

Learning Density Distribution of Reachable States for Autonomous Systems (Supplementary Material)

A Generalization error bound for the learning framework

With sufficient amount of data and a large enough neural network, we can approximate the state and density estimation at arbitrary small errors [76]. In the language of statistical learning theory, the neural network generating functions (Φ_ω, G_θ) is called a *hypothesis* and denoted by h . The set containing all the possible hypotheses is called the hypothesis class \mathcal{H} . For a hypothesis h generating (Φ_ω, G_θ) , we denote $l(h, \xi_i) = \sum_{(x_i^k, k\Delta T) \in \xi_i} (\Phi_\omega(x_0^i, k\Delta T) - x_k^i)^2 + (\frac{\partial G_\theta(x_0^i, k\Delta T)}{\partial t} + G_\theta(x_0^i, k\Delta T) \cdot (\nabla \cdot f(x_k^i)))^2 - \gamma$ where $\gamma \geq 0$ is an error tolerance term which is further used to derive the probabilistic guarantee. Assume that the optimization problem in Eq.(3) is feasible, and $\hat{\omega}$ and $\hat{\theta}$ solve Eq. (3). Let \hat{h}_N be the hypothesis that generates $(\Phi_{\hat{\omega}}, G_{\hat{\theta}})$. Furthermore, assume that $|l(\cdot, \cdot)| \leq B_l$, and denote the sample distribution \mathcal{D} (where the training sample trajectories are sampled from). Then according to Theorem 5 in [77], the following statement holds with probability at least $1 - \delta$ over a training data set consisting of N i.i.d. random trajectories:

$$\mathbb{P}_{\xi \sim \mathcal{D}}(\mathbb{E} l(\hat{h}_N, \xi) > 0) \leq K \left(\frac{\log^3 N}{\gamma^2} \mathfrak{R}_N^2(\mathcal{H}) + \frac{2 \log(\log(4B_l/\gamma)/\delta)}{N} \right) \quad (8)$$

where K is a universal constant, and $\mathfrak{R}_N(\mathcal{H})$ is the Rademacher complexity for \mathcal{H} defined as:

$$\mathfrak{R}_N(\mathcal{H}) = \sup_{\xi_1, \xi_2, \dots, \xi_N} \left[\mathbb{E}_\sigma \left[\sup_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \sigma_i l(h, \xi_i) \right] \right] \quad (9)$$

where $\sigma = [\sigma_1, \sigma_2, \dots, \sigma_N]$ are i.i.d. random variables with $\mathbb{P}(\sigma_i = 1) = \mathbb{P}(\sigma_i = -1) = 0.5$.

Remarks: Here we reduce bounding the generalization error to bounding the Rademacher complexity $\mathfrak{R}_N(\mathcal{H})$, where $\mathfrak{R}_N(\mathcal{H})$ can be further bounded as $\mathfrak{R}_N(\mathcal{H}) \leq o(\frac{k}{N})$ for Lipschitz parametric function classes (including neural networks) where k denotes the number of learnable parameters [78][Theorem 4.2.]. In this way, we show that for a fixed error threshold γ , as the number of training samples N increases, the probability that our learning framework fails to satisfy the Liouville equation or fails to estimate the system dynamics will gradually decrease to zero. We show an empirical result to support this in Figure 1. For the Van der Pol Oscillator benchmark example, we train the neural network with different numbers of training samples (from $8 \times 10^0 \sim 8 \times 10^4$) and report the testing error (mean square error for the state estimation and density concentration function comparing to the groundtruth) for a fixed testing set. As the number of training samples increases, the testing error gradually converges to zero.

Assume the functions on the right hand side of Eq. (2) are uniformly Lipschitz continuous in (x, ρ) , then the function will have a unique solution according to Picard-Lindelöf theorem[79][Theorem I.3.1]. Then if our estimator satisfies the Liouville equation everywhere, we can recover the groundtruth density concentration function as well as the system dynamics.

B Implementation details for system reachable set probability computation using RPM

B.1 Online query set probability bound computation under different initial state distributions

The problem formulation is: given a query set R^q with *density concentration function* constraints $[z_{min}, z_{max}]$ (the range that the *density concentration function* can change from the initial condition to the terminal condition; if this constraint is not specified, the default value is $-\infty \leq z \leq \infty$), compute the probability that the system will reach this query set (with optional density constraints).

In our case, when using RPM to compute the reachable sets, we represent R^q as a polyhedron, and since $[z_{min} \leq z \leq z_{max}]$ is a set of linear inequality constraints, the set $M^q = \{(z, v) | z_{min} \leq z \leq$

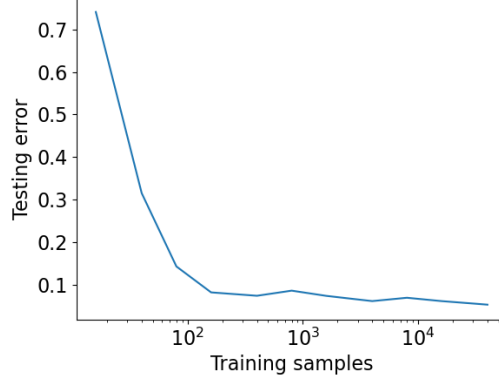


Figure 1: The testing error decreases as more training samples are used.

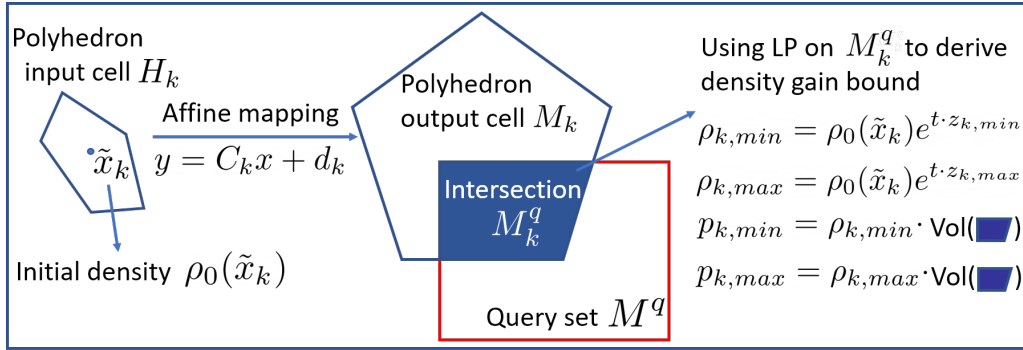


Figure 2: Illustration for the online query set probability bound computation (for an output cell M_k).

375 $z_{max}, v \in R^q$ is also a polyhedron. At each time step t , from Sec. 3.2 we can represent the NN
 376 input cells, affine mapping and output cells at this time step as $\{(A_k, b_k, C_k, d_k, E_k, f_k)\}_{k=1}^N$ (here
 377 we omit the subscript for t for the brevity in the notation) where each input cell is a polyhedron
 378 $H_k = \{x \in \mathbb{R}^{d+1} | A_k x \leq b_k\}$, with an affine mapping $y = C_k x + d_k$ and the resulting output cell
 379 is also polyhedron $M_k = \{y \in \mathbb{R}^{d+1} | E_k y \leq f_k\}$. Then for each output cell M_k , we check for the
 380 intersection between the query cell and the output cell $M_k^q = \{y | y \in M^q, E_k y \leq f_k\}$. Next we can
 381 derive the intermediate *density concentration function* bound $[z_{k,min}, z_{k,max}]$ on M_k^q by solving the
 382 following Linear Programming problem (here taking $z_{k,min}$ as an example; to solve $z_{k,max}$ we just
 383 need to change the “min” to “max” in the objective function in Eq. 10; and here $[y]_0$ denotes the first
 384 coordinate of y , thus $[y]_0 = z$ as we denote $y = (z, x)^T$):

$$\begin{cases} \min [y]_0 \\ \text{s.t. } y \in M_k^q \end{cases} \quad (10)$$

385 After we derive the bound for z on M_k^q , the density bound for M_k^q is computed as (similar to Eq.(6)):
 386

$$\begin{cases} \rho_{k,min} = \rho_0(\tilde{x}_k) e^{t \cdot z_{k,min}} \\ \rho_{k,max} = \rho_0(\tilde{x}_k) e^{t \cdot z_{k,max}} \end{cases} \quad (11)$$

387 where \tilde{x}_k is the center of H_k . And the probability bound can be computed by $p_{k,min} =$
 388 $\rho_{k,min} \cdot \text{Vol}(M_k^q)$ and $p_{k,max} = \rho_{k,max} \cdot \text{Vol}(M_k^q)$ where the $\text{Vol}(M_k^q)$ is the volume for the
 389 intersection. Finally the probability of the system reach this query set at time t is bounded by

$$390 \left[P_{min} = \sum_{k=1}^N p_{k,min}, P_{max} = \sum_{k=1}^N p_{k,max} \right]. \text{ An illustrative figure is shown in Fig. 2}$$

391 **Remarks:** This algorithm can be used for online safety verification under different initial state
 392 distributions by just representing the dangerous set in R^q , and changing the $\rho_0(\cdot)$ function in (11) on

the fly. Here we approximate the density distribution in H_k using the density evaluated at \tilde{x}_k which is the center of H_k - the accuracy of this approximation will converge to 1 as the partition on ρ_0 gets finer.

B.2 Backward reachable set probability computation

The problem formulation is: given a query set R^q with *density concentration function* constraints $[z_{min}, z_{max}]$ (the range that the *density concentration function* can change from the initial condition to the terminal condition; if this constraint is not specified, the default value is $-\infty \leq z \leq \infty$), compute for all possible initial conditions as well as probabilities that lead the system to reach the query set (with optional density constraints).

Similar to Sec. B.1, we can denote this query set as $M^q = \{(z, v) | z_{min} \leq z \leq z_{max}, v \in R^q\}$. At each time step t , the NN input cells, affine mapping and output cells are $\{(A_k, b_k, C_k, d_k, E_k, f_k)\}_{k=1}^N$ where each input cell is a polyhedron $H_k = \{x \in \mathbb{R}^{d+1} | A_k x \leq b_k\}$, with an affine mapping $y = C_k x + d_k$ and the resulting output cell is also polyhedron $M_k = \{y \in \mathbb{R}^{d+1} | E_k y \leq f_k\}$. Then for each output cell M_k , we check for the intersection between the query cell and the output cell $M_k^q = \{y | y \in M^q, E_k y \leq f_k\}$. Using the affine mapping with invertible C_k ⁷, we can derive the pre-image of this intersection to be $H_k^q = \{x | x = C_k^{-1} y - C_k^{-1} d_k, y \in M_k^q\}$. Thus the reachable set can be computed using projection: $R_k^{i,q} = \{x \in \mathcal{X} | (x, t) \in H_k^q\}$ and the corresponding probability is $p_k^{i,q} = \text{Vol}(R_k^{i,q}) \rho_0(\tilde{x}_k^{i,q})$ where $\rho_0(\cdot)$ is the initial state distribution function and $\tilde{x}_k^{i,q}$ is the center of $R_k^{i,q}$. By performing this for all output cells and for all time steps t , we derive the backward reachable set $\{(R_{t,k}^{i,q}, p_{t,k}^{i,q})\}_{k=1}^N\}_{t=0}^{T-1}$.

B.3 Speed up the probability computation by using hyper-rectangle heuristic

The computation in both Sec. B.1 and Sec. B.2 requires checking the intersection between polyhedral H_i and H_j , where one approach is to check whether a feasible solution exists for the linear programming problem: $\min 0^T x$, s.t. $x \in H_i \cap H_j$. Solving this for $x \in \mathbb{R}^n$ requires $O(n^{2.5})$ time when the interior method is used. To speed up the intersection checking process, we introduce a hyper-rectangle heuristic: at the pre-processing stage, we over-approximate each polyhedron H_i by its outer hyper-rectangle \tilde{H}_i (derived by computing the range for the vertices of H_i in each dimension). When checking for the polyhedron intersection between H_i and H_j , we first check whether their corresponding hyper-rectangles \tilde{H}_i and \tilde{H}_j will intersect. If \tilde{H}_i and \tilde{H}_j do not intersect, then it is guaranteed that the polyhedra H_i and H_j won't intersect. Otherwise, we further check the intersection of H_i and H_j by using the interior method. Checking hyper-rectangles' intersection can be implemented in $O(n)$, hence greatly accelerates the computation process. A detailed computation time comparison will be presented in Sec. E.

C Simulation environments

In this section, we present the implementation details for all 10 simulation environments used in our main paper, sorted in the same order as shown in Table. 2.

C.1 Van der Pol Oscillator

Consider the Van der Pol Oscillator problem: $\frac{d^2 x}{dt^2} - \mu(1 - x^2) \frac{dx}{dt} + x = 0$ where the position variable x is a function of t and the scalar parameter μ indicates the strength of the system damping effect. By doing a transformation: $y = \dot{x}$, the original problem can be shaped to the following 2d system dynamics:

$$\begin{cases} \dot{x} = y \\ \dot{y} = \mu(1 - x^2)y - x \end{cases} \quad (12)$$

⁷In practice, C_k is in high probability to be invertible. This is because the set of all non-invertible random matrices forms a hyper-surface with Lebesgue measure zero. When C_k is singular, we can use elimination method like Fourier-Motzkin elimination as in [21] to derive the set representation in the input side.

where the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = \mu(1-x^2)$. In the simulation, we set $\mu = 1.0$, the initial state distribution as an uniform distribution $\mathcal{U}_{[-2.5, 2.5] \times [-2.5, 2.5]}$ and the time step duration $\Delta t = 0.05s$. We run each simulation for 50 time steps to collect the trajectories.

C.2 Double Integrator with an NN controller

We consider a discrete double integrator system introduced in [41]:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u_t \quad (13)$$

where $(x_t, y_t)^T$ denotes the 2d state variable, and u_t is the output of a neural network controller which is trained to mimic the behavior of an MPC controller [41, 42]. We convert the system to the continuous system with state $(x, y)^T$ and time step duration $\Delta t = 1.0s$ as :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u \quad (14)$$

and here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = 0.5 \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}$, where the $\frac{\partial u}{\partial x}$ is the gradient of the neural network controller output u with respect to the input x (and similar for $\frac{\partial u}{\partial y}$ and y) and can be calculated using automatic differentiation engine in PyTorch [71]. We set the initial state distribution as an uniform distribution $\mathcal{U}_{[-0.5, 4.0] \times [-1.0, 1.0]}$. Similar to [41, 42], we run each simulation for 10 time steps to collect the trajectories.

C.3 Kraichnan-Orszag system

The system dynamics of the Kraichnan-Orszag problem [66, 18] is defined as:

$$\begin{cases} \dot{x}_1 = x_1 x_3 \\ \dot{x}_2 = -x_2 x_3 \\ \dot{x}_3 = -x_1^2 + x_2^2 \end{cases} \quad (15)$$

and here an interesting fact is that the divergence term $\nabla \cdot f$ used in (2) is just: $\nabla \cdot f = x_3 - x_3 + 0 = 0$, which means the density along each trajectory won't change over time, and only depends on the initial state distribution. Similar to [18], we set the initial state $x(0) = (x_1(0), x_2(0), x_3(0))^T$ distribution as an Gaussian distribution with:

$$\begin{cases} x_1(0) \sim \mathcal{N}(1, 1/4^2) \\ x_2(0) \sim \mathcal{N}(0, 1/2^2) \\ x_3(0) \sim \mathcal{N}(0, 1/2^2) \end{cases} \quad (16)$$

where we further truncate the initial state within the range $\{0 \leq x_1(0) \leq 2, -2 \leq x_2(0) \leq 2, -2 \leq x_3(0) \leq 2\}$. We set the time step duration $\Delta t = 0.125s$ and run each simulation for 80 time steps to collect the trajectories.

C.4 Inverted pendulum

The inverted pendulum problem [67] is defined as $\ddot{\theta} + \frac{b}{mL^2} \dot{\theta} - \frac{g}{L} \sin \theta - \frac{1}{mL^2} u_{LQR} = 0$, where θ denotes the pendulum's relative angle to the the up-right position, m, L, g, b are pre-defined parameters and u_{LQR} denotes the output of an LQR controller [67] $u = K_1 \theta + K_2 \dot{\theta}$ where K_1 and K_2 are scalar-valued coefficients. To test for the system performance under different coefficient settings for the LQR controller, we include k_1, k_2 into the system state variable and study the following system dynamics:

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \frac{1}{m \cdot L^2} (mgL \sin \theta - b\omega + u) \\ \dot{k}_1 = 0 \\ \dot{k}_2 = 0 \end{cases} \quad (17)$$

where $u = \frac{K_1}{50} e^{k_1 \theta} + \frac{K_2}{50} e^{k_2 \omega}$. Now the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = -\frac{b}{mL^2} + \frac{1}{mL^2} \frac{\partial u}{\partial \omega} = -\frac{b}{mL^2} + \frac{K_2 e^{k_2 \omega}}{50 mL^2}$. Based on [67], we set $g = 9.80, L = 0.50, m =$

0.15, $b = 0.00$, $K_1 = -23.59$, $K_2 = -5.31$, the time step duration $\Delta t = 0.02s$. We set the initial state distribution as a uniform distribution $\mathcal{U}_{[-2.1, 2.1] \times [-5.5, 5.5] \times [-2.0, 2.0] \times [-2.0, 2.0]}$ and run each simulation for 50 time steps to collect the trajectories.

C.5 Ground robot navigation with an NN controller



Figure 3: The screenshot for robot navigation problem.

We design a ground robot navigation experiment (as shown in Fig. 3), where the objective is to reach the green region $\{(x, y) | (x - x_{goal})^2 + (y - y_{goal})^2 \leq r_{goal}^2\}$ while avoiding to enter the red region $\{(x, y) | (x - x_{obs})^2 + (y - y_{obs})^2 \leq r_{obs}^2\}$. The robot is following an Dubins car model:

$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = u_w \\ \dot{v} = u_a \end{cases} \quad (18)$$

where x, y, θ, v represent robot's x and y position, heading angle and velocity respectively. We use an NN controller to output control signals u_w, u_v . The NN controller is a feedforward NN with 2 hidden layers and 32 hidden units in each layer. We use ReLU for the intermediate activation functions and use Tanh as the activation function for the last layer to make sure the control output is always bounded. During training, we use this NN controller to collect trajectory data and do back-

propagation with the loss function: $\mathcal{L} = \sum_{i=0}^{N-1} \sum_{k=0}^{T-1} \alpha [(x_k^i - x_{goal})^2 + (y_k^i - y_{goal})^2] + \mathbb{1}\{d_{obs} < r_{obs}\} (d_{obs} - r_{obs})$ where $d_{obs} = \sqrt{(x_k^i - x_{obs})^2 + (y_k^i - y_{obs})^2}$. Here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = \frac{\partial u_w}{\partial \theta} + \frac{\partial u_a}{\partial v}$. We set the initial state distribution as an uniform distribution $\mathcal{U}_{[-1.8, -1.2] \times [-1.8, -1.2] \times [0, \pi/2] \times [1.0, 1.5]}$. We run each simulation for 50 time steps with time duration $\Delta t = 0.05s$ to collect the trajectories.

C.6 FACTEST car tracking system

Consider a rearwheel kinematic car in 2D scenarios where the dynamics is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (19)$$

and the corresponding errors are measured by:

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - \theta \end{bmatrix} \quad (20)$$

with $x_{ref}, y_{ref}, \theta_{ref}$ being some predefined tracking points (in this experiment, we assume the tracking points are not changing over time). With the following tracking controller defined in (w_{ref} and v_{ref} are referenced angular velocity and velocity respectively, k_1, k_2, k_3 are the parameters controlling how fast the system will converge to the reference point) [68]:

$$\begin{cases} v = v_{ref} \cos(e_\theta) + k_1 e_x \\ \omega = \omega_{ref} + v_{ref}(k_2 e_y + k_3 \sin(e_\theta)) \end{cases} \quad (21)$$

and with an uncertainty error in the dynamics of e_x and e_y (denoted as a), the error dynamics become:

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \\ \dot{a} \end{bmatrix} = \begin{bmatrix} (\omega_{ref} + v_{ref}(k_2 e_y + k_3 \sin(e_\theta)))e_y - k_1 e_x + a e_x \\ -(\omega_{ref} + v_{ref}(k_2 e_y + k_3 \sin(e_\theta)))e_x + v_{ref} \sin(e_\theta) + a e_y \\ -v_{ref}(k_2 e_y + k_3 \sin(e_\theta)) \\ 0 \end{bmatrix} \quad (22)$$

The uncertain parameter $a \in [0, 1]$. We will show that although now the reachable set will be much larger than the case when $a = 0$, the probability that the system does not converge to the origin (zero-error) is very low.

Here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = 2a - k_1 - k_2 v_{ref} e_x - v_{ref} k_3 \cos e_\theta$. In our experiment, we set $x_{ref} = y_{ref} = \theta_{ref} = 0$, $k_1 = k_2 = 0.5$, $k_3 = 1.0$, $w_{ref} = 0$, $v_{ref} = 1$. We set the initial state distribution as an uniform distribution $\mathcal{U}_{[-2.1, 2.1] \times [-2.1, 2.1] \times [0, 0.1] \times [0, 0.1, 0]}$. We run each simulation for 50 time steps with time duration $\Delta t = 0.10s$ to collect the trajectories.

C.7 6D Quadrotor with an NN controller

Consider a 6D quadrotor [42]:

$$\dot{x} = \begin{bmatrix} 0_{3 \times 3} & I_3 \\ 0_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} x + \begin{bmatrix} g & 0 & 0 \\ 0 & -g & 0 \\ 0 & 0 & 1 \end{bmatrix}^T u + \begin{bmatrix} 0_{5 \times 1} \\ -g \end{bmatrix} \quad (23)$$

where the state vector x contains 3D positions and velocities $[p_x, p_y, p_z, v_x, v_y, v_z]$, g is the gravity (set to $9.8m/s^2$), and the control $u = (u_1, u_2, u_3)^T$ is from the output of an NN controller taking the state vector as the input [42]. Here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = g \cdot \frac{\partial u_1}{\partial v_x} - g \cdot \frac{\partial u_2}{\partial v_y} + \frac{\partial u_3}{\partial v_z}$. Similar to [42], we set the initial state distribution as an uniform distribution $\mathcal{U}_{[4.65, 4.75] \times [4.65, 4.75] \times [2.95, 3.05] \times [0.94, 0.96] \times [-0.05, 0.05] \times [-0.5, 0.5]}$. We run each simulation for 12 time steps with time duration $\Delta t = 0.10s$ to collect the trajectories.

C.8 Adaptive cruise control system

Consider a learning-based adaptive cruise control (ACC) problem with plant dynamics [4]:

$$\begin{cases} \dot{x}_{rel} = v_{lead} - v_{ego} \\ \dot{v}_{lead} = \gamma_{lead} \\ \dot{\gamma}_{lead} = a_{lead} \\ \dot{v}_{ego} = \gamma_{ego} \\ \dot{\gamma}_{ego} = -2\gamma_{ego} + 2u(x_{rel}, v_{lead} - v_{ego} - \gamma_{ego}\tau, v_{ego} + \gamma_{ego}\tau) \\ \dot{a}_{lead} = -2\gamma_{lead} \\ \dot{\tau} = 0 \end{cases} \quad (24)$$

here x_{rel} denotes the relative distance from the leading vehicle to the ego vehicle, v_{lead} and v_{ego} denote the velocity of leading and ego vehicles and γ_{lead} and γ_{ego} denote the corresponding acceleration rates of the two vehicles (a_{lead} models the change in the leading vehicle's acceleration rate, similar to the MATLAB implementation in [4]). And the controller u is taking the relative distance, velocity, and ego vehicle's velocity as input and outputs the change in the ego vehicle's acceleration rate. We model the velocity perception uncertainty as τ and pass it through the neural network. Here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f = -2 - \frac{\partial u}{\partial (v_{lead} - v_{ego} - \gamma_{ego}\tau)} \tau + \frac{\partial u}{\partial (v_{ego} + \gamma_{ego}\tau)} \tau$. We set the initial state distribution as an uniform distribution $\mathcal{U}_{[59.0, 62.0] \times [26.0, 30.0] \times [-0.01, 0.01] \times [30.0, 30.5] \times [-0.01, 0.01] \times [-10.1, -9.9] \times [-2.0, 2.0]}$ and run each simulation for 50 time steps with time duration $\Delta t = 0.10s$ to collect the trajectories.

519 C.9 F-16 ground-collision avoidance system

520 This F-16 Ground-Collision Avoidance System (GCAS) performs a recovery maneuver for the F-16
 521 aircraft when a ground collision is detected. The F-16 aircraft is modelled with 6 degrees of freedom
 522 (DoF) associated with 13 nonlinear equations (three equations each for forces, kinematics, moments
 523 and position of the aircraft, and one extra to capture the F-16 turbojet engine). The hierarchical
 524 control system has an outer-loop autopilot controller and an inner loop tracking and stabilizing
 525 controller (ILC). More details can be found in [69]. Specifically in this experiment, the GCAS
 526 drives the roll angle and its rate to 0 and then accelerates upwards to avoid ground collision. The
 527 safety specification is to make sure the altitude is always non-negative (not hitting the ground).
 528 We collect the trajectories using the F-16 simulator provided in [69]. The trajectories has a time
 529 step duration as $0.0333s$ and has 106 time steps in total. The hierarchical controller made the
 530 closed-loop F-16 system a black-box system without a clean ODE expression. Therefore, there is
 531 no analytical way to compute for the system dynamics. As we discussed in the main paper, we could
 532 approximate the divergence of the system dynamics by using gradient perturbation. Recall that for
 533 system $\dot{x} = (f_1(x), f_2(x), \dots, f_d(x))^T$, the system divergence is $\nabla \cdot f = \sum_{i=1}^d \frac{\partial f_i}{\partial x_i}$, so we approximate
 534 the gradient for $\frac{\partial f_i(x)}{\partial x_i}$ by $(f_i(x_1, \dots, x_i + \epsilon, \dots, x_n) - f_i(x_1, \dots, x_i - \epsilon, \dots, x_n)) / (2\epsilon)$ where ϵ is a
 535 very small number and we set $\epsilon = 10^{-8}$ in our experiments.

536 C.10 8-car platooning with model error

537 In this experiment we consider a 8-car platoon model [80, 70]. The state variable is $x \in \mathbb{R}^{15}$, where
 538 x_1 represents the first vehicle's (which is also the leading vehicle in the platoon) velocity, x_{2k-1}
 539 ($k=2,3,\dots,8$) represents the relative velocity of the $k-1$ -th vehicle comparing to the k -th vehicle, and
 540 x_{2k-2} ($k=2,3,\dots,8$) represents the relative longitudinal offset of the $k-1$ -th vehicle comparing to the
 541 k -th vehicle. The dynamics of the system hence is given by:

$$\begin{aligned} \dot{x}_{2k-1} &= \begin{cases} u_1, & k = 1 \\ u_{k-1} - u_k, & k = 2, 3, \dots, 8 \end{cases} \\ \dot{x}_{2k-2} &= x_{2k-1} + w, \quad k = 2, 3, \dots, 8 \end{aligned} \quad (25)$$

542 where $u = (u_1, \dots, u_8)^T$ is the NN controller's output (for changing the vehicles' acceleration rates)
 543 and w models the noise in the vehicles' velocity dynamics. Here the neural network controller is
 544 trained via RL [81]. Here the divergence term $\nabla \cdot f$ used in (2) can be computed as: $\nabla \cdot f =$
 545 $\frac{\partial u_1}{\partial x_1} + \sum_{k=2}^8 (\frac{\partial u_{k-1}}{\partial x_{2k-1}} - \frac{\partial u_k}{\partial x_{2k-2}})$. We set the initial state distribution as an uniform distribution \mathcal{U} for
 546 $19.9 \leq x_1 \leq 20.1$, $0.9 \leq x_{2k-3} \leq 1.1$, $k = 2, 3, \dots, 8$ and $-0.1 \leq x_{2k-2} \leq 0.1$, $k = 2, 3, \dots, 8$ and
 547 $-0.01 \leq w \leq 0.01$. We run each simulation for 50 time steps with time duration $\Delta t = 0.15s$ to
 548 collect the trajectories.

549 D Forward reachable set distribution under different probability thresholds

550 Instead of over-approximating the reachable sets like traditional methods, our approach can ren-
 551 der varied sizes of reachable sets under different probability thresholds and under different initial
 552 state distributions. We compute for the varied reachable sets at $t=1.0s$ for ground robot navigation
 553 experiment under three different (truncated) multivariate Gaussian distributions: $\mathcal{N}_1 = \mathcal{N}(\mu =$
 554 $(-1.7, -1.7, 0.2, 1.4)^T; \Sigma = 0.02I)$, $\mathcal{N}_2 = \mathcal{N}(\mu = (-1.7, -1.7, 1.3, 1.4)^T; \Sigma = 0.02I)$, and
 555 $\mathcal{N}_3 = \mathcal{N}(\mu = (-1.7, -1.7, 0.2, 1.4)^T; \Sigma = 0.1I)$. The difference between \mathcal{N}_1 and \mathcal{N}_2 is the
 556 change of the mean vector, and the difference between \mathcal{N}_1 and \mathcal{N}_3 is the change in the covariance
 557 matrix. As shown in Fig. 4~Fig. 6, as the probability threshold decreases, the relative volume of the
 558 reachable set (comparing to the volume in Fig.3(a)) decreases drastically. And our approach shows
 559 that under the initial distribution \mathcal{N}_1 , a large portion of the states ($p \geq 0.8980$) actually only reside
 560 in a small region ($\text{vol}=0.03X$) in the state space (as shown in Fig. 4(e)). Whereas under different ini-
 561 tial state distributions, the concentration region might be different (comparing Fig. 4(f) and Fig. 5(f))
 562 or the degree of concentration is different (comparing Fig. 4(f) and Fig. 6(e)).

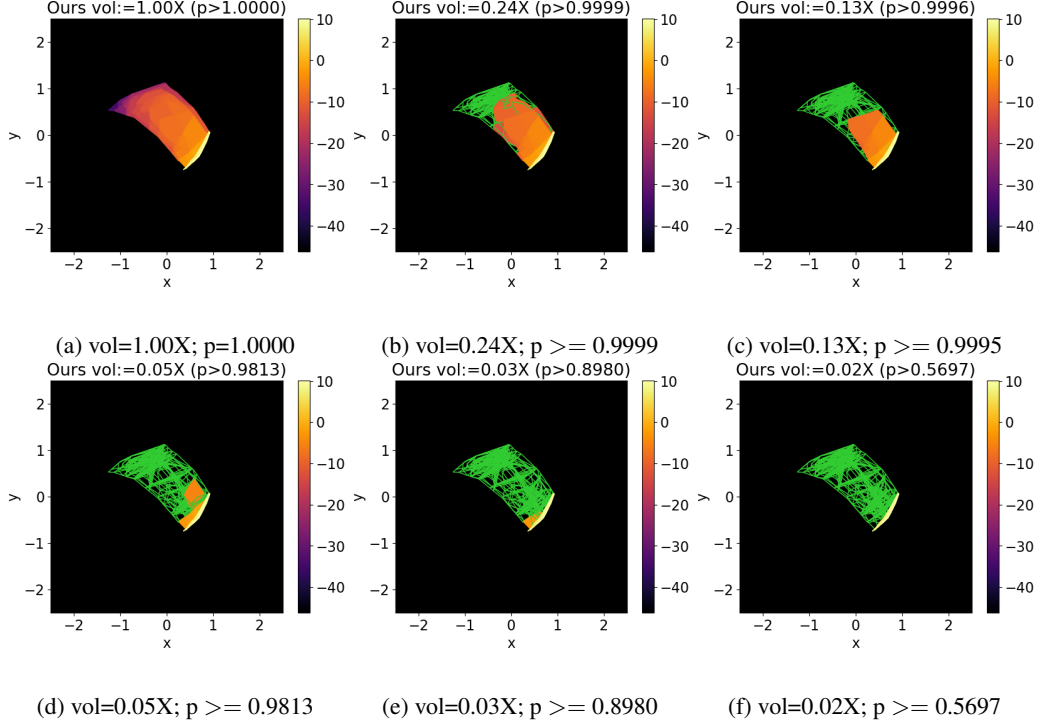


Figure 4: The system forward reach set distribution under different probability thresholds at $t=1.0s$ with initial condition $\mathcal{N}_1 = N(\mu = (-1.7, -1.7, 0.2, 1.4)^T; \Sigma = 0.02I)$. The color ranged from dark purple to light yellow indicates the density inside the polyhedral cells. The density is shown in logarithm magnitude. The edges colored in green indicate the boundaries of the RPM polyhedral cells with density below a threshold. As the probability thresholds (p) decreases, the relative volume (vol) of the reachable set decreases drastically. Our approach indicates under this distribution, the system state has large probability concentrating in the right bottom curve as shown in Fig. 4(f).

563 E Runtime for fast safety checking

	Low density $e^{-10} \leq \rho \leq e^2$		Medium density $e^2 \leq \rho \leq e^3$		High density $\rho \geq e^3$	
	Vanilla	Heuristic	Vanilla	Heuristic	Vanilla	Heuristic
Time (sec)	3.1594	0.8425	3.0854	0.8122	3.0585	0.7644
#(Rect)	-	391	-	31	-	4
#(Poly)	303	303	2	2	0	0
Is safe?	No	No	No	No	Yes	Yes

Table 2: Online safe verification comparison under different density conditions (Low / Medium / High). We measure the computation time (“Time”), number of rectangle intersections (“#(Rect)”), number of polyhedral intersections (“#(Poly)”) and whether the initial condition will avoid to drive to unsafe region (“Is safe?”) under each density condition with and without using the hyper-rectangle heuristics (“Heuristic”/“Vanilla”). As shown in Table. 2, the trajectories sampled from the initial state will only reach the unsafe region under low and medium densities while won’t reach the unsafe region in high density. Using heuristics can reduce the computation time in all conditions by 70%.

564 We also perform the system safety verification for the ground robot task. Specifically, we want
 565 to verify whether the trajectories starting from the initial condition S_{init} will drive to the unsafe
 566 region S_{unsafe} under different density conditions. We set $S_{init} = \{-1.8 \leq x \leq -1.2, -1.8 \leq$
 567 $y \leq -1.2, 0.0 \leq \theta \leq \pi/2, 0.0 \leq v \leq 1.0\}$, $S_{unsafe} = \{-0.5 \leq x \leq 0.0, -0.5 \leq y \leq$
 568 $0.0\}$ and try three different density constraints: which are low density ($e^{-10} \leq \rho \leq e^2$), medium

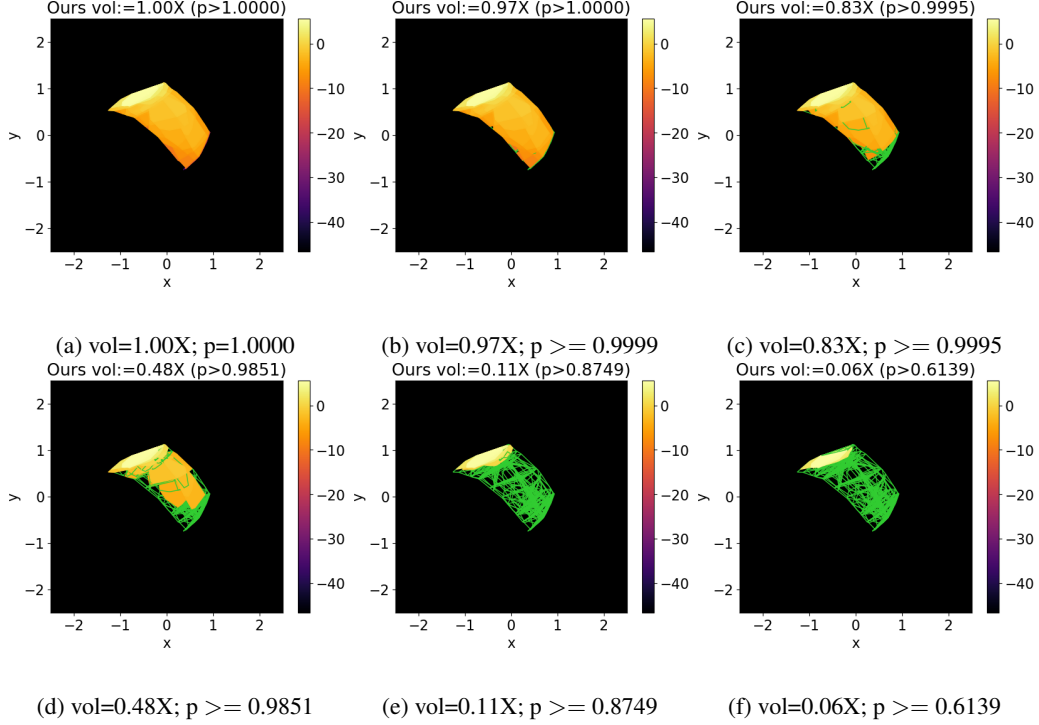


Figure 5: The system forward reach set distribution under different probability thresholds at $t=1.0s$ with initial condition $\mathcal{N}_2 = N(\mu = (-1.7, -1.7, 1.3, 1.4)^T; \Sigma = 0.02I)$. The color ranged from dark purple to light yellow indicates the density inside the polyhedral cells. The density is shown in logarithm magnitude. The edges colored in green indicate the boundaries of the RPM polyhedral cells with density below a threshold. As the probability thresholds (p) decreases, the relative volume (vol) of the reachable set decreases drastically. Our approach indicates under this distribution, the system state has large probability concentrating in the top left curve as shown in Fig. 5(f).

density ($e^2 \leq \rho \leq e^3$) and high density ($\rho \geq e^3$). We measure whether the initial condition will avoid to lead the system to reach the unsafe region under each density condition (“Is safe?”). To illustrate how the heuristic method introduced in Sec. B.3 accelerates the computation process, we also measure the computation time (“Time”), number of rectangle intersections (“#(Rect)”), number of polyhedral intersections (“#(Poly)”) and , with and without using the hyper-rectangle heuristics(“Heuristic”/“Vanilla”). Our program is implemented in Python with parallel computation deployed on a 12-core CPU.

As shown in Table. 2, the trajectories sampled from the initial state will only reach the unsafe region under low and medium densities, and won’t reach the unsafe region in high density. This can be helpful when we are considering planning problems with density constraints. Besides, our approach with hyper-rectangle heuristic can finish the online safety verification for 50 time steps in only 0.8 seconds, which reduces 70% of the computation time comparing to the vanilla algorithm. Doing safety verification for each time step only needs 0.016s, which is much smaller than the actual Δt used for the ground robot navigation benchmark ($\Delta t = 0.05s$). With code-level optimization (e.g. write the program in C++ or Julia) and more CPU cores being used in parallel, our approach can further benefit for real-time applications.

F Density (Ours, KDE, histogram, groundtruth) and reachability visualizations

Here we compare the density prediction results on all 10 benchmark examples mentioned in Table 1, and compare our reachable set result with other worst-case reachability tools (Convex Hull [9], GSG [73] and DryVR [68]) on 4 of the benchmark examples. As shown in figures in F.1, our

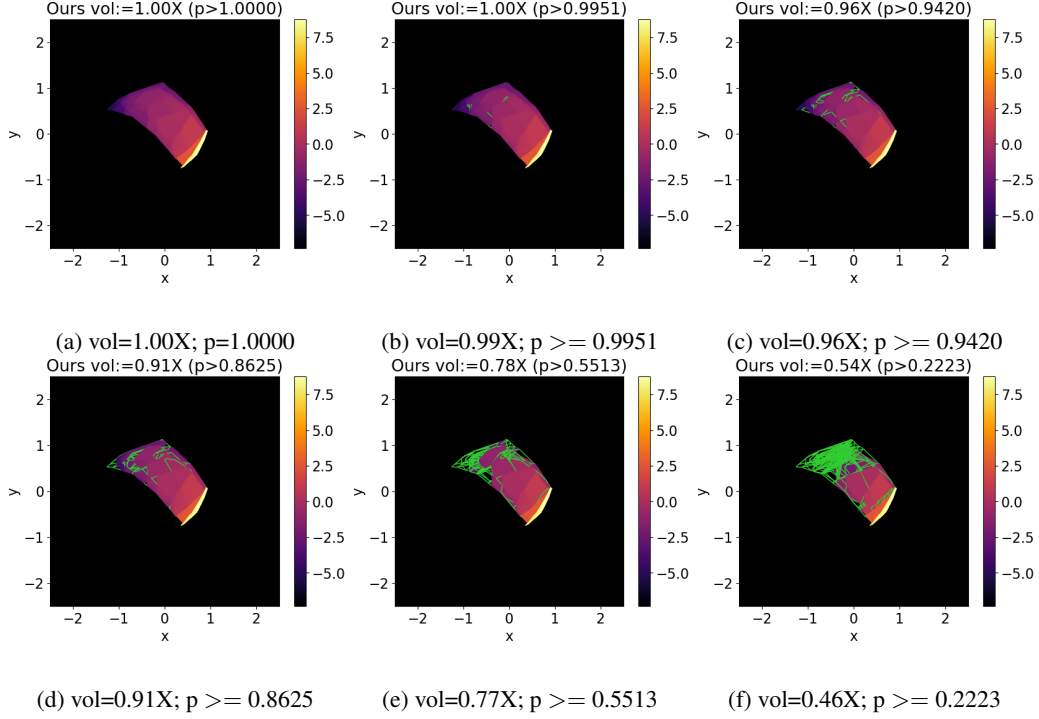


Figure 6: The system forward reach set distribution under different probability thresholds at $t=1.0s$ with initial condition $\mathcal{N}_3 = N(\mu = (-1.7, -1.7, 0.2, 1.4)^T; \Sigma = 0.1I)$. The color ranged from dark purple to light yellow indicates the density inside the polyhedral cells. The density is shown in logarithm magnitude. The edges colored in green indicate the boundaries of the RPM polyhedral cells with density below a threshold. As the probability thresholds (p) decreases, the relative volume (vol) of the reachable set decreases drastically. Our approach indicates under this distribution, the system state has large probability concentrating in the right bottom y curve as shown in Fig. 6(f), but is not as concentrated as shown in Fig. 4(f).

approach can consistently achieve the closest state density distribution among other approaches (Kernel density, histogram), and doesn't have a restriction for high-dimension systems (whereas the histogram method cannot estimate the density for high-dimension systems like in Fig. 28 ~ Fig. 36). For the reachability comparison, different from the worst-case reachability analysis tools (Convex Hull [9], GSG [73] and DryVR [68]), our approach can compute the density and probability for each of the reachable set, hence is able to tell where do states concentrate (a high probability of states only reside in a small region in the state space, as shown in Fig. 39, Fig. 42, Fig. 44, Fig. 48, etc). Our method is more precise and informative than those worst-case reachability analysis approaches. More figures can be found out in the supplementary video.

E.1 Comparison of density prediction accuracies

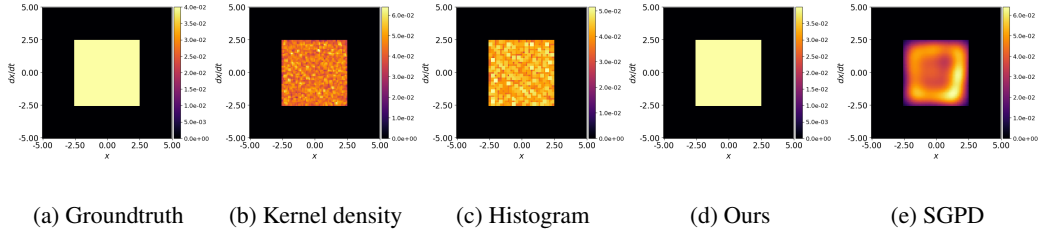
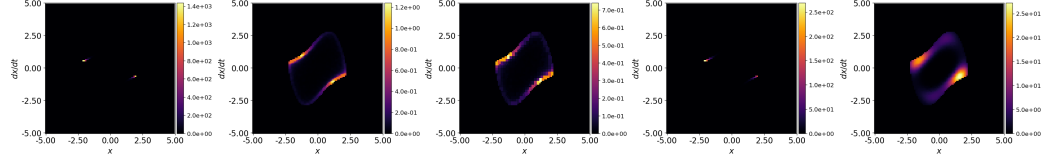
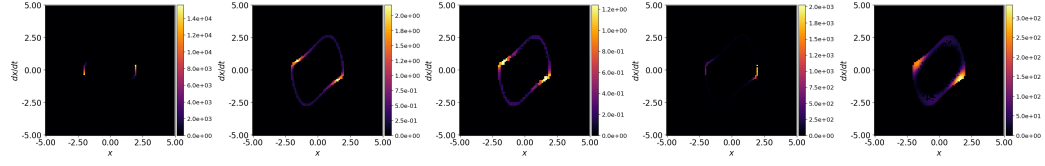


Figure 7: Comparison of density prediction accuracies (Van der Pol, $t=0$)



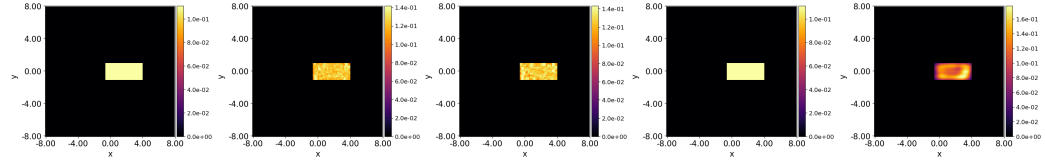
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 8: Comparison of density prediction accuracies (Van der Pol, $t=20$)



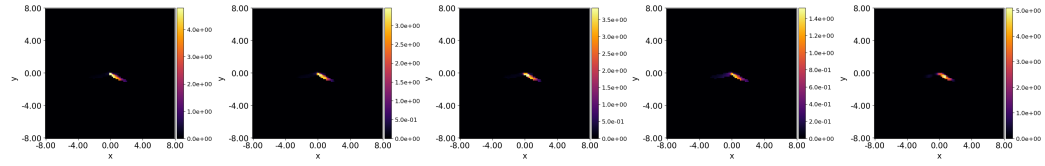
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 9: Comparison of density prediction accuracies (Van der Pol, $t=49$)



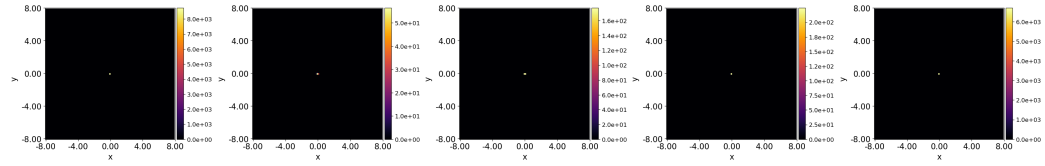
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 10: Comparison of density prediction accuracies (Double integrator, $t=0$)



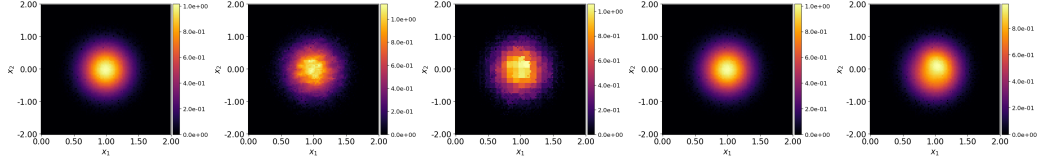
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 11: Comparison of density prediction accuracies (Double integrator, $t=3$)



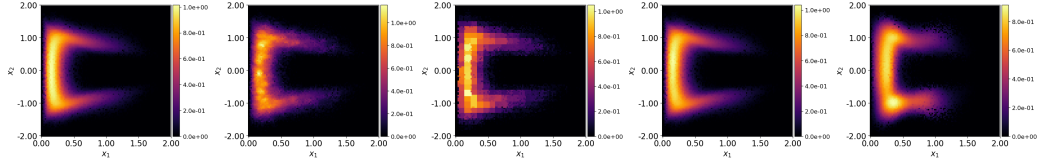
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 12: Comparison of density prediction accuracies (Double integrator, $t=9$)



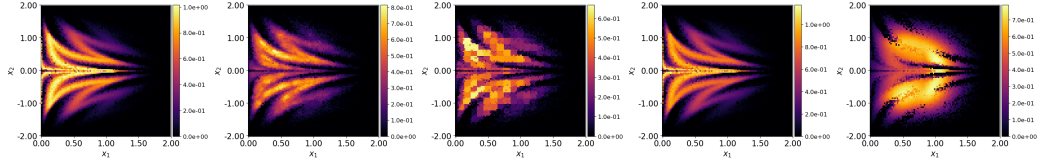
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 13: Comparison of density prediction accuracies (Kraichnan-Orszag system, $t=0$)



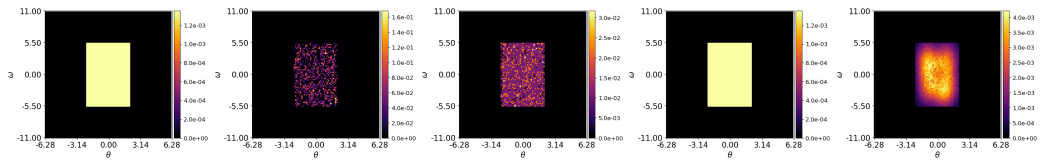
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 14: Comparison of density prediction accuracies (Kraichnan-Orszag system, $t=20$)



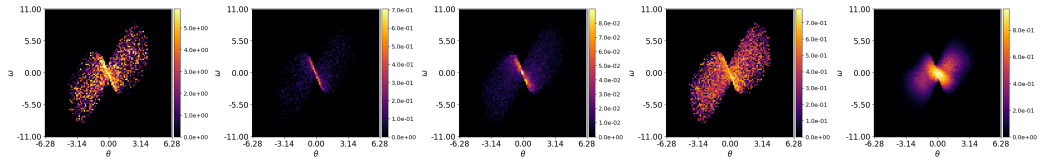
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 15: Comparison of density prediction accuracies (Kraichnan-Orszag system, $t=79$)



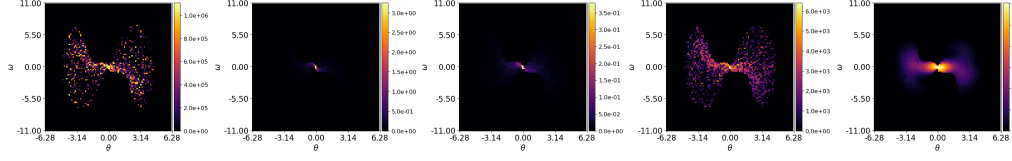
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 16: Comparison of density prediction accuracies (Inverted pendulum, $t=0$)



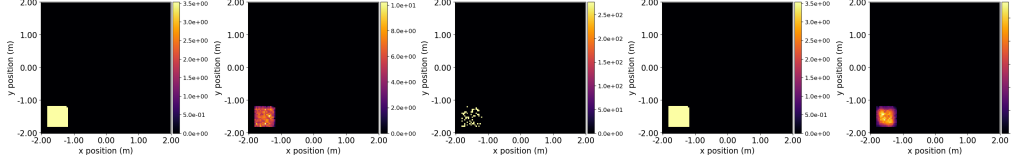
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 17: Comparison of density prediction accuracies (Inverted pendulum, $t=20$)



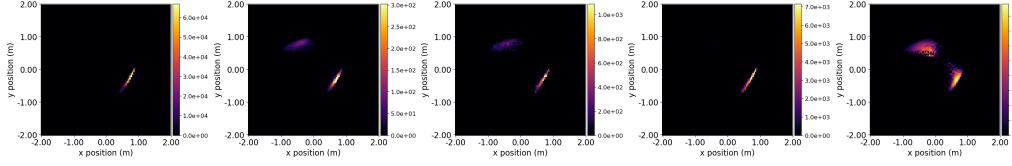
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 18: Comparison of density prediction accuracies (Inverted pendulum, $t=49$)



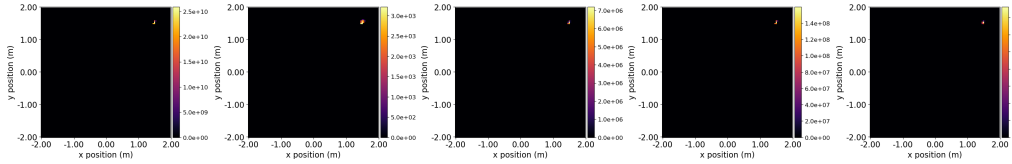
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 19: Comparison of density prediction accuracies (Ground robot navigation, $t=0$)



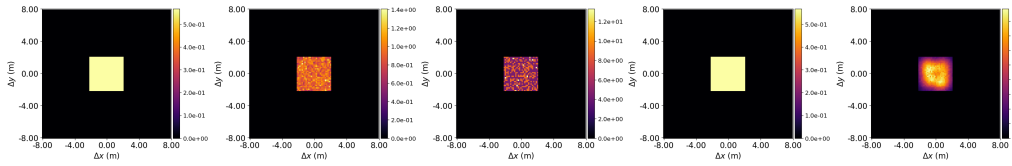
(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 20: Comparison of density prediction accuracies (Ground robot navigation, $t=20$)



(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 21: Comparison of density prediction accuracies (Ground robot navigation, $t=49$)



(a) Groundtruth (b) Kernel density (c) Histogram (d) Ours (e) SGPD

Figure 22: Comparison of density prediction accuracies (FACTEST car model, $t=0$)

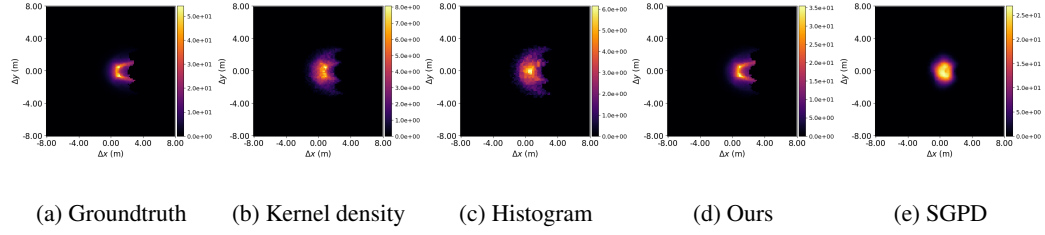


Figure 23: Comparison of density prediction accuracies (FACTEST car model, t=20)

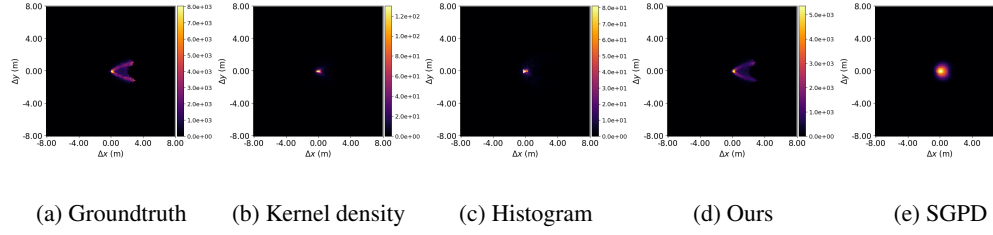


Figure 24: Comparison of density prediction accuracies (FACTEST car model, t=49)

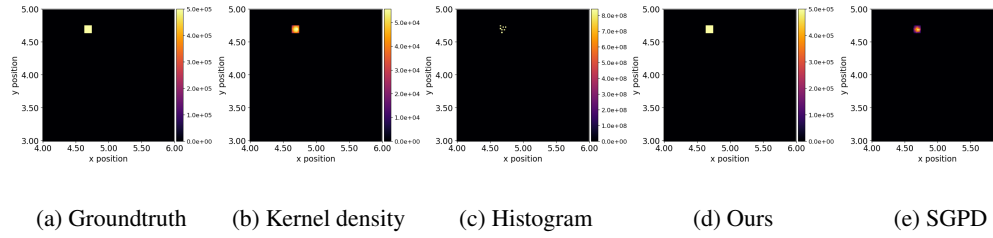


Figure 25: Comparison of density prediction accuracies (Quadrotor control system, t=0)

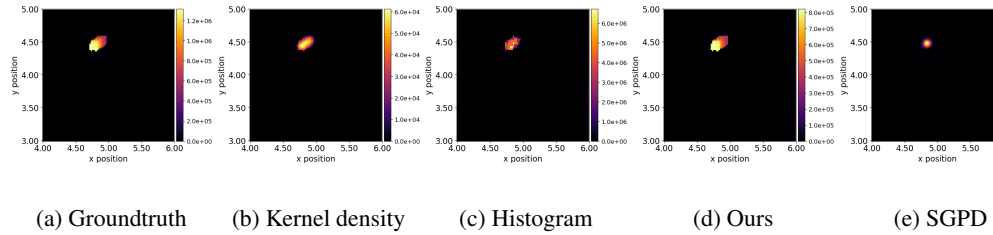


Figure 26: Comparison of density prediction accuracies (Quadrotor control system, t=4)

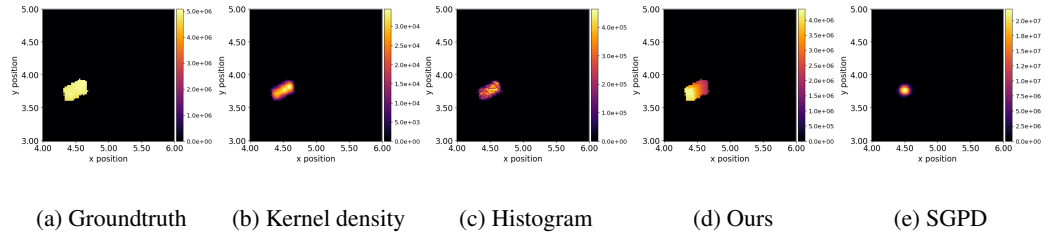


Figure 27: Comparison of density prediction accuracies (Quadrotor control system, t=11)

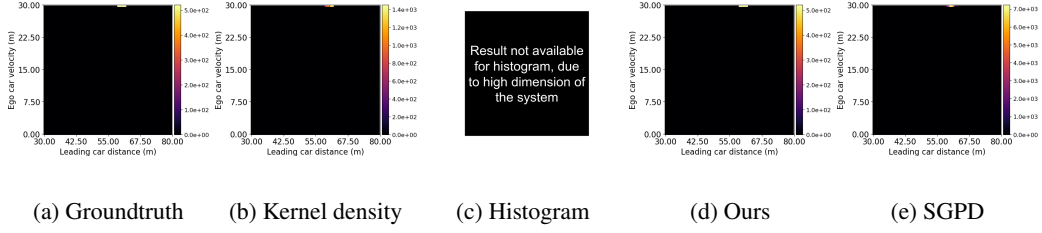


Figure 28: Comparison of density prediction accuracies (Adaptive cruise control system, $t=0$)

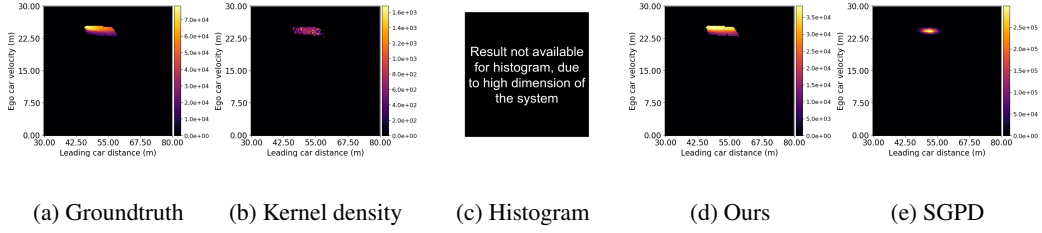


Figure 29: Comparison of density prediction accuracies (Adaptive cruise control system, $t=20$)

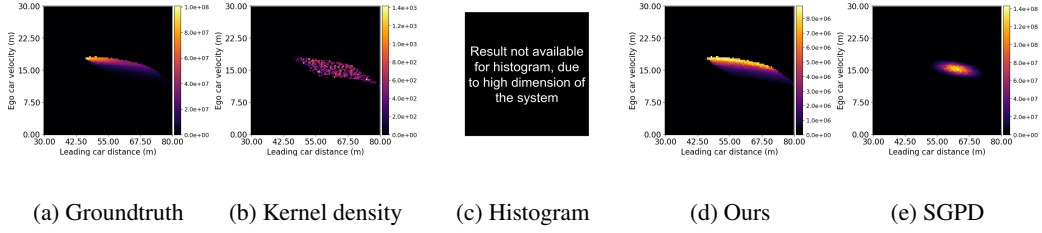


Figure 30: Comparison of density prediction accuracies (Adaptive cruise control system, $t=49$)

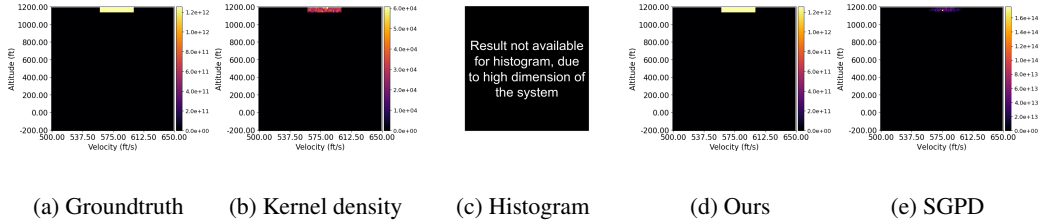


Figure 31: Comparison of density prediction accuracies (Ground collision avoidance system, $t=0$)

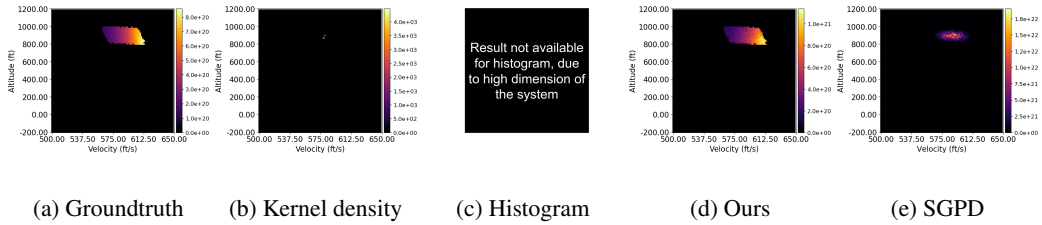


Figure 32: Comparison of density prediction accuracies (Ground collision avoidance system, $t=20$)

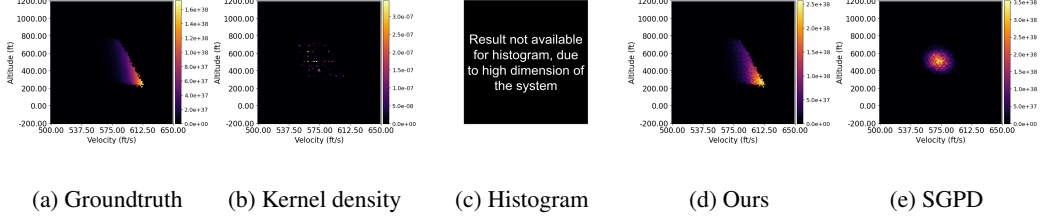


Figure 33: Comparison of density prediction accuracies (Ground collision avoidance system, $t=59$)

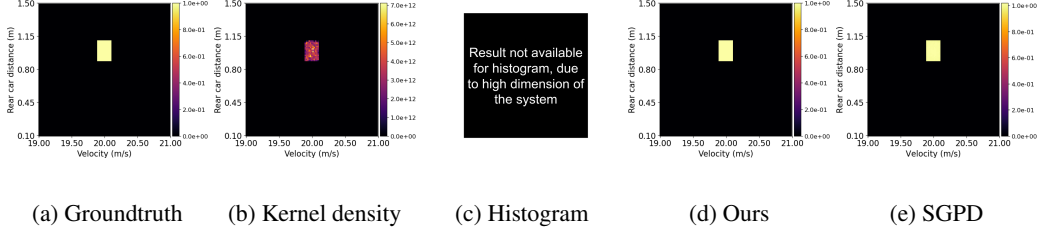


Figure 34: Comparison of density prediction accuracies (8-Car platoon system, $t=0$)

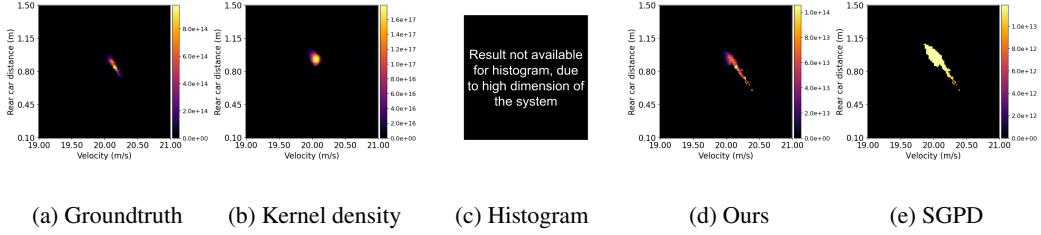


Figure 35: Comparison of density prediction accuracies (8-Car platoon system, $t=20$)

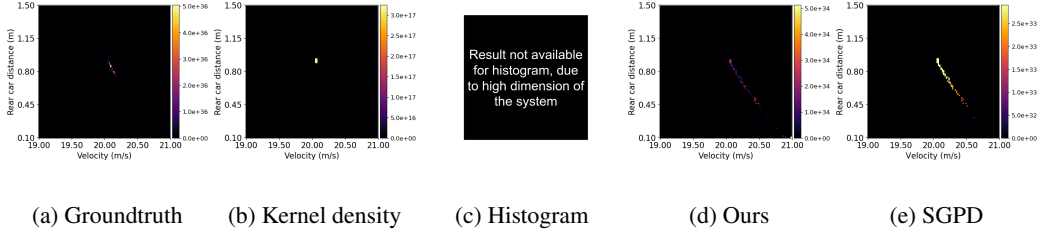


Figure 36: Comparison of density prediction accuracies (8-Car platoon system, $t=49$)

600 F.2 Comparison of reachable set computation among different tools

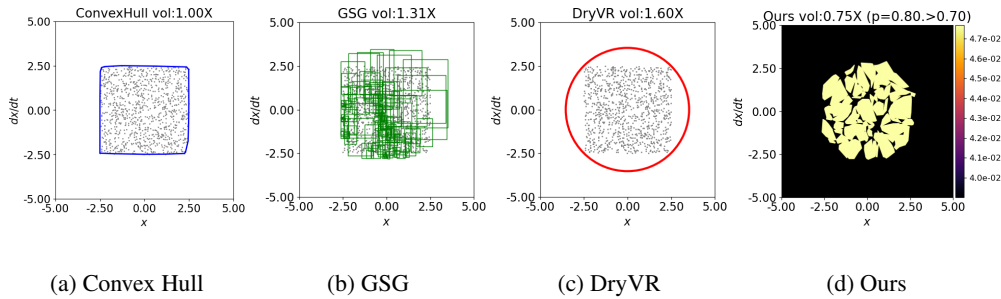


Figure 37: Comparison of reachable set computation among different tools (Van der Pol, $t=0$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

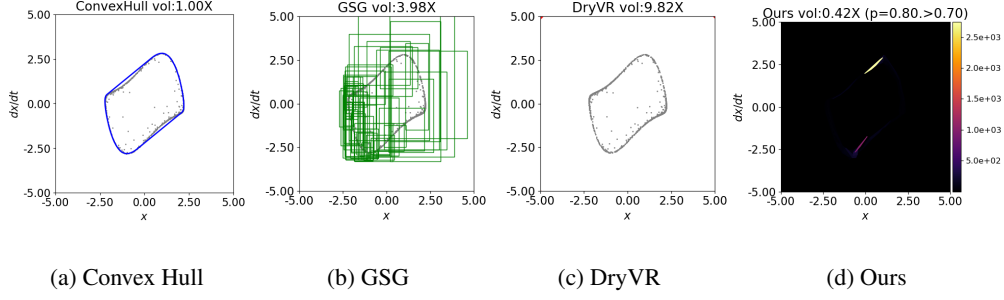


Figure 38: Comparison of reachable set computation among different tools (Van der Pol, $t=40$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

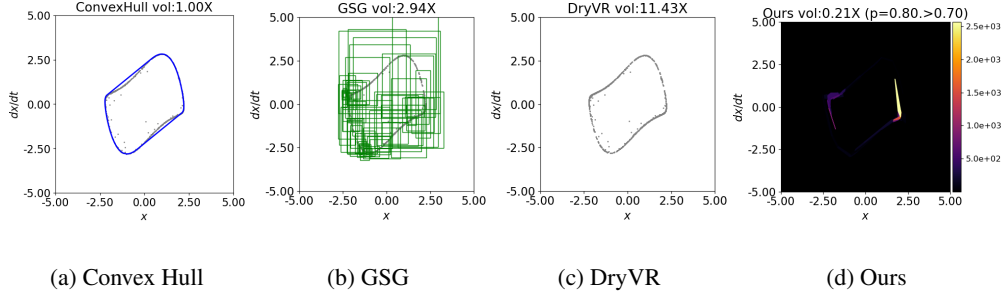


Figure 39: Comparison of reachable set computation among different tools (Van der Pol, $t=49$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

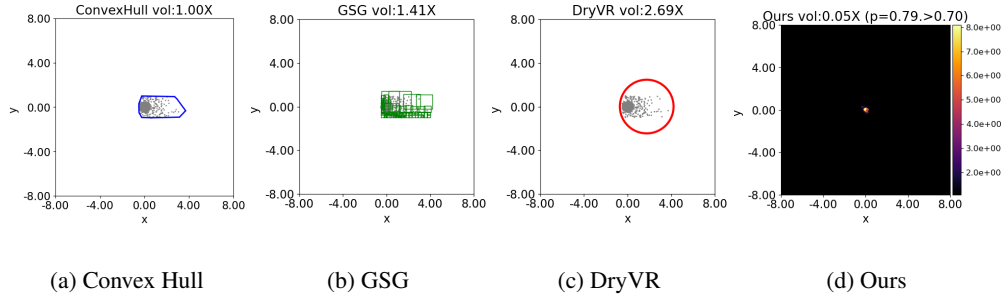


Figure 40: Comparison of reachable set computation among different tools (Double integrator, $t=0$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

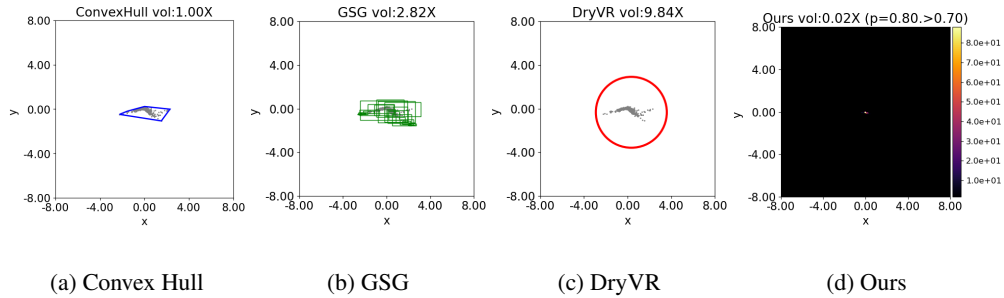


Figure 41: Comparison of reachable set computation among different tools (Double integrator, $t=3$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

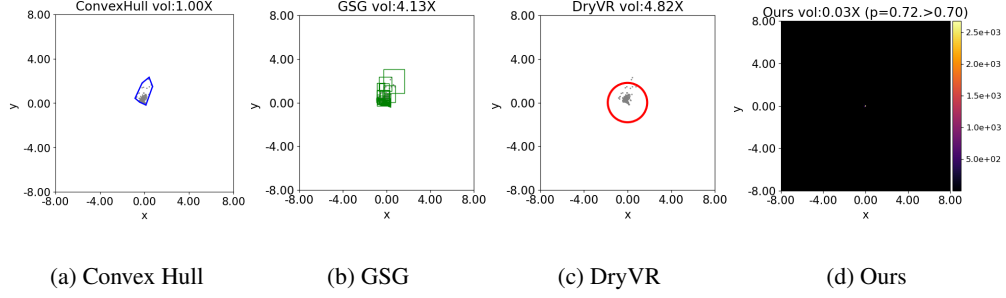


Figure 42: Comparison of reachable set computation among different tools (Double integrator, $t=7$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

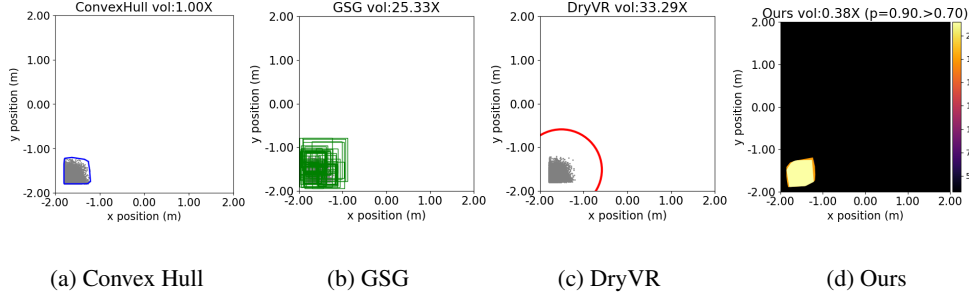


Figure 43: Comparison of reachable set computation among different tools (Ground robot navigation, $t=0$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

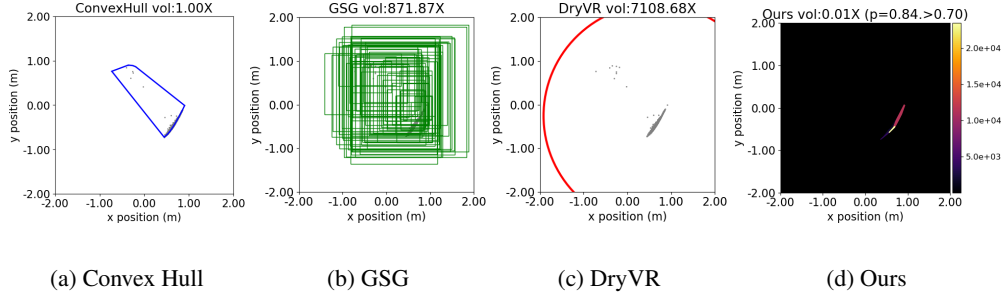


Figure 44: Comparison of reachable set computation among different tools (Ground robot navigation, $t=20$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

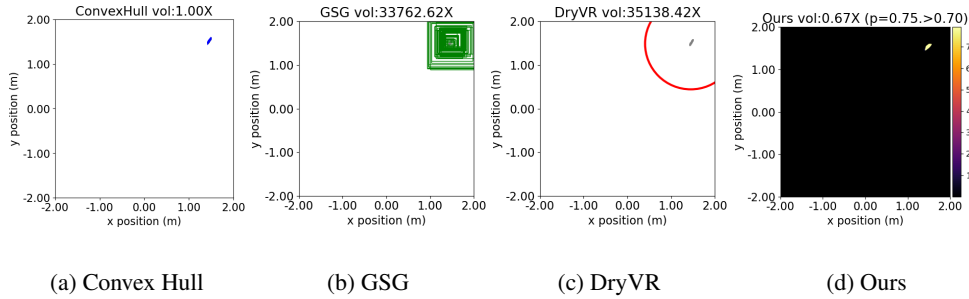


Figure 45: Comparison of reachable set computation among different tools (Ground robot navigation, $t=40$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

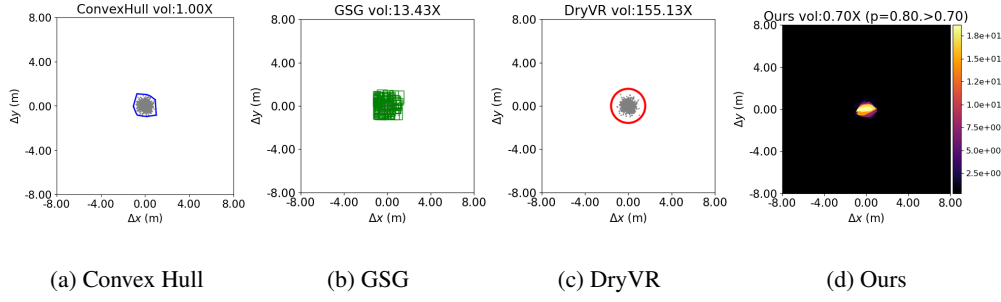


Figure 46: Comparison of reachable set computation among different tools (FACTEST car model, $t=0$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

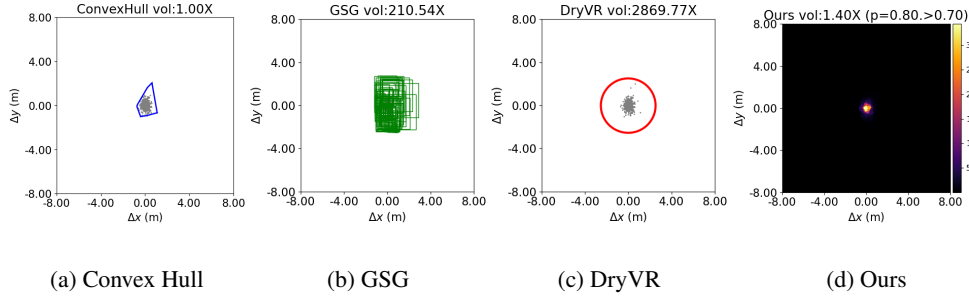


Figure 47: Comparison of reachable set computation among different tools (FACTEST car model, $t=20$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

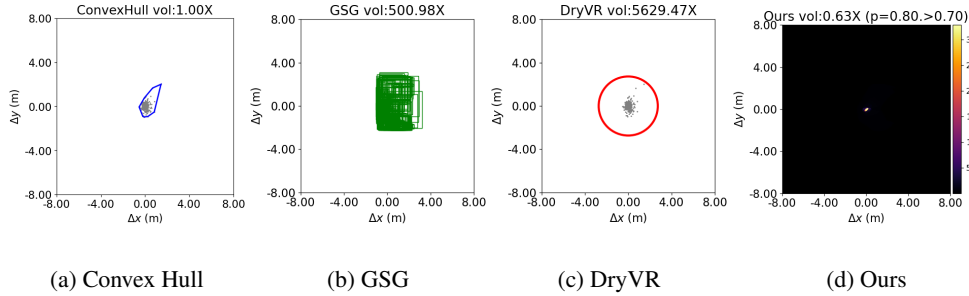


Figure 48: Comparison of reachable set computation among different tools (FACTEST car model, $t=49$). The gray dots are sampled points and blue / green / red / colored regions are reachability results

601

602 **G Comparison between histogram-based and Liouville-based approaches**

603 The advantage of learning density distribution by solving Liouville ODE is that it requires less
 604 training samples than histogram-based approaches, hence has the potential to generalize to high-
 605 dimension cases. To show its advantages in training efficiency and testing accuracy, we compare the
 606 histogram-based approach and Liouville-based approach’s density estimation for the following sys-
 607 tem, under different number of training samples. To make sure we can compare to the “groundtruth”
 608 density, we manually design the system such that the state density distribution at each time step has
 609 a closed form solution.

610 Consider a 1-d system: $\dot{x} = -x^2$ with initial states ranged from $[0, 1]$. Under uniformly distributed
 611 initialization, the system dynamics $x(t)$ and density distribution $\beta(x, t)$ (here $\beta(x, t)$ denotes the
 612 density at time t at location x) can be directly written out in the closed form:

$$\begin{cases} x(t) = 1/(C + t) \\ \beta(x, t) = 1/(1 - x \cdot t)^2 \end{cases} \quad (26)$$

613 where the parameter C can be derived from the initial condition $x(0) = x_0 = 1/C$.

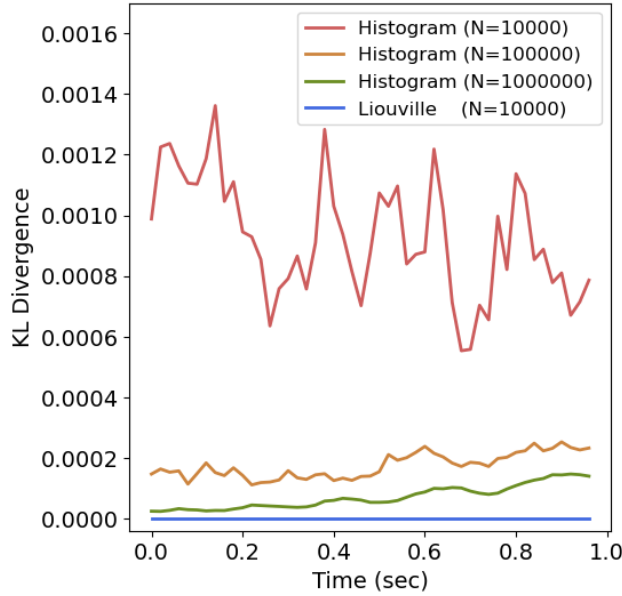


Figure 49: KL Divergence (comparing to groundtruth state density) for histogram-based approach and Liouville-based approach in different numbers of training samples. The groundtruth state density is computed analytically in closed form. We use 10000, 100000, 1000000 training samples for the histogram-based approach and use only 10000 training samples for the Liouville-based approach. The KL divergence is computed on a separate testing set (200 samples). Comparing to the histogram-based approach, the Liouville-based approach can achieve a smaller KL divergence while using 0.01X number of training samples.

614 We then use histogram-based and Liouville-based approach to estimate the state density for this
 615 system. We uniformly sample initial states and generate 1000000 trajectories using ODE45 solver.
 616 We use 10000, 100000 and 1000000 training samples for the histogram-based approach and use only
 617 10000 training samples for the Liouville-based approach, then we estimate the density on a separate
 618 testing set of trajectories using nearest neighbor interpolation. At each time step, we measure the
 619 estimation accuracy on the test set by computing the KL divergence to the groundtruth density.
 620 As shown in Fig. 49, histogram-based approach needs lots of samples to accurately approximate a
 621 good distribution (the KL divergence converges to zero at each time step as the number of samples
 622 increases), where our approach can learn the density distribution with the lowest KL divergence

using just 0.01X of the sampled trajectories. This shows the advantage of solving Liouville ODE to estimate the state density.

H Comparison with state-of-the-art worst-case reachability approaches

We compare our approach with three state-of-the-art worst-case reachability methods: Sherlock [75], Verisig [39] and ReachNN [40]. We use the official implementation of Verisig and ReachNN which focus on reachable set computation for neural network control systems (NNCS), and use the re-implementation of Sherlock from [6], which is for neural network verification.

To make a fair comparison, we set a timeout limit of six hours for all approaches. Among all the four datasets that our method has computed, Sherlock can solve for the reachable sets for the datasets “Double integrator”, “Ground robot navigation” and “FACTEST car tracking system”, and Verisig and ReachNN can only calculate for the “Double integrator” dataset - Verisig encounters numerical issue on this dataset at first due to the large initial set, and we have to divide the initial set to smaller sets and run the program multiple times in parallel to compute for the reachable sets. Similar in Sec. 4.3, we measure the reachable sets by computing the volume of the reachable sets relative to the volume of the convex hull of the sampled points.

We use different networks when doing reachability analysis, because all those methods have different requirements for the analyzed system:

- (a) The RPM used in our approach is doing reachability analysis for ReLU-based NNs. For the “Double Integrator” system, the controller is another ReLU-based NN that has a clip function at the output (to rectify the control output between $[-1, 1]$)
- (b) The Sherlock approach we used in [6] can only work with ReLU-based NN (not NNCS). Thus we used the same NN used in (a) and conducted the experiments. Since we only compute for the reachable set, we just collect the flow map estimation Φ_ω part of this NN (i.e. we did not need to use the density estimator part of the NN).
- (c) Verisig can only work with a Neural Network Controlled System (NNCS) with Sigmoid/Tanh-based NN controllers. Thus we re-trained a Tanh-based NN controller (using the same number of hidden layers and hidden units) to reproduce the output of the original controller in (a) and use this new controller to do reachability analysis. We verified that the L2 error between the Tanh-based NN controller and the original controller is less than 0.001 on the testing set, and we also inspected the trajectories generated using these two controllers and cannot find a substantial difference.
- (d) ReachNN can work with NNCS that has Sigmoid/Tanh/ReLU-based NN controllers. However, it cannot directly process the controller we had in (a) because the controller in (a) has a clip function at the output to rectify the control output between $[-1, 1]$. Therefore, we trained another ReLU-based NN controller that does not have that clip function to reproduce the output of the original controller in (a). We use this newly trained controller to do reachability analysis in ReachNN.

As shown in Fig. 50 ~ Fig. 53, in the “Double integrator” experiment, all of the three worst-case reachability analysis methods can only over-approximate the reachable sets of the system, with the reachable volume increasing over time. The approximation error for Verisig and ReachNN will severely accumulate, hence the corresponding reachable sets gradually occupy the whole figure (where the growths is 32.24X for Verisig and 67.96X for ReachNN respectively), whereas our approach estimated reachable sets have volume less than the convex hull volume, and can reflect the convergence of the majority of the system states owing to the ability to predict the state density. For higher dimension benchmarks like “Ground robot navigation” and “FACTEST car tracking system” (as shown in Fig. 54 ~ Fig. 57), only our approach and Sherlock are able to compute the reachable set under the timeout limit. Due to the high dimensionality, Sherlock’s estimated volume grows dramatically over time (16.51X for the “Ground robot navigation”, and 42.60X for the “FACTEST car model”), while our approach still gives more compact reachable sets. These observations illustrate the advantages of our approach in precisely estimating the system reachable sets as well as the state density distribution. One advantage of Sherlock over ours is that it can also solve for other benchmarks listed in Table. 2, where our approach cannot solve due to the numerical issues in

676 RPM. Another limitation is that our approach only solves for NN with ReLU activations, which is
 677 again a restriction inherited from RPM. We believe combining our learning framework with a more
 678 advanced exact reachability tool will resolve this issue in the future.

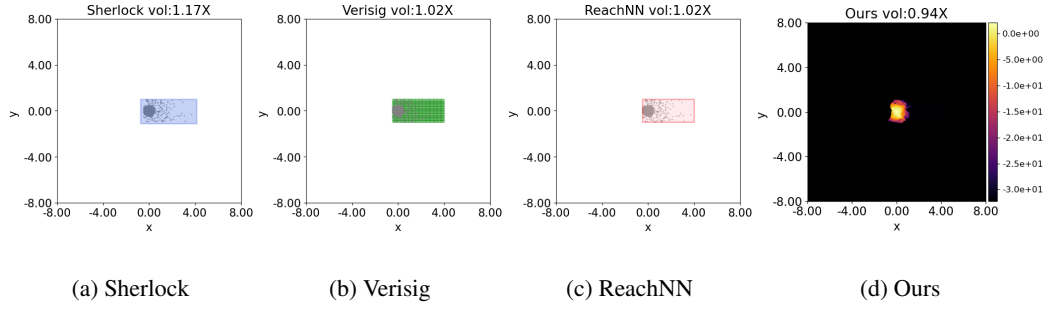


Figure 50: Comparison of reachable sets (Double integrator, $t=0$)

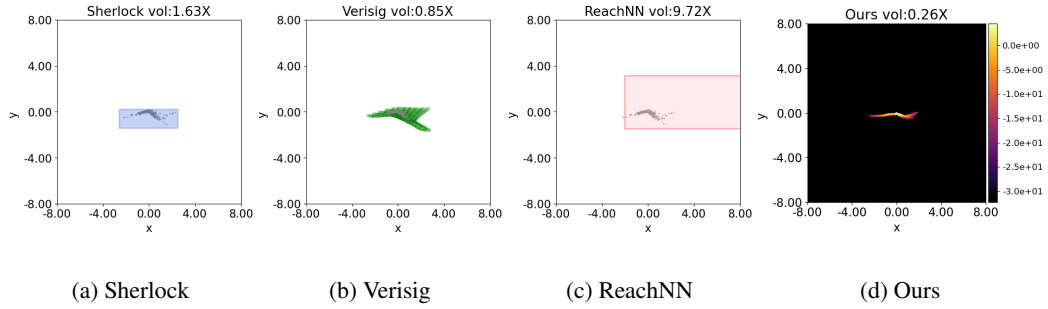


Figure 51: Comparison of reachable sets (Double integrator, $t=3$)

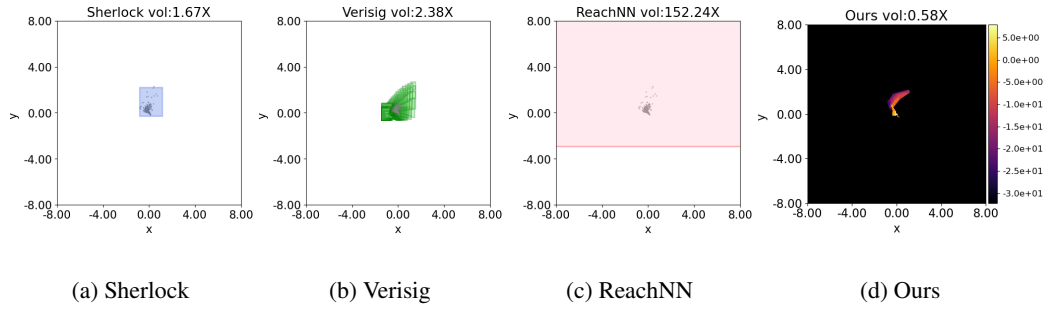


Figure 52: Comparison of reachable sets (Double integrator, $t=7$)

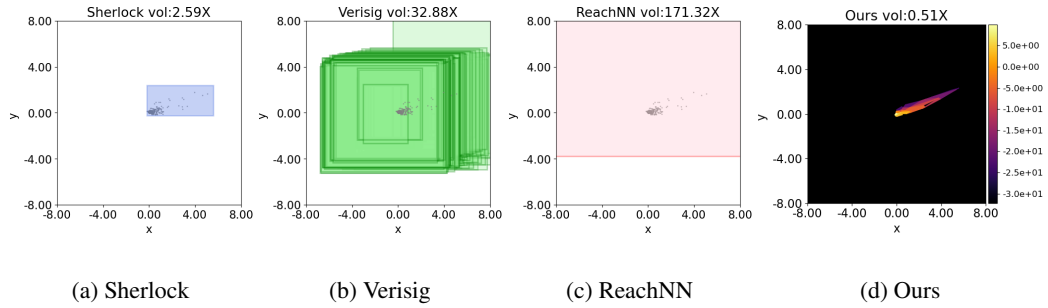
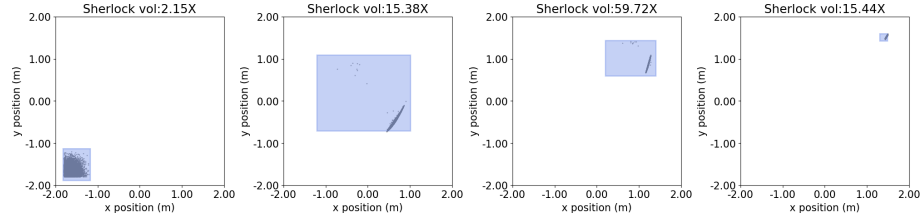
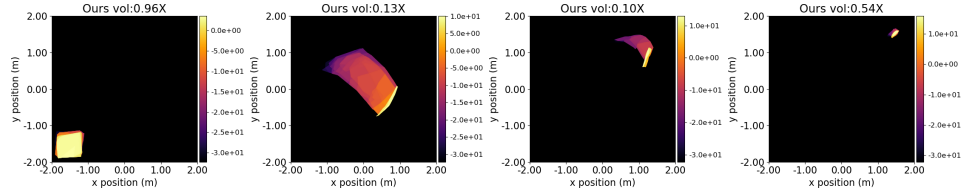


Figure 53: Comparison of reachable sets (Double integrator, $t=9$)



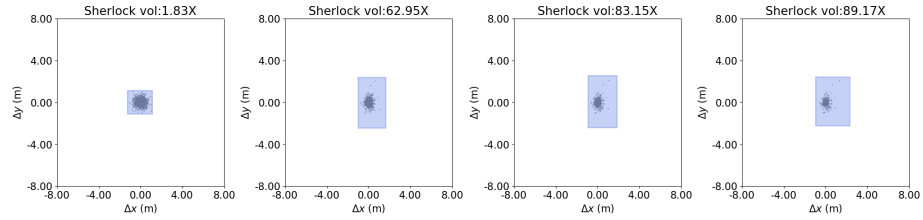
(a) Sherlock, $t=0$ (b) Sherlock, $t=20$ (c) Sherlock, $t=30$ (d) Sherlock, $t=40$

Figure 54: Sherlock results (Ground robot navigation, $t=0, 20, 30, 40$)



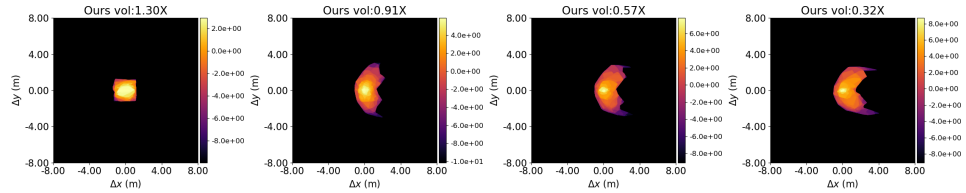
(a) Ours, $t=0$ (b) Ours, $t=20$ (c) Ours, $t=30$ (d) Ours, $t=40$

Figure 55: Our results (Ground robot navigation, $t=0, 20, 30, 40$)



(a) Sherlock, $t=0$ (b) Sherlock, $t=20$ (c) Sherlock, $t=30$ (d) Sherlock, $t=40$

Figure 56: Sherlock results (FACTEST car model, $t=0, 20, 30, 40$)



(a) Ours, $t=0$ (b) Ours, $t=20$ (c) Ours, $t=30$ (d) Ours, $t=40$

Figure 57: Our results (FACTEST car model, $t=0, 20, 30, 40$)

References

- [1] M. Chen and C. J. Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:333–358, 2018.
- [2] A. Devonport and M. Arcak. Data-driven reachable set computation using adaptive gaussian process classification and monte carlo methods. In *2020 American Control Conference (ACC)*, pages 2629–2634. IEEE, 2020.
- [3] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.
- [4] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.
- [5] S. Bansal and C. Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. *arXiv preprint arXiv:2011.02082*, 2020.
- [6] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.
- [7] A. Devonport and M. Arcak. Estimating reachable sets with scenario optimization. In *Learning for Dynamics and Control*, pages 75–84. PMLR, 2020.
- [8] L. Liebenwein, C. Baykal, I. Gilitschenski, S. Karaman, and D. Rus. Sampling-based approximation algorithms for reachability analysis with provable guarantees. *RSS*, 2018.
- [9] T. Lew and M. Pavone. Sampling-based reachability analysis: A random set theory approach with adversarial sampling. *arXiv preprint arXiv:2008.10180*, 2020.
- [10] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *International Workshop on Hybrid Systems: Computation and Control*, pages 202–214. Springer, 2000.
- [11] A. Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.
- [12] P. S. Duggirala and M. Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer, 2016.
- [13] P.-J. Meyer, A. Devonport, and M. Arcak. Tira: Toolbox for interval reachability analysis. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 224–229, 2019.
- [14] S. Bak. Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, pages 40–43, 2014.
- [15] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048. IEEE, 2008.
- [16] J. Cyranka, M. A. Islam, G. Byrne, P. Jones, S. A. Smolka, and R. Grosu. Lagrangian reachability. In *International Conference on Computer Aided Verification*, pages 379–400. Springer, 2017.
- [17] M. Ehrendorfer. The liouville equation and prediction of forecast skill. In *Predictability and Nonlinear Modelling in Natural Sciences and Economics*, pages 29–44. Springer, 1994.

- [18] T. Nakamura-Zimmerer, D. Venturi, Q. Gong, and W. Kang. Density propagation with characteristics-based deep learning. *arXiv preprint arXiv:1911.09311*, 2019.
- [19] Y. Chen, M. Ahmadi, and A. D. Ames. Optimal safe controller synthesis: A density function approach. In *2020 American Control Conference (ACC)*, pages 5407–5412. IEEE, 2020.
- [20] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [21] J. A. Vincent and M. Schwager. Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components. *arXiv preprint arXiv:2011.11609*, 2020.
- [22] International competition on verifying continuous and hybrid systems. <https://cps-vo.org/group/ARCH/FriendlyCompetition>. Accessed: 2021-06-18.
- [23] G. Agha and K. Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1):1–39, 2018.
- [24] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.
- [25] B. Xue, M. Zhang, A. Easwaran, and Q. Li. 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.
- [26] A. Berndt, A. Alanwar, K. H. Johansson, and H. Sandberg. Data-driven set-based estimation using matrix zonotopes with set containment guarantees. *arXiv preprint arXiv:2101.10784*, 2021.
- [27] R. E. Allen, A. A. Clark, J. A. Starek, and M. Pavone. A machine learning approach for real-time reachability analysis. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, pages 2202–2208. IEEE, 2014.
- [28] M. Rasmussen, J. Rieger, and K. N. Webster. Approximation of reachable sets using optimal control and support vector machines. *Journal of Computational and Applied Mathematics*, 311:68–83, 2017.
- [29] A. J. Thorpe, K. R. Ortiz, and M. M. Oishi. Data-driven stochastic reachability using hilbert space embeddings. *arXiv preprint arXiv:2010.08036*, 2020.
- [30] A. Chakrabarty, C. Danielson, S. Di Cairano, and A. Raghunathan. Active learning for estimating reachable sets for systems with unknown dynamics. *IEEE Transactions on Cybernetics*, 2020.
- [31] D. Fridovich-Keil, A. Bajcsy, J. F. Fisac, S. L. Herbert, S. Wang, A. D. Dragan, and C. J. Tomlin. Confidence-aware motion prediction for real-time collision avoidance1. *The International Journal of Robotics Research*, 39(2-3):250–265, 2020.
- [32] A. Majumdar, R. Vasudevan, M. M. Tobenkin, and R. Tedrake. Convex optimization of nonlinear feedback controllers via occupation measures. *The International Journal of Robotics Research*, 33(9):1209–1230, 2014.
- [33] A. Abate. *Probabilistic reachability for stochastic hybrid systems: theory, computations, and applications*. University of California, Berkeley, 2007.
- [34] A. R. R. Matavalam, U. Vaidya, and V. Ajjarapu. Data-driven approach for uncertainty propagation and reachability analysis in dynamical systems. In *2020 American Control Conference (ACC)*, pages 3393–3398. IEEE, 2020.
- [35] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

- [36] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.
- [37] W. Xiang, H.-D. Tran, and T. T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5777–5783, 2018.
- [38] X. Yang, H.-D. Tran, W. Xiang, and T. Johnson. Reachability analysis for feed-forward neural networks using face lattices. *arXiv preprint arXiv:2003.01226*, 2020.
- [39] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.
- [40] J. Fan, C. Huang, X. Chen, W. Li, and Q. Zhu. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 537–542. Springer, 2020.
- [41] H. Hu, M. Fazlyab, M. Morari, and G. J. Pappas. Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 5929–5934. IEEE, 2020.
- [42] M. Everett, G. Habibi, and J. P. How. Efficient reachability analysis of closed-loop systems with neural network controllers. *arXiv preprint arXiv:2101.01815*, 2021.
- [43] D. A. Spencer and R. D. Braun. Mars pathfinder atmospheric entry-trajectory design and dispersion analysis. *Journal of Spacecraft and Rockets*, 33(5):670–676, 1996.
- [44] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.
- [45] C. Pantano and B. Shotorban. Least-squares dynamic approximation method for evolution of uncertainty in initial conditions of dynamical systems. *Physical Review E*, 76(6):066705, 2007.
- [46] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.
- [47] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [48] T. Uchiyama and N. Sonehara. Solving inverse problems in nonlinear pdes by recurrent neural networks. In *IEEE International Conference on Neural Networks*, pages 99–102. IEEE, 1993.
- [49] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- [50] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.
- [51] E. Weinan and B. Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.
- [52] J. He, L. Li, J. Xu, and C. Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.
- [53] J. Han, A. Jentzen, et al. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. *arXiv preprint arXiv:2008.13333*, 2020.
- [54] Y. Khoo, J. Lu, and L. Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.
- [55] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider. A theoretical analysis of deep neural networks and parametric pdes. *Constructive Approximation*, pages 1–53, 2021.

- [56] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.
- [57] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.
- [58] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [59] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.
- [60] L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.
- [61] A. Koryagin, R. Khudorozkov, and S. Tsimfer. Pydens: A python framework for solving differential equations with neural networks. *arXiv preprint arXiv:1909.11544*, 2019.
- [62] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.
- [63] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. Di Giovanni. Neurodiffq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46):1931, 2020.
- [64] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *icml*, 2010.
- [65] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?—arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.
- [66] S. A. Orszag and L. Bissonnette. Dynamical properties of truncated wiener-hermite expansions. *The Physics of Fluids*, 10(12):2603–2613, 1967.
- [67] Y.-C. Chang, N. Roohi, and S. Gao. Neural lyapunov control. *arXiv preprint arXiv:2005.00611*, 2020.
- [68] C. Fan, K. Miller, and S. Mitra. Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In *International Conference on Computer Aided Verification*, pages 629–652. Springer, 2020.
- [69] P. Heidlauf, A. Collins, M. Bolender, and S. Bak. Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In *ARCH@ ADHS*, pages 208–217, 2018.
- [70] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 686–701, 2019.
- [71] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.
- [72] C. Donner and M. Opper. Efficient bayesian inference of sigmoidal gaussian cox processes. *10.14279/depositonce-8398*, 2018.
- [73] M. Everett, G. Habibi, and J. P. How. Robustness analysis of neural networks via efficient partitioning with applications in control systems. *IEEE Control Systems Letters*, 2020.
- [74] P. Du, Z. Huang, T. Liu, T. Ji, K. Xu, Q. Gao, H. Sibai, K. Driggs-Campbell, and S. Mitra. Online monitoring for safe pedestrian-vehicle interactions. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

- 864 [75] S. Dutta, S. Jha, S. Sanakaranarayanan, and A. Tiwari. Output range analysis for deep neural
865 networks. *arXiv preprint arXiv:1709.09130*, 2017.
- 866 [76] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with
867 a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):
868 861–867, 1993.
- 869 [77] N. Srebro, K. Sridharan, and A. Tewari. Smoothness, low noise and fast rates. *Advances in*
870 *neural information processing systems*, 23, 2010.
- 871 [78] N. M. Boffi, S. Tu, N. Matni, J.-J. E. Slotine, and V. Sindhvani. Learning stability certificates
872 from data. *arXiv preprint arXiv:2008.05952*, 2020.
- 873 [79] E. A. Coddington and N. Levinson. *Theory of ordinary differential equations*. Tata McGraw-
874 Hill Education, 1955.
- 875 [80] B. Schürmann and M. Althoff. Optimal control of sets of solutions to formally guarantee
876 constraints of disturbed linear systems. In *2017 American Control Conference (ACC)*, pages
877 2522–2529. IEEE, 2017.
- 878 [81] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra.
879 Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.