In [10], we added to it an automated and optimal method to select the rejection thresholds and an active learning framework.

## 8 Conclusion

We have presented neural predictive monitoring, an approach for runtime predictive monitoring of hybrid systems that complements reachability predictions with principled estimates of the prediction uncertainty. NPM uses these estimates to derive optimal rejection criteria that identify potentially erroneous predictions without knowing the true reachability values. We have further designed an active learning strategy that, leveraging such uncertainty-based rejection criteria, increases the accuracy of the reachability predictor and reduces the overall rejection rate. Our approach overcomes the computational footprint of reachability checking (infeasible at runtime), while improving on traditional runtime verification by being able to detect future violations in a preemptive way.

We have devised two alternative solution methods for NPM. The first one follows a frequentist approach, with state classifiers expressed as deterministic DNNs and rejection rules expressed as support vector classifiers, where the rejection rules are optimized to detect unreliable predictions from uncertainty measures derived via Conformal Prediction. The second one follows a Bayesian approach, with a probabilistic state classifier based on Bayesian neural networks, rejection rules given as Gaussian process classifiers, and uncertainty measures extracted from statistics of the BNN predictive distribution.

The strengths of our NPM technique are its effectiveness in identifying and rejecting prediction errors and its computational efficiency: executing the classifier and the rule take on the order of milliseconds. NPM's efficiency is not directly affected by the complexity of the system under analysis but only by the complexity of the underlying learning problem and classifier.

Our experimental evaluation demonstrates that the frequentist approach outperforms the Bayesian one: the state classifier is simpler to train and faster to evaluate, and the error detection criteria are more accurate. Regarding BNN inference, we found that VI scales better than HMC with respect to the dimension of the system. The assumptions on the prior and the necessary tuning of hyperparameters represent important drawbacks of the Bayesian techniques, which, however, tend to be more consistent than deterministic models in regions with no data observed: here the posterior distribution will typically have high variance.

Among directions for future work, we plan to extend our approach to support more complex and real-world systems that include noise and partial observability, settings where the Bayesian approach can potentially provide more robust performance than the frequentist one.

**Acknowledgements** This work has been partially supported by the UK NCSC project "PM-CPS" n. 4217549, the Italian PRIN project "SEDUCE" n. 2017TWRCNB, the German DFG "multimode" project, U.S. ONR grant N00014-20-1-2751 and U.S. NSF grants CCF-1954837, CPS-1446832, and CCF-1918225.

## Appendix A conformal predictions on SVC

### A.1 Mondrian approach in conformal predictions

CP works as explained in Sect. 3.1. Given a test point $x_*$, we compute a $p$ value for every possible class and, given a significance level $\varepsilon$, the prediction region $\Gamma_*^{\varepsilon}$ is the set of labels whose p value exceeds the significance level. In practice, the prediction region for each test point can be interpreted as the set of classes that guarantees that the true class is not in the set no more than a fraction $\varepsilon$ of the times. This is called the *validity property*. It provides a statistical guarantee on the expected number of errors, i.e., number of times the true label is not in the prediction set. The validity property, as stated above, guarantees an error rate over all possible labels, not on per-label basis. The latter can be achieved with a CP variant, called label-conditional CP, which is in turn a variant of the Mondrian CP approach. The only change is in the calculation of the p values. The p value associated with class $y^j$ on a test point $x_*$ is defined as:

$$p_*^j = \frac{|\{z_i \in Z_c : y_i = y^j, \alpha_i > \alpha_*^j\}|}{|\{z_i \in Z_c : y_i = y^j\}| + 1} + \tag{25}$$

$$+ \theta \frac{|\{z_i \in Z_c : y_i = y^j, \alpha_i = \alpha_*^j\}| + 1}{|\{z_i \in Z_c : y_i = y^j\}| + 1}. \tag{26}$$

In words, we consider only the $\alpha_i$ corresponding to examples with the same label $y^j$ as the hypothetical label that we are assigning at the test point.

Label-conditional validity [57] is extremely important when the CP is applied to an unbalanced dataset, as in our case $W_v$. It has been shown empirically, that, with the plain validity property, the overall error rates tend to the chosen significance level, but the minority class are disproportionally affected by errors. The Mondrian approach ensures that, even for the minority class, the expected error rate will tend to the chosen significance level $\varepsilon$. We refer the reader to the existing literature [58,59] for further details.

## A.2 SVC non-conformity measure

CP relies on the definition of a non-conformity measure, which captures the extent to which a given input data conforms to the associated class. Every classification algorithm has a specific non-conformity measure, that makes the CP algorithm perform well in each framework. SVC is a kernel-based method that transforms the original data by mapping them into a new space, called feature space, via a feature map $\phi(x)$. By doing so, patterns that are not linearly separable, can be converted to be linearly separable in the feature space [20]. The linear decision boundary for a binary SVC is defined as $d(x) = a \cdot \phi(x) + b = 0$, for $x$ in the original space. Support vectors are the data points that lie closest to the decision hyper-plane. SVC maximizes the margin around the separating hyperplane and the decision function, which depends only on the support vectors. For ease of notation, we are assuming $Y = \{-1, 1\}$, rather than $\{0, 1\}$. The distance of a point $x_*$ from the separating hyperplane of the SVC is given by:

$$d_*^h = \frac{|d(x_*)|}{||a||},$$

where $d(x_*)$ is SVC decision function $d(\cdot)$ evaluated in $x_*$ and $||a||$ is the weighted sum of the support vectors. Then,

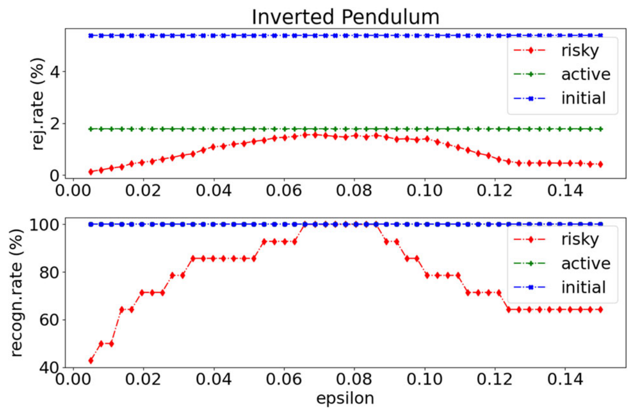

**Fig. 5** Results as in Figure 4 for the IP model. A good choice for $\varepsilon$ is around 0.08

the distance to the margin boundary of the class under consideration is given by:

$$d_*^m = \frac{|d(x_*)| - 1}{||a||}.$$

The non-conformity measure is thus defined as

$$\alpha_*^j = \exp(-d_*^m).$$

Such definition is presented in [60]. In case of transductive CP (TCP), the NCM can be derived directly from the value of Lagrange multipliers associated with the support vectors, as proposed in [59].

***Remark 5*** When CP are applied to the SVC, the input space is the uncertainty domain U, rather than X. However, for ease of presentation, we stick to the p values notation introduced in Section 3.1.

## Appendix B Proofs

**Proposition 1** *For the NCF function (6), if $\Gamma_*^\varepsilon = \{y^{j_1}\}$, then $y^{j_1} = F(x_*)$.*

***Proof*** Suppose by contradiction that $\Gamma_*^\varepsilon = \{y^{j_1}\}$ and $y^{j_1} \neq F(x_*) = y^{j_2}$. Then, by Equation 5, this implies that $p_*^{j_1} > \varepsilon$ and that $p_*^{j_2} \leq \varepsilon$, i.e., $p_*^{j_1} > p_*^{j_2}$. In turn, this implies that the corresponding NCF scores are such that $\alpha_*^{j_1} \leq \alpha_*^{j_2}$ (the inequality is not strict due to the tie-braking factor $\theta$ in Equation 4). But according to the definition of our NCF function (6), this means that $f_{y^{j_1}}(x_*) \geq f_{y^{j_2}}(x_*)$, i.e., that the likelihood of the non-predicted class $y^{j_1}$ is not below than that of the predicted class $y^{j_2}$, which, by Definition 4 and the assumption of well-formed discriminant, is a contradiction. □

## Appendix C Models and case studies

We briefly introduce the case studies used in our experimental evaluation.

*Spiking Neuron.* This model describes the evolution of a neuron's action potential. It is a deterministic HA with two continuous variables, one mode, one transition and nonlinear polynomial dynamics. We consider the unsafe set $D$ defined by $v_2 \leq 68.5$, expressing that the neuron should not undershoot its resting potential. The time bound for the reachability property is $T = 20$.

*Inverted Pendulum.* We consider the classic inverted pendulum on a cart nonlinear system. We consider the unsafe set $D$ defined by $|\theta| > \pi/4$, corresponding to the safety property that keeps the pendulum within 45° of the vertical axis. The time bound is $T = 5$.

*Artificial Pancreas* For the AP model, the unsafe set $D$ corresponds to hypoglycemia states, i.e., $D = BG \leq 3.9$ mmol/L, where $BG$ is the blood glucose variable. The state distribution considers uniformly distributed values of plasma glucose and insulin. The insulin control input is fixed to the basal value. The time bound is $T = 240$.

*Triple Water Tank* For the TWT model, $D$ is given by states where the water level of any of the tanks falls outside a given safe interval $I$, i.e., $D = \vee_{i=1}^{3} x_i \notin I$, where $x_i$ is the water level of tank $i$. The state distribution considers water levels uniformly distributed within the safe interval. The time bound is $T = 1$.

*Cruise Control.* It is a non-deterministic HA with 3 continuous variables, 6 modes, 11 transitions, and nonlinear polynomial dynamics. The unsafe set $D$ is defined by $v \leq -1$, which expresses that the vehicle's speed should not be below a reference speed by $1m/s$ or more. The reachability time bound is $T = 10$.

*Helicopter Controller.* We augment the 28-variable helicopter controller available on SpaceEx website[9] with a variable $z$ denoting the helicopter's altitude. The dynamics of $z$ is given by $\dot{z} = v_z$ , where $v_z$ is the vertical velocity and represented by variable $x_8$. The unsafe set $D$ is defined by $z \leq 0$. The time bound is $T = 5$. Since this model is large and publicly available on SpaceEx website, we do not provide the details here.

### C.1 Spiking neuron

We consider the spiking neuron model on the Flow* website[10]. It is a hybrid system with one mode and one jump.

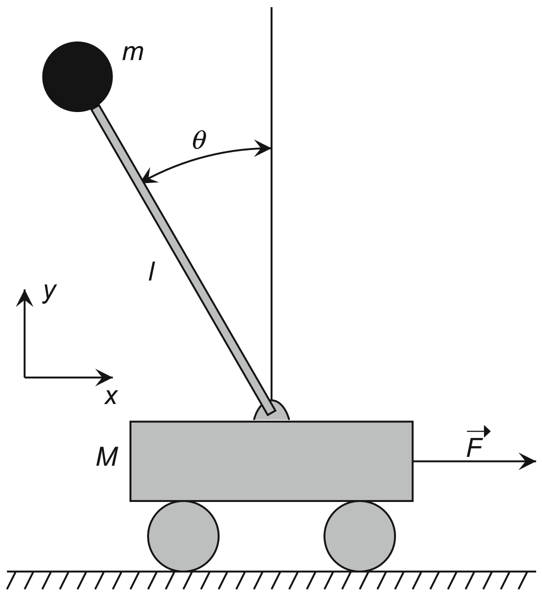

Fig. 6 Schematic of the inverted pendulum on a cart. Source: https://en.wikipedia.org/wiki/Inverted_pendulumWikipedia

The dynamics is defined by the ODE

$$\begin{cases} \dot{v_2} &= 0.04v_2^2 + 5v_2 + 140 - v_1 + I \\ \dot{v_1} &= a \cdot (b \cdot v_2 - v_1) \end{cases}. \tag{27}$$

The jump condition is $v_2 \geq 30$, and the associated reset is $v_2' := c \wedge v_1' := v_1 + d$, where, for any variable $x$, $x'$ denotes the value of $x$ after the reset.

The parameters are $a = 0.02$, $b = 0.2$, $c = -65$, $d = 8$, and $I = 40$ as reported on the Flow* website. We consider the unsafe state set $D = \{(v_2, v_1) \mid v_2 \leq 68.5\}$. This corresponds to a safety property that can be understood as the neuron does not undershoot its resting-potential region of $[-68.5, -60]$. The domain for sampling is $68.5 < v_2 \leq 30 \wedge 0 \leq v_1 \leq 25$. The time bound for the reachability property was set to $T = 20$.

### C.2 Inverted pendulum

We consider the control system for an inverted pendulum on a cart. This is a classic, widely used example of a nonlinear system. As shown in Fig. 6, the control input $F$ is a force applied to the cart with the goal of keeping the pendulum in upright position, i.e., $\theta = 0$. The dynamics is given by

$$J \cdot \ddot{\theta} = m \cdot l \cdot g \cdot \sin(\theta) - m \cdot l \cos(\theta) \cdot F \tag{28}$$

where $J$ is the moment of inertia, $m$ is the mass of the pendulum, $l$ is the length of the rod, and $g$ is the gravitational acceleration.

[9] http://spaceex.imag.fr/.

[10] https://flowstar.org/examples/.

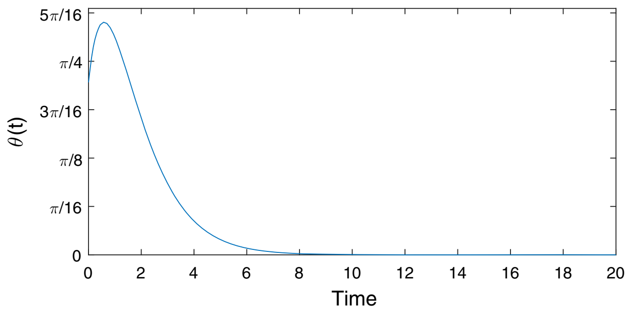

**Fig. 7** An evolution of the inverted pendulum state variable $\theta$ from initial state $(\theta_0, \omega_0) = (0.5, 1.0)$

We set $J = 1, m = 1/g, l = 1$, and let $u = F/g$. Eq. 28 becomes

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \sin(\theta) - \cos(\theta) \cdot u \end{cases} \quad . \tag{29}$$

We consider the control law of Eq. 30. Figure 7 shows an evolution of $\theta$ under this control law. We consider the unsafe state set $D = \{(\theta, \omega) \mid \theta < -\pi/4 \vee \theta > \pi/4\}$. This unsafe region corresponds to the safety property that keeps the pendulum within 45° of the vertical axis. The domain for sampling is $\theta \in [-\pi/4, \pi/4] \wedge \omega \in [-1.5, 1.5]$. We used time bound $T = 5$.

$$u = \begin{cases} \dfrac{2 \cdot \omega + \theta + \sin(\theta)}{\cos(\theta)}, & E \in [-1, 1], \mid \omega \mid + \mid \theta \mid \leq 1.85 \\ 0, & E \in [-1, 1], \mid \omega \mid + \mid \theta \mid > 1.85 \\ \dfrac{\omega}{1 + \mid \omega \mid} \cos(\theta), & E < -1 \\ \dfrac{-\omega}{1 + \mid \omega \mid} \cos(\theta), & E > 1 \end{cases} \tag{30}$$

where $E = 0.5 \cdot \omega + (\cos(\theta) - 1)$ is the pendulum energy.

### C.3 Cruise control

The cruise control is a non-deterministic HA with three continuous variables, six modes, eleven transitions, and non-linear polynomial dynamics. It is shown in Fig. 8. The continuous variable $v$ denotes the difference between the vehicle's speed and the cruise speed in $m/s$, $x$ is the integral term for the proportional-integral (PI) controller in mode 5, and $t$ is a clock.

In mode 5, the PI controller tries to stabilize $v$ to zero, i.e., to match the vehicle's speed with the cruise speed. Modes 3 and 4 represent the first level of brakes where deceleration increases smoothly from 1.2 to 2.5 $m/s^2$ in mode 4 and stays constant at 2.5 $m/s^2$ in mode 3. Modes 1 and 2 represent the second level of brakes and work in the same way but with higher starting and peak deceleration. Mode 6 constantly accelerates the vehicle. The guards are designed to prevent chattering or Zeno behavior.

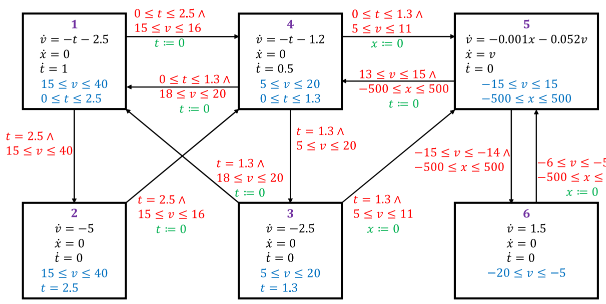

**Fig. 8** Hybrid automaton for the cruise control system. Invariants are in blue, guards are in red, and reset mappings are in green

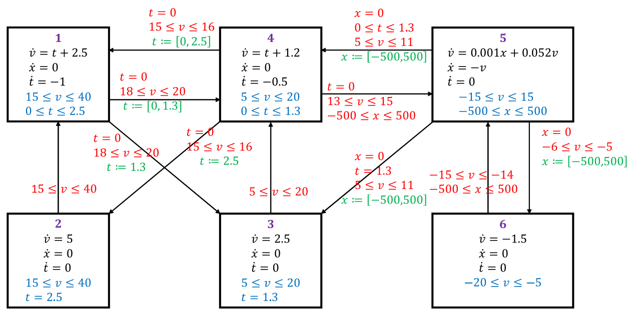

**Fig. 9** Reverse hybrid automaton for the cruise control system. Invariants are in blue, guards are in red, and reset mappings are in green

The unsafe set $D$ is defined by $v \leq -1$, which expresses that the vehicle's speed should not be below a reference speed by 1 m/s or more. The reachability time bound is $T = 10$ (Fig. 9).

## Appendix D Sensitivity analysis

Figure 10.

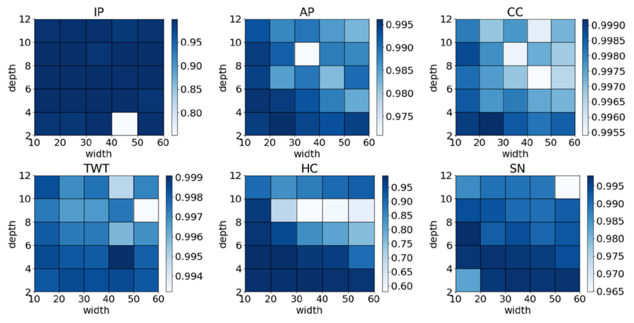

**Fig. 10** Sensitivity analysis: for each model the width and depth of the DNN has been varied. The colormap indicates the accuracy of the predictive monitor

## Appendix E Labeling data

Labeling a state $x$ of an HA $M$ means deciding whether $\mathcal{M} \models \mathsf{Reach}(D, x, T)$, i.e., solving a reachability checking problem. For non-deterministic HAs, we use an SMT solver that supports bounded model checking of hybrid systems. In particular, we choose dReal [12], which provides sound unsatisfiability proofs, but approximates satisfiability up to a user-defined precision ($\delta$-satisfiability). So, we label $x$ as negative (positive) if $\mathcal{M} \models \mathsf{Reach}(D, x, T)$ ($\mathcal{M} \models \neg\mathsf{Reach}(D, x, T)$) is unsatisfiable. If both $\mathsf{Reach}(D, x, T)$ and $\neg\mathsf{Reach}(D, x, T)$ are $\delta$-sat, then the model checker cannot make a decision about $x$, and in this case, we choose to be conservative and mark the state as positive. However, choosing a small $\delta$ makes this situation less likely to happen.

In case of deterministic systems, it is sufficient to simulate the system with an ODE solver and use an event-detection method to check guard conditions and whether the trajectory reaches $D$.

During dataset generation, we sample the HA states to label using either a uniform sampling or a balanced sampling strategy. The former ensures that all states in $X \setminus D$ are equiprobable. The latter produces a balanced amount of positive and negative samples, and is used in cases when the unsafe region $D$ is a small subset of the state space, where a uniform sampling strategy would result in imbalanced datasets with insufficient positive samples. See [61] for more details.

## References

1. Alur, R.: Formal verification of hybrid systems. In Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), pages 273–278, Oct (2011)
2. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998)
3. Chen, X. and Sankaranarayanan, S.: Model predictive real-time monitoring of linear systems. In Real-Time Systems Symposium (RTSS), 2017 IEEE, pages 297–306. IEEE, (2017)
4. Sha, L.: Using simplicity to control complexity. IEEE Softw. **18**(4), 20–28 (2001)
5. Bartocci, E., Deshmukh, J., Donze, A., Fainekos, G., Maler, O., Nickovic, D. and Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In Lectures on Runtime Verification, pages 135–175. Springer, (2018)
6. Phan, D., Paoletti, N., Zhang, T., Grosu, R., Smolka, S.A. and Stoller, S.D.: Neural state classification for hybrid systems. In Automated Technology for Verification and Analysis, volume 11138 of Lecture Notes in Computer Science, pages 422–440, (2018)
7. Vovk, V., Gammerman, A., Shafer G.: Algorithmic learning in a random world. Springer Science & Business Media, Glenn (2005)
8. Neal, Radford M et al.: MCMC using Hamiltonian dynamics. Handbook of markov chain monte carlo, 2(11):2, (2011)
9. Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., Saul, L.K.: An introduction to variational methods for graphical models. Mach. Learn. **37**(2), 183–233 (1999)
10. Bortolussi, L., Cairoli, F., Paoletti, N., Smolka, S.A. and Stoller, S.D.: Neural predictive monitoring. In International Conference on Runtime Verification, pages 129–147. Springer, (2019)
11. Bak, S., Beg, O.A., Bogomolov, S., Johnson, T.T., Nguyen, L.V., Schilling, C.: Hybrid automata: from verification to implementation. Int. J. Softw. Tools Technol. Transf. **21**(1), 87–104 (2019)
12. Gao, S., Kong, S. and Clarke, E.M.: dreal: An smt solver for nonlinear theories over the reals. In International conference on automated deduction, pages 208–214. Springer, (2013)
13. Chen, Xin, Ábrahám, Erika, Sankaranarayanan, Sriram: Flow*: An analyzer for non-linear hybrid systems. In International Conference on Computer Aided Verification, pages 258–263. Springer, (2013)
14. Althoff M.: An introduction to CORA 2015. In Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems, (2015)
15. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K. and Schilling, C,: Juliareach: a toolbox for set-based reachability. In Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pages 39–44, (2019)
16. Hornik, K., Stinchcombe, M., White, H., et al.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
17. Rumelhart, D.E., Hinton, G.E. and Williams, R.J.: Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, (1985)
18. Papadopoulos, H.: Inductive conformal prediction: Theory and application to neural networks. In Tools in artificial intelligence, InTech (2008)
19. Vineeth B., Shen-Shyang H., Vladimir V.: Conformal prediction for reliable machine learning: theory, adaptations and applications. Newnes (2014)
20. Christopher M Bishop. Pattern recognition and machine learning. Springer, 2006
21. Gal, Y.: Uncertainty in deep learning. PhD thesis, University of Cambridge, (2016)
22. MacKay, D.J.C.: A practical bayesian framework for backpropagation networks. Neural Comput. **4**(3), 448–472 (1992)
23. Van der Vaart, A.W.: Asymptotic statistics. Cambridge University Press, Cambridge (2000)
24. Rasch, D., Pilz, J., Verdooren L.R., and Gebhardt A.: Chapman and Hall/CRC, Optimal experimental design with R (2011)
25. Massart, P.: The tight constant in the dvoretzky-kiefer-wolfowitz inequality. The annals of Probability, pages 1269–1283, (1990)
26. Deodato, G., Ball, C. and Zhang, X.: Bayesian neural networks for cellular image classification and uncertainty analysis. bioRxiv, page 824862, 2019
27. Jordaney, R., Sharad, K., Dash, S.K., Wang, Z., Papini, D., Nouretdinov, I. and Cavallaro, L.: Transcend: Detecting concept drift in malware classification models. In 26th USENIX Security Symposium (USENIX Security 17), pages 625–642, (2017)
28. Guo, C., Pleiss, G., Sun, Y. and Weinberger, K.Q: On calibration of modern neural networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1321–1330. JMLR. org, (2017)
29. Bortolussi, L., Cairoli, F., Paoletti, N. and Stoller, S.D.: Conformal predictions for hybrid system state classification. In From Reactive Systems to Cyber-Physical Systems, pages 225–241. Springer, (2019)
30. Brefeld, U., Geibel, P. and Wysotzki, F.: Support vector machines with example dependent costs. In European Conference on Machine Learning, pages 23–34. Springer, (2003)

31. Batuwita, R., Palade, V.: Class imbalance learning methods for support vector machines, chapter 5, pages 83–99. John Wiley & Sons, Ltd, (2013)
32. Rasmussen, C.E., Williams, C.K.I: Gaussian processes for machine learning, volume 1. MIT press Cambridge, (2006)
33. Boughorbel, S., Jarray, F., El-Anbari, M.: Optimal classifier for imbalanced data using matthews correlation coefficient metric. PloS one **12**(6), e0177678 (2017)
34. Paoletti, N., Liu, K.S., Smolka, S.A. and Lin, S.: Data-driven robust control for type 1 diabetes under meal and exercise uncertainties. In International Conference on Computational Methods in Systems Biology, pages 214–232. Springer, (2017)
35. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., et al.: Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, (2016)
36. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A. et al.: Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, pages 8024–8035, (2019)
37. Chollet, F. et al.: Keras: The Python deep learning library. Astrophysics Source Code Library, (2018)
38. Tran, D., Kucukelbir, A., Dieng, A.B., Rudolph, M., Liang, D. and Blei, D.M.: Edward: A library for probabilistic modeling, inference, and criticism. arXiv preprint arXiv:1610.09787, (2016)
39. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., Goodman, N.D.: Pyro: deep universal probabilistic programming. J. Mach. Learn. Res. **20**(1), 973–978 (2019)
40. Yoon, H., Chou, Y., Chen, X., Frew, E. and Sankaranarayanan, S.: Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for uavs. In International Conference on Runtime Verification, pages 349–367. Springer, (2019)
41. Stanley B., Taylor J.T., Marco C., Lui S.: Real-time reachability for verified simplex design. In Real-Time Systems Symposium (RTSS), 2014 IEEE, pages 138–148. IEEE, (2014)
42. Sauter, G., Dierks, H., Franzle, M. and Hansen, M.R.: Lightweight hybrid model checking facilitating online prediction of temporal properties. In Proceedings of the 21st Nordic Workshop on Programming Theory, pages 20–22, 2009
43. Djeridane, B. and Lygeros, J.: Neural approximation of PDE solutions: An application to reachability computations. In Proceedings of the 45th IEEE Conference on Decision and Control, pages 3034–3039. IEEE, (2006)
44. Royo, V.R., Fridovich-Keil, D., Herbert, S. and Tomlin, C.J.: Classification-based approximate reachability with guarantees applied to safe trajectory tracking. arXiv preprint arXiv:1803.03237, (2018)
45. Yel, E., Carpenter, T.J., Di Franco, C., Ivanov, R., Kantaros, Y., Lee, I., Weimer, J., Bezzo, N.: Assured runtime monitoring and planning: toward verification of neural networks for safe autonomous operations. IEEE Robotics Automation Magazine **27**(2), 102–116 (2020)
46. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J. and Lee, I.: 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, pages 169–178, (2019)
47. Babaee R., Gurfinkel A., Fischmeister S.: Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning. In International Conference on Runtime Verification, pages 187–204. Springer, (2018)
48. Babaee R., Ganesh V., Sedwards S.: Accelerated learning of predictive runtime monitors for rare failure. In International Conference on Runtime Verification, pages 111–128. Springer, (2019)
49. Qin, X. and Deshmukh, J.V.: Predictive monitoring for signal temporal logic with probabilistic guarantees. In Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pages 266–267. ACM, (2019)
50. Dokhanchi, A., Hoxha, B. and Fainekos, G.: On-line monitoring for temporal logic robustness. In International Conference on Runtime Verification, pages 231–246. Springer, (2014)
51. Deshmukh J.V., Donze A., Ghosh S., Jin X., Juniwal G., Seshia S.A. (2017) Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**(1), 5–30
52. Bortolussi, L., Milios, D., Sanguinetti, G.: Smoothed model checking for uncertain continuous-time Markov chains. Inf. Comput. **247**, 235–253 (2016)
53. Pop, R. and Fulop, P.: Deep ensemble bayesian active learning: Addressing the mode collapse issue in monte carlo dropout via ensembles. arXiv preprint arXiv:1811.03897, (2018)
54. Gal, Y., Islam, R. and Ghahramani, Z.: Deep bayesian active learning with image data. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1183–1192. JMLR. org, (2017)
55. Dayoub, F., Sunderhauf, N. and Corke, P.I.: Episode-based active learning with bayesian neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 26–28, (2017)
56. Makili, Lázaro Emílio., Vega Sánchez, Jesús A., Dormido-Canto, Sebastián: Active learning using conformal predictors: application to image classification. Fusion Science and Technology **62**(2), 347–355 (2012)
57. Toccaceli, P., Gammerman, A.: Combination of inductive mondrian conformal predictors. Mach. Learn. **108**(3), 489–510 (2019)
58. Gammerman, A., Vovk, V.: Hedging predictions in machine learning. Comput. J. **50**(2), 151–163 (2007)
59. Shafer, G., Vovk, V.: A tutorial on conformal prediction. J. Mach. Learn. Res. **9**, 371–421 (2008)
60. Balasubramanian, V.N., Gouripeddi, R., Panchanathan, S., Vermillion, J., Bhaskaran, A., & Siegel, R. M.: Support vector machine based conformal predictors for risk of complications following a coronary drug eluting stent procedure. In 2009 36th Annual Computers in Cardiology Conference (CinC), pages 5–8. IEEE, (2009)
61. Phan, D., Paoletti, N., Zhang, T., Grosu, R., Smolka, S. A., Stoller, S. D.: Neural state classification for hybrid systems. ArXiv e-prints, July (2018)