NeuReach: Learning Reachability Functions from Simulations*

Dawei Sun(✉) and Sayan Mitra

University of Illinois at Urbana-Champaign, Urbana IL 61801, USA
{daweis2,mitras}@illinois.edu

Abstract. We present NeuReach, a tool that uses neural networks for predicting reachable sets from executions of a dynamical system. Unlike existing reachability tools, NeuReach computes a *reachability function* that outputs an accurate over-approximation of the reachable set for *any* initial set in a parameterized family. Such reachability functions are useful for online monitoring, verification, and safe planning. NeuReach implements empirical risk minimization for learning reachability functions. We discuss the design rationale behind the optimization problem and establish that the computed output is probably approximately correct. Our experimental evaluations over a variety of systems show promise. NeuReach can learn accurate reachability functions for complex nonlinear systems, including some that are beyond existing methods. From a learned reachability function, arbitrary reachtubes can be computed in milliseconds. NeuReach is available at https://github.com/sundw2014/NeuReach.

Keywords: Reachability analysis · Data-driven methods · Machine learning

1 Introduction

Reachability has traditionally been a fundamental building block for verification, monitoring, and prediction, and it is finding ever-expanding set of applications in control of cyber-physical and autonomous systems [19,23]. Reachtubes cannot be computed exactly for general hybrid models, but remarkable progress over the past two decades have led to approximation algorithms for nonlinear and very high-dimensional linear models (See, for example, [11,18,5,3,25,12,1,26,34]). All of these algorithms and tools compute the reachtube from scratch, every time the algorithm is invoked for a new initial set $\mathcal{X}_0$, even if the system model does not change. This is a missed opportunity in amortizing the cost of reachability over multiple invocations. All the applications mentioned above, like verification, monitoring, and prediction, indeed use multiple reachtubes of the same system, but from different initial sets.

* The authors were supported by research grants from the National Security Agency's Science of Security (SoS) program and National Science Foundation's Formal Methods in the Field (FMITF) program.

D. Fisman and G. Rosu (Eds.): TACAS 2022, LNCS 13243, pp. 322–337, 2022.
https://doi.org/10.1007/978-3-030-99524-9_17

In this paper, we present NeuReach, a tool that learns a *reachability function* from executions of dynamical systems. With the learned reachability function, for every new initial set a corresponding reachtube can be computed quickly. To use NeuReach, the user has to implement a simulator function of the underlying dynamical (or hybrid) system for generating trajectories, and several other functions for sampling initial sets. As output, the tool will generate a function which can be serialized and stored for repeated use. This function takes as input a query which is an initial set $\mathcal{X}_0$ and a time instant t, and outputs an ellipsoid, which is guaranteed to be an accurate over-approximation of the actual reachable set.

Formally, NeuReach solves a probabilistic variant of the well-studied reachability problem: the problem is to compute a *reachability function* $R(\cdot, \cdot)$ for a given model (or simulator), such that for *any* initial set $\mathcal{X}_0$ and time t, the output of the function $R(\mathcal{X}_0, t)$ is an over-approximation of the actual reachset from $\mathcal{X}_0$ at time t. That is, R is computed once and for all—possibly with an expensive algorithm—and thereafter, for every new initial set $\mathcal{X}_0$ and time t, the reachset over-approximation $R(\mathcal{X}_0, t)$ is computed simply by calling R. Thus, it enables online and even real-time applications of reachset approximations.

NeuReach computes reachability functions using machine learning. We view this as a statistical learning problem where samples of the system's trajectories have to be used to learn a parameterized reachability function $R_\theta(\cdot, \cdot)$. Because the trajectory samples are the only requirements from the underlying dynamical system to run NeuReach, it can be applied to systems with or without analytical models. In this paper, we discuss how the above problem can be cast as an optimization problem. This involves carefully designing a loss function that penalizes error and conservatism of the reachability function. With this loss function, it becomes possible to solve the problem using empirical risk minimization and stochastic gradient descent. For the sake of justifying our design, we derive a theoretical guarantee on the sample complexity using standard statistical learning theory tools.

We evaluate NeuReach on several benchmark systems and compare it with DryVR [21] which also uses machine learning for single-shot reachset computations. Results show that, with the same training data, NeuReach generates more accurate and tighter reachsets. Using NeuReach we are able to check the key safety properties of the challenging F-16 benchmark presented in [28]. To our knowledge, this is the first successful verification of at least some scenarios in this benchmark. Furthermore, as expected, once $R(\cdot, \cdot)$ is computed, it can be invoked to rapidly compute reachsets for arbitrary $\mathcal{X}_0$ and t. For example, estimating a reachset for an 8-dimensional dynamical system with an NN-controller only takes ~ 0.3 milliseconds. This makes NeuReach attractive for online and real-time applications.

Contributions. (1) We present a simple but effective and useful machine-learning algorithm for learning reachability functions from simulations. With the learned reachability function, accurate over-approximation of the reachable set for *any* initial set in a parameterized family can be quickly computed, which

enables real-time safety check and online planning; (2) We derive a probably approximately correct (PAC) bound on the error of the learned reachability function (Theorem 1) using techniques in statistical learning theory; (3) We evaluate the proposed tool on several benchmark dynamical systems and compare it with another data-driven reachability tool. Experiments show that NeuReach can learn more accurate and tighter reachability functions for complex nonlinear and hybrid systems, including some that are beyond existing methods.

2 Related work

Reachability analysis for models with known dynamics. This category of approaches consider the reachability analysis of models with known dynamics (i.e., white-box models). This is an active research area, and there is an extensive body of theory and tools on this topic [11,2,15,5,25,33,27,16,38,10,39]. Reachability analysis is hard in general. Exact reachability is undecidable even for deterministic linear and rectangular models [29,24]. For dynamical models described with ordinary differential equations (ODE), Hamilton–Jacobi–Bellman (HJB) equations can be used to derive the exact reachable sets [30,6,7]. An HJB equation is a partial differential equation (PDE). Solutions of this PDE defines the reachabiltiy of the underlying dynamical system. However, solving HJB equations is difficult, and such approaches do not scale to high-dimensional systems. In practice, the exact reachable set might be unnecessary. For example, over-approximations of the reachable sets could suffice for safety check purpose. To this end, many approaches and tools have been developed. For example, Flow* [11] uses the technique of Taylor model integration to compute over-approximations of the solution of an ODE.

Another series of work [22,20] leverage the sensitivity analysis of ODE to bound the discrepancy of solutions starting from a small initial set, and thus can compute an over-approximation of the exact reachable set. In [12], a Lagrangian-based algorithm is proposed, which makes use of the Cauchy-Green stretching factor derived from an over-approximation of the gradient of the solution-flows of an ODE. All of the above approaches consider set-based reachability analysis.

Data-driven reachability analysis. In the cases where the exact dynamics of the systems is unknown or partially known, the above approaches cannot be applied. One straight-forward direction is to learn the reachability from behaviors [42] of the dynamical system. Several approaches have been proposed for reachability *only* using simulations of the underlying system. These approaches include scenario optimization [14,44], sensitivity analysis [21], Gaussian processes [13], adversarial sampling [32,9], etc.

NeuReach falls in the category of approaches that use randomized algorithms for reachability analysis of deterministic (and not stochastic) systems. Another member in this category is the scenario optimization approach presented in [14]. Different from NeuReach, this method learns a single reachset for a fixed initial set and time interval instead of a mapping from arbitrary initial sets and time

to the reachsets. Another approach based on scenario optimization is presented in [44]. This method computes a fixed-width reachtube by learning a function of time to represent the central axis of the reachtube. Moreover, it uses polynomials with handcrafted feature vectors for learning, which requires case-by-case design and fine-tuning. In contrast, our method learns a more flexible reachability function using neural networks and avoids the use of handcrafted feature vectors. DryVR [21] computes the reachtubes based on sensitivity analysis. It first learns a sensitivity function with theoretical guarantees, and then uses it to compute the reachset. Among all these tools or methods, we found that DryVR is the only one that has a publicly available implementation. Thus, we compared NeuReach with DryVR.

Neural networks for reachability analysis. Applications of machine learning with neural networks for reachability and monitoring has become an active research area. The approach in [23] aims to learn the reachtube from data using neural networks, with a focus in motion planning. Unlike NeuReach, this approach learns the dynamics of the reachtube, and the reachtube can be obtained by integrating that dynamics. In [30,7], neural networks are used as a PDE solver to approximate the solution of HJB equations. The approach in [36] makes use of neural networks to approximate the reachability of dynamical systems with control input. In [38,10], the authors develop a framework for runtime predictive monitoring of hybrid automata using neural networks and conformal prediction.

3 Problem setup and an overview of the tool

NeuReach works with deterministic dynamical systems. The state of the system is denoted by $x \in \mathcal{X} \subseteq \mathbb{R}^n$. We assume that we have access to a simulator function $\xi : \mathcal{X} \times \mathbb{R}_{\geq 0} \mapsto \mathcal{X}$ that generates trajectories of the system up to a time bound T. That is, given an initial state $x_0 \in \mathcal{X}$ and a time instant $t \in [0, T]$, $\xi(x_0, t)$ is the state at time t.[1]

Consider the evolution of the system from a set of initial states (*initial set*) $\mathcal{X}_0 \subset \mathcal{X}$. Lifting the notation of ξ to sets, we write the *reachset* from $\mathcal{X}_0$ as $\xi(\mathcal{X}_0, t) := \cup_{x_0 \in \mathcal{X}_0} \xi(x_0, t)$. In general, $\xi(\mathcal{X}_0, t)$ cannot be computed precisely, and thus, we resort to over-approximations of $\xi(\mathcal{X}_0, t)$ which are usually sufficient for verification and monitoring of safety and general temporal logic requirements, and also for planning. Beyond computing over-approximations of $\xi(\mathcal{X}_0, t)$ for a single $\mathcal{X}_0$ and t, we are interested in finding a *reachability function* $R : 2^{\mathcal{X}} \times [0, T] \mapsto 2^{\mathcal{X}}$ such that, ideally, $\xi(\mathcal{X}_0, t) \subseteq R(\mathcal{X}_0, t)$ for all valid $\mathcal{X}_0$ and t. NeuReach implements a solution to this problem which provides a probabilistic version of the above guarantee with some restrictions on the shape of the initial set $\mathcal{X}_0$.

In order to discuss the error of a reachability function R, we have to assume that its arguments $\mathcal{X}_0$ and t are independently chosen according to some dis-

[1] For the sake of simplicity, here we ignore issues arising from quantization and numerical errors in simulators. Such issues have been extensively studied in the numerical analysis and we refer the reader to [17] for a discussion related to verification.

tributions P_1 and P_2, i.e. $\mathcal{X}_0 \sim P_1$ and $t \sim P_2$. Also, we need a distribution function $\mathcal{D}(\cdot)$ such that $\mathcal{D}(\mathcal{X}_0)$ is distribution over $\mathcal{X}_0$. For example, $\mathcal{D}(\mathcal{X}_0)$ could be the uniform distribution over $\mathcal{X}_0$. Given these distributions, the *error* of a reachability function is defined as:

$$\Pr_{\mathcal{X}_0 \sim P_1, t \sim P_2, x_0 \sim \mathcal{D}(\mathcal{X}_0)} \left[\xi(x_0, t) \notin R(\mathcal{X}_0, t)\right]. \tag{1}$$

Here, we assume that the joint distribution of $(\mathcal{X}_0, t, x_0)$ is defined on the Borel $\sigma-$algebra such that any Borel set is measurable. Given the fact that R is continuous[2] and ξ as the trajectory of a dynamical system is at least piece-wise continuous, the set of all tuples $(\mathcal{X}_0, t, x_0)$ that satisfy $\xi(x_0, t) \notin R(\mathcal{X}_0, t)$ must be a Borel set, and thus is measurable. Therefore, the above probability is well defined.

User interface and data representation. P_1, P_2 and $\mathcal{D}$ are specified by the user as functions generating samples (explained below). The input and output of the reachability function $R(\mathcal{X}_0, t)$ involve infinite objects, and in order to learn R, first, we need some finite representations of these objects. In NeuReach, $\mathcal{X}_0$ is picked from a user-specified family of sets where each set can be represented by a finite number of parameters. For example, $\mathcal{X}_0$ could be a ball and represented by two parameters — center and radius. From here on, we will not distinguish between $\mathcal{X}_0$ and its parameterized representation. Similarly, the reachset $R(\mathcal{X}_0, t)$ also needs a representation. NeuReach represents the reachsets with ellipsoids. Given a vector $x_0 \in \mathbb{R}^n$ and a matrix $C \in \mathbb{R}^{n \times n}$, the set $\mathcal{E}(x_0, C) := \{x \in \mathbb{R}^n : \|C \cdot (x - x_0)\|_2 \leq 1\}$ is an *ellipsoid*. Thus, given the center, an ellipsoid can be represented by an $n \times n$ matrix.

In order to use NeuReach, the user has to implement the following functions.

(i) `sample_X0()`: Produces a random initial set $\mathcal{X}_0$ from a distribution P_1. Specifically, the parameterized representation of $\mathcal{X}_0$ is returned.
(ii) `sample_t()`: Produces a random sample of t from a distribution P_2.
(iii) `sample_x0(X0)`: Takes an initial set $\mathcal{X}_0$, and produces a random sample of $x_0 \in \mathcal{X}_0$ according to a distribution $\mathcal{D}(\mathcal{X}_0)$.
(iv) `simulate(x0)`: Takes an initial state x_0 and generates a finite trajectory $\xi(x_0, \cdot)$ which is a sequence of states at some time instants. The user should make sure that for every time instant returned by `sample_t()`, a state corresponding to it can be found in the simulated trajectory.
(v) `get_init_center(X0)`: Takes an initial set $\mathcal{X}_0$ and returns $\mathbb{E}\left[\mathcal{D}(\mathcal{X}_0)\right] := \mathbb{E}_{x \sim \mathcal{D}(\mathcal{X}_0)}[x]$, which is the mean value of x over the initial states.

Given these functions, NeuReach computes a reachability function R with an error guarantee (Theorem 1). The reachset $R(\mathcal{X}_0, t)$ is an ellipsoid centered at $\xi(\mathbb{E}\left[\mathcal{D}(\mathcal{X}_0)\right], t)$. As the output, NeuReach will generate a Python function `R(X0, t)`. This function can be serialized and stored on disk for future use. When calling this function, the user provides the initial set $\mathcal{X}_0$ and t, and then an $n \times n$ matrix representing the shape of the ellipsoid will be returned.

[2] As will be stated later, R is a neural network, which is indeed continuous.

4 Design of NeuReach: Learning reachability functions

We present the design rationale behind NeuReach and discuss the learning algorithm it implements. We show that standard results in statistical learning theory give a probabilistic guarantee on the error of the learned reachability function.

4.1 Reachability with Empirical Risk Minimization

The basic idea is to model the reachset $R(\mathcal{X}_0, t)$ as an ellipsoid around $\xi(\mathbb{E}[\mathcal{D}(\mathcal{X}_0)], t)$. As stated earlier, given the center, an n-dimensional ellipsoid can be represented by an $n \times n$ matrix. Thus, learning the set-valued reachability function $R(\mathcal{X}_0, t)$ becomes the problem of learning a matrix-valued function $C(\mathcal{X}_0, t)$ that describes the shape of the set. We represent function C using parametric models, such as neural networks. Let us denote this parametric, matrix-valued function by C_θ, where $\theta \in \mathcal{W} \subseteq \mathbb{R}^p$ is the vector of parameters. The parameter θ could be, for example, a scalar representing a coefficient of a polynomial, a vector representing weights of a neural network, etc. Thus, the parametric reachability function is:

$$R_\theta(\mathcal{X}_0, t) := \mathcal{E}(\xi(\mathbb{E}[\mathcal{D}(\mathcal{X}_0)], t), C_\theta(\mathcal{X}_0, t)). \tag{2}$$

To simplify the notations, for $X = (\mathcal{X}_0, t, x_0)$ and parameter θ, we define a function $g_\theta(X) := \|C_\theta(\mathcal{X}_0, t)(\xi(x_0, t) - \xi(\mathbb{E}[\mathcal{D}(\mathcal{X}_0)], t))\|_2$. For a particular sample X and a parameter θ, if $g_\theta(X) \leq 1$, then $\xi(x_0, t) \in R_\theta(\mathcal{X}_0, t)$, otherwise it is outside and contributes to the error. The goal of our learning algorithm is to find a θ to minimize the error of the resulting reachability function R_θ, which gives the following optimization problem:

$$\begin{aligned}
\theta^* &= \arg\min_\theta \Pr_{\mathcal{X}_0 \sim P_1, t \sim P_2, x_0 \sim \mathcal{D}(\mathcal{X}_0)} [\xi(x_0, t) \notin R_\theta(\mathcal{X}_0, t)] \\
&= \arg\min_\theta \mathop{\mathbb{E}}_{\mathcal{X}_0, t, x_0} \left[\mathbb{I}\left(\left\| C_\theta(\mathcal{X}_0, t) \cdot \left(\xi(x_0, t) - \xi(\mathbb{E}[\mathcal{D}(\mathcal{X}_0)], t) \right) \right\|_2 > 1 \right) \right] \\
&= \arg\min_\theta \mathop{\mathbb{E}}_{X := (\mathcal{X}_0, t, x_0)} [\mathbb{I}(g_\theta(X) - 1 > 0)],
\end{aligned}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

In order to solve the above optimization problem using empirical risk minimization, we consider the following setup. First, a training set is constructed. We denote a training set with N samples by $S = \{X_i\}_{i=1}^N$, where the samples $X_i = (\mathcal{X}_0^{(i)}, t^{(i)}, x_0^{(i)})$ are independently drawn from the data distribution defined by $\mathcal{X}_0 \sim P_1, t \sim P_2, x_0 \sim \mathcal{D}(\mathcal{X}_0)$. The *empirical loss* on S for a parameter θ is

$$L_{ERM}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(g_\theta(X_i) - 1), \tag{3}$$

where $\ell(x) := \max\{0, \frac{x}{\alpha} + 1\}$ is the hinge loss function with the hyper-parameter $\alpha > 0$, which is a soft proxy for the indicator function. Therefore, the empirical loss L_{ERM} is a soft, empirical proxy of the actual error as defined in Equation (1).

Arguments	Default Value	Description
`system`	-	Name of the Python file containing the model.
`lambda`	0.03	λ in Eq. (4).
`alpha`	0.001	α in Eq. (3).
`N_X0`	100	$N_{\mathcal{X}_0}$: Number of initial sets.
`N_x0`	10	N_{x_0}: Number of initial states.
`N_t`	100	N_t: Number of time instants.
`layer1`	64	L_1: Number of neurons in the first layer of the NN.
`layer2`	64	L_2: Number of neurons in the second layer of the NN.
`epochs`	30	Number of epochs for training.
`lr`	0.01	Learning rate.

Table 1: Command-line arguments passed to the tool.

In addition to minimizing the empirical loss, we would also like the over-approximation of the reachset to be as tight as possible. Thus, the volume of the ellipsoid should be penalized. Inspired by [14], we use $-\log(\det(C^\intercal C))$ as a proxy of the volume of an ellipsoid $\mathcal{E}(x_0, C)$, and the following regularization term is added to penalize large ellipsoids.

$$L_{REG}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\det\left(C_\theta(\mathcal{X}_0^{(i)}, t^{(i)})^\intercal C_\theta(\mathcal{X}_0^{(i)}, t^{(i)})\right)\right).$$

Combining the two terms, we define the overall optimization problem:

$$\hat{\theta} = \arg\min_\theta L_{ERM}(\theta) + \lambda L_{REG}(\theta), \tag{4}$$

where λ is a hyper-parameter balancing two loss terms.

Machine learning setup. The training set is constructed as follows. First, we sample $N_{\mathcal{X}_0}$ initial sets by calling `sample_X0()`. Then, for each initial set, we sample N_{x_0} initial states from it using `sample_x0(X0)` and then get N_{x_0} trajectories by calling `simulate(x0)`. Finally, for each trajectory, we sample N_t time instants by calling `sample_t()`. Thus, the resulting training set contains $N := N_{\mathcal{X}_0} \times N_{x_0} \times N_t$ samples, but generating such a training set only needs $N_{\mathcal{X}_0} \times N_{x_0}$ trajectory simulations. NeuReach implements the optimization problem of Equation (4) in Pytorch [37] and solves it with stochastic gradient descent. By default, a three-layer neural network is used to represent C_θ. For n-dimensional reachsets, the number of neurons in each layer are L_1, L_2, and n^2, where L_1 and L_2 can be specified by the user. The output vector of the neural network is then reshaped to be an $n \times n$ matrix. By default, we set $\alpha = 0.001$ and $\lambda = 0.03$. The neural network is trained for 30 epochs with a learning rate of 0.01. Hyper-parameters including learning rate, α, λ, and size of the training set can be easily changed via the user interface as shown in Table 1.

4.2 Probabilistic Correctness of NeuReach

The following theorem shows that the error of the learned reachability function $R_{\hat{\theta}}$ can be bounded. Specifically, the difference between the error and the empirical loss is $O(\sqrt{\frac{1}{N}})$, where N is the size of the training set.

Theorem 1. *For any $\epsilon > 0$, and a random training set S with N i.i.d. samples, with probability at least $1 - 2\exp(-2N\epsilon^2)$, the following inequality holds,*

$$\mathbb{E}_X\left[\mathbb{I}\left(g_{\hat{\theta}}(X) - 1 > 0\right)\right] \leq \frac{1}{N}\sum_{i-1}^{N}\tilde{\ell}(g_{\hat{\theta}}(X_i) - 1) + \frac{12}{\alpha}L_g\sqrt{\frac{p}{N}} + \epsilon, \tag{5}$$

where p is the number of parameters, i.e. $\theta \in \mathbb{R}^p$, and $\tilde{\ell}(\cdot) = \min\{1, \ell(\cdot)\}$ is the truncated hinge loss, and L_g is the Lipschitz constant of g_θ w.r.t. θ.

Theorem 1 shows that by controlling ϵ and N, the actual error $\mathbb{E}_X\left[\mathbb{I}\left(g_{\hat{\theta}}(X) - 1 > 0\right)\right]$ can be made arbitrarily close to the empirical loss $\frac{1}{N}\sum_{i-1}^{N}\tilde{\ell}(g_{\hat{\theta}}(X_i) - 1)$, with arbitrarily high probability. The *empirical loss* on the training set S can be made very small in practice due to the high capacity of the neural network. Of course, there is no free lunch, in general. In order to drive the empirical loss to 0, we might have to increase the number of parameters, which in turn increases the term $\frac{12}{\alpha}L_g\sqrt{\frac{p}{N}}$. Furthermore, the hyper-parameter λ also affects the empirical loss. A smaller λ results in lower empirical loss but more conservative reachsets. Actually, conservatism and accuracy are conflicting requirements. As shown in [21], when using reachability to verify safety, accuracy determines the soundness of the verification, while conservatism influences the sample efficiency. We wanted to focus more on soundness than on efficiency. Thus, a theoretical guarantee is derived for accuracy but not for conservatism.

Proof. Starting from the left hand side and using the definition of hinge loss, we get $\mathbb{E}_X\left[\mathbb{I}\left(g_{\hat{\theta}}(X) - 1 > 0\right)\right] \leq \mathbb{E}_X\left[\tilde{\ell}(g_{\hat{\theta}}(X) - 1)\right]$. By adding and subtracting the empirical loss term, we get:

$$\begin{aligned}
&\mathbb{E}_X\left[\tilde{\ell}(g_{\hat{\theta}}(X) - 1)\right] - \frac{1}{N}\sum_{i=1}^{N}\tilde{\ell}(g_{\hat{\theta}}(X_i) - 1) + \frac{1}{N}\sum_{i=1}^{N}\tilde{\ell}(g_{\hat{\theta}}(X_i) - 1) \\
&\leq \sup_{\theta\in\mathcal{W}}\left(\mathbb{E}_X\left[\tilde{\ell}(g_\theta(X) - 1)\right] - \frac{1}{N}\sum_{i=1}^{N}\tilde{\ell}(g_\theta(X_i) - 1)\right) + \frac{1}{N}\sum_{i=1}^{N}\tilde{\ell}(g_{\hat{\theta}}(X_i) - 1),
\end{aligned}$$

where the inequality follows from the definition of supremum.
Let $\mathcal{V} = \sup_{\theta\in\mathcal{W}}\left(\mathbb{E}_X\left[\tilde{\ell}(g_\theta(X) - 1)\right] - \frac{1}{N}\sum_{i=1}^{N}\tilde{\ell}(g_\theta(X_i) - 1)\right)$, i.e. the worst-case difference between the empirical average and the expectation of the loss. Note that $\mathcal{V}$ is a random quantity since $S = \{X_i\}_{i=1}^{N}$ is random. Next, we derive an upper bound on $\mathcal{V}$ that holds with high probability.

First, we derive an upper bound on $\mathbb{E}_S[\mathcal{V}]$. Let $\mathcal{G}$ be the function class containing g_θ parameterized by θ, i.e. $\mathcal{G} := \{g_\theta(\cdot) \mid \theta \in \mathcal{W}\}$. Similarly, $\mathcal{F} := \{\tilde{\ell}(g_\theta(\cdot) -$

$1) \mid \theta \in \mathcal{W}\}$. Applying $\mathcal{G}$ to the set of inputs S generates a new set $\mathcal{G}(S) := \{(g(X_1), g(X_2), \cdots, g(X_N)) : g \in \mathcal{G}\}$. Define $\mathcal{F}(S) := \{(f(X_1), \cdots, f(X_N)) : f \in \mathcal{F}\}$ in the same way.

Notice that $\mathcal{V}$ is the worst-case (among all $f_\theta \in \mathcal{F}$) gap between the expectation and the empirical average of $f_\theta(X)$. A fundamental result in PAC learning (Theorem 3.3 in [35]) shows that this gap can be bounded as $\mathbb{E}_S[\mathcal{V}] \leq 2\mathbb{E}_S[\mathtt{Rad}(\mathcal{F}(S))]$, where $\mathtt{Rad}(\mathcal{F}(S))$ is the Rademacher complexity [35] of $\mathcal{F}(S)$. Furthermore, notice that $\mathcal{F}(S)$ can be generated from $\mathcal{G}(S)$ by shifting it and composing it with $\tilde{\ell}$. It follows from Talagrand's contraction lemma [31] that $\mathbb{E}_S[\mathtt{Rad}(\mathcal{F}(S))] \leq 2L_{\tilde{\ell}}\mathbb{E}_S[\mathtt{Rad}(\mathcal{G}(S))]$, where $L_{\tilde{\ell}} = \frac{1}{\alpha}$ is the Lipschitz constant.

Finally, following from a conclusion on Rademacher complexity of Lipschitz parameterized function classes (See page 13 in [8]), we get $\mathbb{E}_S[\mathtt{Rad}(\mathcal{G}(S))] \leq 3L_g\sqrt{\frac{p}{N}}$. Therefore, we get

$$\mathbb{E}_S[\mathcal{V}] \leq \frac{12}{\alpha} L_g \sqrt{\frac{p}{N}}. \tag{6}$$

Then, applying McDiarmid's inequality [35] gives a high-probability bound on $\mathcal{V}$. That is,

$$\Pr_S\left(\left|\mathcal{V} - \mathbb{E}[\mathcal{V}]\right| \geq \epsilon\right) \leq 2\exp(-2N\epsilon^2).$$

Together with Eq. (6), we have $\mathcal{V} \leq \mathbb{E}[\mathcal{V}] + \epsilon \leq \frac{12}{\alpha} L_g \sqrt{\frac{p}{N}} + \epsilon$ with probability at least $1 - 2\exp(-2N\epsilon^2)$. This implies

$$\mathbb{E}_X\left[\mathbb{I}\left(g_{\hat{\theta}}(X) - 1 > 0\right)\right] \leq \frac{1}{N}\sum_{i-1}^{N} \tilde{\ell}(g_{\hat{\theta}}(X_i) - 1) + \frac{12}{\alpha} L_g \sqrt{\frac{p}{N}} + \epsilon,$$

with probability at least $1 - 2\exp(-2N\epsilon^2)$, which completes the proof. □

5 Experimental evaluation

We evaluated NeuReach on several benchmark systems including the Van der Pol oscillator, the Moore-Greitzer model of a jet engine, an 8-dimensional quadrotor controlled by a neural network [40], and an F-16 Ground Collision Avoidance system [28]. We also compare our method with DryVR [21]. Since NeuReach is fully data-driven and does not rely on the analytical model of the system, it would not make sense to compare against model-based methods like Hamilton-Jacobi reachability analysis [6], Flow* [11], C2E2 [18], or SReach [41]. Some of our benchmarks cannot be handled by these tools. Also, once the reachability function is learned, many reachsets can be computed very quickly by our method. Given that other tools need to compute the reachset from scratch for each new query, comparisons based on running times, would not make sense either.

5.1 Benchmark systems

The simulators available for the benchmark systems allow us to specify fixed time-steps Δt and a time bound T. As for the distribution P_2, we adopt the uniform distribution, i.e. $P_2 = \mathsf{Unif}(\{\Delta t, 2\Delta t, \cdots, \lfloor \frac{T}{\Delta t} \rfloor \Delta t\})$ (Recall, the definition of this distribution in Section 3). For a given initial set $\mathcal{X}_0$, $\mathcal{D}(\mathcal{X}_0)$ is defined as the uniform distribution on the boundary of $\mathcal{X}_0$. As shown in Corollary 1 of [43], the boundary of the reachable set of an initial set is equal to the reachable set of the initial set's boundary for ODEs. That is, if the estimated reachable set contains the reachable set of the initial set's boundary, it automatically contains that of the interior. Thus, we only sample points on the boundary of $\mathcal{X}_0$ to improve sample efficiency. As for the distribution P_1, we will give details for each benchmark below.

Van der Pol oscillator is a widely used 2-dimensional nonlinear model. An initial set $\mathcal{X}_0$ is a ball centered at c with radius r. The distribution P_1 for choosing $\mathcal{X}_0$ is specified by the distributions for choosing these parameters. In our experiments, we use $c \sim \mathsf{Unif}([1,2] \times [2,3])$ and $r \sim \mathsf{Unif}([0, 0.5])$. The time bound is set to $T = 4$, and time step is $\Delta t = 0.05$.

JetEngine model from [4] is also 2-dimensional and commonly used as a verification benchmark. Again, we use balls for the initial sets with $c \sim \mathsf{Unif}([0.3, 1.3] \times [0.3, 1.3])$ and $r \sim \mathsf{Unif}([0, 0.5])$. The time bound is set to $T = 10$, and time step is $\Delta t = 0.05$.

F-16 Ground Collision Avoidance System [28] is a challenging benchmark for formal analysis tools. This system consists of 16 state variables (See Table 1 in [28]) among which `Vt` and `alt` are air speed and altitude. The key safety property of interest is ground collision avoidance, and therefore, in our experiments we focus on estimating the reachset only for `Vt` and `alt`. We consider initial uncertainty in up to 6 state variables, $[\texttt{Vt}, \alpha, \phi, \psi, \texttt{Q}, \texttt{alt}]$. The function `simulate(x0)` is designed to return projections of trajectories to `Vt` and `alt`, while `sample_X0()` returns 6-dimensional initial sets. We restrict the initial set to be hyper-rectangles as in [28]. An initial set $\mathcal{X}_0$ is determined by a center $c \in \mathbb{R}^6$ and a radius $r \in \mathbb{R}^6$ with $\mathcal{X}_0 = \{x \in \mathbb{R}^6 : c - r \leq x \leq c + r\}$. As for the distribution, we choose $c \sim \mathsf{Unif}([560, 600] \times [-0.1, 0.1] \times [0, \frac{\pi}{4}] \times [-\frac{\pi}{4}, \frac{\pi}{4}] \times [-0.1, 0.1] \times [70, 80])$ and $r \sim \mathsf{Unif}([0, 10] \times [0, 0.1] \times [0, \frac{\pi}{16}] \times [0, \frac{\pi}{8}] \times [0, 0.1] \times [0, 1])$. The time bound is set to $T = 20$, and time step is $\Delta t = \frac{1}{30}$. However, DryVR does not support hyper-rectangles as initial sets. Thus, we also use another setting for comparison where the initial sets are balls. To do this, we sample balls from a cube with $c \sim \mathsf{Unif}([-1, 1] \times \cdots \times [-1, 1])$ and $r \sim \mathsf{Unif}([0, 0.5])$. Then, we transform this ball to the original coordinate system by scaling each dimension. This setting is shown in Fig. 2 (Left) as F-16 (Spherical).

Quadrotor controlled by a neural controller is based on [40]. The state of the quadrotor system is $x = [p_x, p_y, p_z, v_x, v_y, v_z, \theta_x, \theta_y]$, and the control input

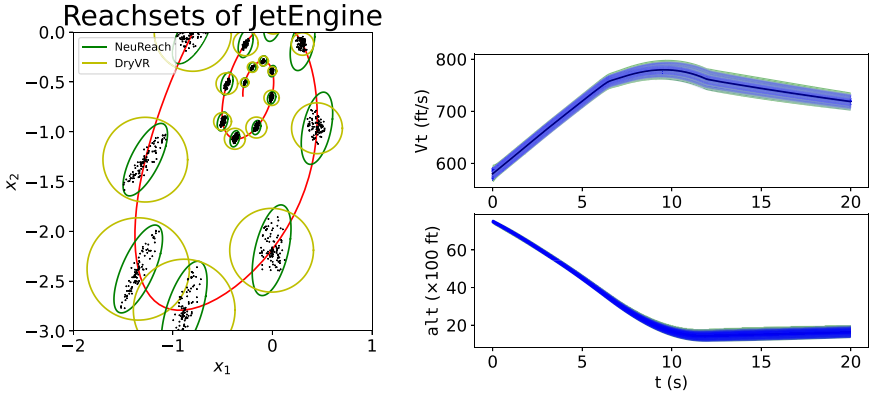

Fig. 1: Left: Some reachsets of JetEngine. Red curve is $\xi(\mathbb{E}[\mathcal{D}(\mathcal{X}_0)], \cdot)$. We randomly sample 100 trajectories starting from $\mathcal{X}_0$. Points on sampled trajectories are shown as black dots. Boundaries of the estimated reachsets at some selected time instants are shown. Clearly, ellipsoids can approximate the actual reachsets better; **Right**: A sample reachtube of F-16. Green region is the reachtube estimated by NeuReach, which is the union of all reachsets. Blue curves are sampled trajectories from the initial set. The blue region can be viewed as the actual reachtube. The estimated reachtube verifies the safety, i.e. $\texttt{alt} > 0$ always holds.

is $u := [a_z, \omega_x, \omega_y]$. We are only interested in estimating the reachability of the position variables, i.e., the first 3 dimensions of the state vector. We use balls for the initial sets with $c \sim \mathsf{Unif}([-1,1] \times \cdots \times [-1,1])$ and $r \sim \mathsf{Unif}([0, \sqrt{8}])$. The time bound is set to $T = 10$, and time step is $\Delta t = 0.05$.

5.2 Experimental results

Evaluation metrics. In order to evaluate the learned reachability function, we randomly sample 10 initial sets for testing. For each initial set $\mathcal{X}_0$, we then sample 100 trajectories starting from it. For every sampled time instant on the sampled trajectories, we check whether the state is contained in the estimated reachset and compute the empirical error (i.e., the frequency that a sample is not in the estimated reachset). In order to evaluate the conservatism of the over-approximations, we also compare the size of the over-approximations. For each initial set $\mathcal{X}_0$, we compute the total volume of the over-approximations $R(\mathcal{X}_0, t_i)$ where $t_i = \Delta t, 2\Delta t, \cdots, \lfloor \frac{T}{\Delta t} \rfloor \Delta t$. Then, the total volume averaged over 10 sampled initial sets are reported. Results are summarized in Figure 2 (Left). Please note that we use the default settings in Table 1 for all benchmarks.

All experiments were conducted on a Linux workstation with two Xeon Silver 4110 CPUs and 32 GB RAM. As shown in Figure 2 (Left), NeuReach learns an accurate reachability function for each benchmark. Please note that due to the complicated dynamics and the neural controller, the F-16 model and the quadrotor are beyond the reach of current model-based tools. As shown in Figure 1 (Right), NeuReach successfully verified the safety of the F-16 model.

Benchmark	NeuReach		DryVR	
	Volume	Error	Volume	Error
JetEngine	17.9	0.001	38.3	0.003
VanDerPol	39.2	0.001	76.4	0.002
Quadrotor	373.9	0.019	1025146.2	0.021
F-16 (Spherical)	28153.7	0.004	62651.5	0.004
F-16	31465.9	0.025	-	-

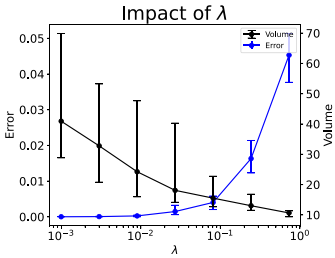

Fig. 2: Left: Volume and error of the estimated reachtube. Results are averaged over 10 random choices of $\mathcal{X}_0$; **Right**: Impact of λ. Error bars are the range over 10 runs.

Comparison with DryVR. DryVR [21] computes reachsets for spherical initial sets by learning a piece-wise exponential discrepancy (PED) function that bounds the sensitivity of the trajectories to the initial state. This function is of the form:

$$\beta(r,t) = rKe^{\sum_{j=1}^{i-1}\gamma_j(t_i-t_{i-1}+\gamma_i(t-t_{i-1})}, \forall t \in [t_{i-1}, t_i],$$

where r is the radius of the initial set, $[t_{i-1}, t_i]$ is the i-th time interval, and K, γ are learned parameters. For an spherical initial set $\mathcal{X}_0 = \mathcal{B}(c,r)$, the computed reachset is $R(\mathcal{B}(c,r),t) := \mathcal{B}(\xi(\mathbb{E}\left[\mathcal{D}(\mathcal{B}(c,r))\right],t), \beta(r,t))$, where $\mathcal{B}(c,r)$ is a ball centered at c with radius r. It is important to recall that, similar to other reachability tools, for every new initial set $\mathcal{X}_0$, DryVR computes the PED function and the reachset from scratch. For a fair comparison, we compute the parameters K and γ on the exact same training set as the one used in NeuReach and reuse the resulting PED for further queries.

Accuracy and conservatism. As shown in Figure 2 (Left), the reachsets estimated by NeuReach are tighter and more accurate than those computed by DryVR. There are two reasons for this. First, DryVR uses piece-wise exponential functions to capture the relationship between the initial radius and the radius at time t, while NeuReach uses more expressive neural networks. Second, the use of ellipsoids allows coordinate-specific accuracy. As seen in Figure 1, the reachset of JetEngine is not a perfect circle even if the initial set is a circle. Ellipsoids can approximate the actual reachsets better.

Running time. As expected, the training phase of NeuReach takes several minutes, but once a reachability function has been learned, computation of the reachset from a new initial set is very fast. For the quadrotor system, for example, this takes $\sim$ 0.3 ms on the aforementioned workstation. We believe that this makes NeuReach suitable for online safety checking and motion planning.

Impact of the hyper-parameter λ. λ influences the error and volume of the reachsets computed by NeuReach. Figure 2 (Right) shows the result of running NeuReach on JetEngine with different settings of λ. As expected, larger λ results in smaller reachsets but hurts the accuracy. On the other hand, we do not need

to tune λ case by case. Note that we use $\lambda = 0.03$ for all the results in Figure 2 (Left), and it works reasonably well for all our benchmarks.

6 Conclusion

In this paper, we presented a tool for computing reachability of systems using machine learning. NeuReach can learn accurate reachability functions for complex nonlinear systems, including some that are beyond existing methods. From a learned reachability function, arbitrary reachtubes can be computed in milliseconds. There are several limitations in the current implementation of NeuReach. First, the simulator is assumed to be deterministic—this can be too restrictive for autonomous systems with complex perception and vehicle models. We plan to extend the theory and implementation to support more general simulators. Secondly, the over-approximations are restricted to be represented as ellipsoids. Other representations will be supported in the future.

References

1. dReach. http://dreal.github.io/dReach/
2. Althoff, M., Grebenyuk, D.: Implementation of interval arithmetic in CORA 2016. In: Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems. pp. 91–105 (2016)
3. Althoff, M., Grebenyuk, D., Kochdumper, N.: Implementation of Taylor models in cora 2018. In: Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems (2018). https://doi.org/10.29007/zzc7
4. Aylward, E.M., Parrilo, P.A., Slotine, J.J.E.: Stability and robustness analysis of nonlinear systems via contraction metrics and sos programming. Automatica **44**(8), 2163–2170 (2008)
5. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control. pp. 173–178. ACM (2017)
6. Bansal, S., Chen, M., Herbert, S., Tomlin, C.J.: Hamilton-jacobi reachability: A brief overview and recent advances. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC). pp. 2242–2253. IEEE (2017)
7. Bansal, S., Tomlin, C.: Deepreach: A deep learning approach to high-dimensional reachability. arXiv preprint arXiv:2011.02082 (2020)
8. Bartlett, P.: Lecture notes in theoretical statistics (February 2013), https://www.stat.berkeley.edu/~bartlett/courses/2013spring-stat210b/notes/14notes.pdf
9. Berndt, A., Alanwar, A., Johansson, K.H., Sandberg, H.: Data-driven set-based estimation using matrix zonotopes with set containment guarantees. arXiv preprint arXiv:2101.10784 (2021)
10. Bortolussi, L., Cairoli, F., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural predictive monitoring. In: International Conference on Runtime Verification. pp. 129–147. Springer (2019)
11. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: CAV. pp. 258–263. Springer (2013)

12. Cyranka, J., Islam, M.A., Byrne, G., Jones, P., Smolka, S.A., Grosu, R.: Lagrangian reachabililty. In: International Conference on Computer Aided Verification. pp. 379–400. Springer (2017)
13. Devonport, A., Arcak, M.: Data-driven reachable set computation using adaptive gaussian process classification and monte carlo methods. In: 2020 American Control Conference (ACC). pp. 2629–2634. IEEE (2020)
14. Devonport, A., Arcak, M.: Estimating reachable sets with scenario optimization. In: Learning for dynamics and control. pp. 75–84. PMLR (2020)
15. Devonport, A., Khaled, M., Arcak, M., Zamani, M.: PIRK: scalable interval reachability analysis for high-dimensional nonlinear systems. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 556–568. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_27, https://doi.org/10.1007/978-3-030-53288-8_27
16. Donzé, A., Jin, X., Deshmukh, J.V., Seshia, S.A.: Automotive systems requirement mining using breach. In: American Control Conference, ACC 2015, Chicago, IL, USA, July 1-3, 2015. p. 4097. IEEE (2015). https://doi.org/10.1109/ACC.2015.7171970, https://doi.org/10.1109/ACC.2015.7171970
17. Duggirala, P.S., Mitra, S., Viswanathan, M.: Verification of annotated models from executions. In: EMSOFT (2013)
18. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2e2: A verification tool for stateflow models. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 68–82. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
19. Everett, M., Habibi, G., Sun, C., How, J.P.: Reachability analysis of neural feedback loops. IEEE Access **9**, 163938–163953 (2021)
20. Fan, C., Kapinski, J., Jin, X., Mitra, S.: Locally optimal reach set overapproximation for nonlinear systems. In: EMSOFT. pp. 6:1–6:10. ACM (2016)
21. Fan, C., Qi, B., Mitra, S., Viswanathan, M.: Data-driven verification and compositional reasoning for automotive systems. In: Computer Aided Verification. pp. 441–461. Springer International Publishing (2017)
22. Fan, C., Qi, B., Mitra, S., Viswanathan, M., Duggirala, P.S.: Automatic reachability analysis for nonlinear hybrid models with C2E2. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I. pp. 531–538 (2016). https://doi.org/10.1007/978-3-319-41528-4_29, https://doi.org/10.1007/978-3-319-41528-4_29
23. Fan, D.D., Agha-mohammadi, A.a., Theodorou, E.A.: Deep learning tubes for tube mpc. arXiv preprint arXiv:2002.01587 (2020)
24. Fijalkow, N., Ouaknine, J., Pouly, A., Sousa-Pinto, J.a., Worrell, J.: On the decidability of reachability in linear time-invariant systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. p. 77–86. HSCC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3302504.3311796, https://doi.org/10.1145/3302504.3311796
25. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification. Lecture Notes in Computer Science, vol. 6806, pp. 379–395. Springer (2011)

26. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: International Joint Conference on Automated Reasoning. pp. 286–300. Springer (2012)
27. Gurung, A., Ray, R., Bartocci, E., Bogomolov, S., Grosu, R.: Parallel reachability analysis of hybrid systems in xspeed. Int. J. Softw. Tools Technol. Transf. **21**(4), 401–423 (2019). https://doi.org/10.1007/s10009-018-0485-6, https://doi.org/10.1007/s10009-018-0485-6
28. Heidlauf, P., Collins, A., Bolender, M., Bak, S.: Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In: ARCH@ ADHS. pp. 208–217 (2018)
29. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: ACM Symposium on Theory of Computing. pp. 373–382 (1995), citeseer.nj.nec.com/henzinger95whats.html
30. Jiang, F., Chou, G., Chen, M., Tomlin, C.J.: Using neural networks to compute approximate and guaranteed feasible hamilton-jacobi-bellman pde solutions. arXiv preprint arXiv:1611.03158 (2016)
31. Ledoux, M., Talagrand, M.: Probability in Banach Spaces: isoperimetry and processes. Springer Science & Business Media (2013)
32. Lew, T., Pavone, M.: Sampling-based reachability analysis: A random set theory approach with adversarial sampling. arXiv preprint arXiv:2008.10180 (2020)
33. Maidens, J., Arcak, M.: Reachability analysis of nonlinear systems using matrix measures. Automatic Control, IEEE Transactions on **60**(1), 265–270 (2015)
34. Mitra, S.: Verifying Cyber-Physical Systems: A Path to Safe Autonomy. MIT Press (2021), https://mitpress.mit.edu/contributors/sayan-mitra
35. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of machine learning. MIT press (2018)
36. Niarchos, K., Lygeros, J.: A neural approximation to continuous time reachability computations. In: Proceedings of the 45th IEEE Conference on Decision and Control. pp. 6313–6318. IEEE (2006)
37. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
38. Phan, D., Paoletti, N., Zhang, T., Grosu, R., Smolka, S.A., Stoller, S.D.: Neural state classification for hybrid systems. In: International Symposium on Automated Technology for Verification and Analysis. pp. 422–440. Springer (2018)
39. Shmarov, F., Zuliani, P.: Probreach: verified probabilistic delta-reachability for stochastic hybrid systems. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control. pp. 134–139 (2015)
40. Sun, D., Jha, S., Fan, C.: Learning certified control using contraction metric. arXiv preprint arXiv:2011.12569 (2020)
41. Wang, Q., Zuliani, P., Kong, S., Gao, S., Clarke, E.M.: Sreach: A bounded model checker for stochastic hybrid systems. arXiv preprint arXiv:1404.7206 (2014)
42. Willems, J.C.: The behavioral approach to open and interconnected systems. IEEE Control Systems Magazine **27**(6), 46–99 (2007). https://doi.org/10.1109/MCS.2007.906923

43. Xue, B., Easwaran, A., Cho, N.J., Fränzle, M.: Reach-avoid verification for nonlinear systems based on boundary analysis. IEEE Transactions on Automatic Control **62**(7), 3518–3523 (2016)
44. Xue, B., Zhang, M., Easwaran, A., Li, Q.: PAC model checking of black-box continuous-time dynamical systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(11), 3944–3955 (2020)