Scalable Synthesis of Verified Controllers in Deep Reinforcement Learning

No Author Given

No Institute Given

Abstract. There has been significant recent interest in devising verification techniques for learning-enabled controllers (LECs) that manage safety-critical systems. Given the opacity and lack of interpretability of the neural policies that govern the behavior of such controllers, many existing approaches enforce safety properties through the use of shields, a dynamic monitoring and repair mechanism that ensures a LEC does not emit actions that would violate desired safety conditions. These methods, however, have shown to have significant scalability limitations because verification costs grow as problem dimensionality and objective complexity increases. In this paper, we propose a new automated verification pipeline capable of synthesizing high-quality safety shields even when the problem domain involves hundreds of dimensions, or when the desired objective involves stochastic perturbations, liveness considerations, and other complex non-functional properties. Our key insight involves separating safety verification from neural controller, using pre-computed verified safety shields to constrain neural controller training which does not only focus on safety. Experimental results over a range of realistic high-dimensional deep RL benchmarks demonstrate the effectiveness of our approach.

Keywords: Safe Reinforcement Learning, Robust Control, Synthesis, Shields, Verification, Probabilistic Reachability Analysis

1 Introduction

Deep Reinforcement Learning (DRL) has proven to be a powerful tool to implement autonomous controllers for various kinds of cyber-physical systems (CPS) [21,14,24]. Since these learning-enabled controllers are intended to operate in safety-critical environments, there has been significant recent interest in developing verification methods that ensure their behavior conforms to desired safety properties [15,29,2,31,3,20]. While these different approaches all provide strong guarantees on controller safety, scaling their techniques, both with respect to problem dimensionality as well as objective complexity, has proven to be challenging. Techniques that attempt to statically verify a neural controller always preserves safety guarantees may reject high-quality controllers that may only infrequently violate safety [15,29]. Alternatively, techniques that dynamically monitor controller actions, triggering a safety shield when these actions may

lead to an unsafe state, require the shield's behavior to align closely with the neural controller, which is typically not only trained with safety in mind [3,31]. Balancing these competing goals of ensuring safety on the one hand and maximizing objective reward on the other poses its own set of challenges that can compromise verifiability, performance, and safety.

This paper considers a novel pipeline that addresses these shortcomings. Similar to other shielding-based approaches [2,3,31], our work does not statically verify a neural controller directly. Instead, it generates a *family* of linear controllers from known system dynamics and the problem specification; this family collectively serves as a shield that can dynamically enforce the safe operation of the system. Controllers are synthesized using linear-quadratic regulators (LQR) that yield different instantiations based on a cost function expressed in terms of different parameter values associated with states and control inputs. Our verification strategy searches for a combination of different controller instances that satisfy a desired safety property in the aggregate; this search is driven by a novel probabilistic reachability analysis. Different controllers in this aggregation are optimized for different time steps in possible system trajectories. Not all combinations of different controllers can be verified to be safe. Nonetheless, even combinations that are possibly unsafe can be used to provide strong probabilistic guarantees on their safety efficacy, which the verifier can leverage as part of its subsequent search. Because our technique considers safety verification independently of neural controller behavior and neural network internal structure, it is highly scalable with respect to objective complexity and problem dimensionality. In our experiments, for example, we demonstrate successful verification of CPS benchmarks with over 800 dimensions, a scale that is significantly higher than reported with existing approaches.

Because these linear controllers are generated based only on safety considerations, they are not intended to serve as the primary mechanism for governing the actual operation of the system, which must also take into account other performance-related objectives. To ensure the safety of a neural controller, tailored with optimal execution in mind, with a synthesized linear controller family, developed with safety as its only goal, our pipeline additionally integrates the controller family as part of the DRL training process. This integration biases the training process of the neural controller towards verified safety characteristics. However, because neural controller objectives do not impose safety requirements on the training procedure, our technique allows neural policies to be learnt with complex objectives (e.g., achieving different goals in a specific temporal order, ensuring various kinds of liveness properties, etc.). A well-trained neural controller is expected to be generally well-behaved, but is not guaranteed to be safe. However, safety can be enforced by operating the neural controller in conjunction with the verified linear controller family. With scalability considerations in mind, our work demonstrates a fully automated safety verification pipeline for DRL-based learning-enabled controllers trained with complex objectives.

The remainder of the paper is organized as follows. We provide additional motivation and an informal overview of our technique in the next section. Section 3
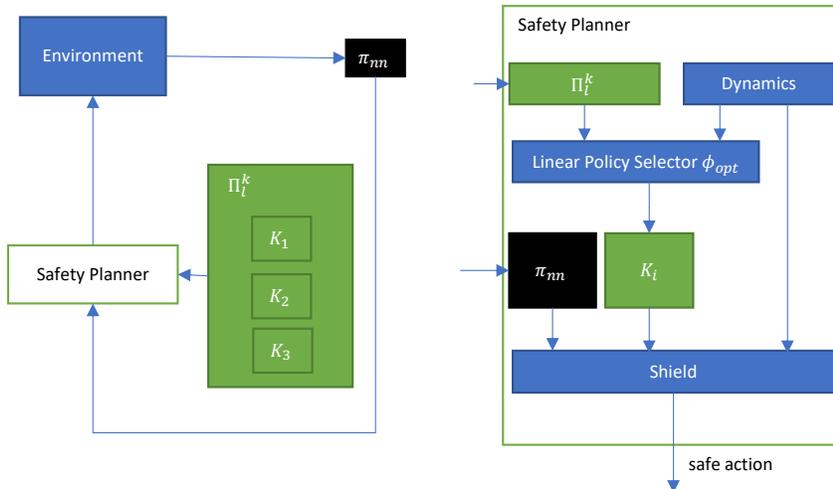
Fig. 1. Approach Overview

provides necessary background material used in our approach. Section 4 formalizes our approach and presents our verification algorithm. Section 5 presents a detailed experimental study on a range of complex CPS benchmarks; these benchmarks exhibit high dimensionality, have non-trivial noise terms leading to stochastic non-deterministic behavior, and are trained with objective functions that reflect important non-functional properties such as liveness property. Related work and conclusions are given in Section 6.

2   Motivating Example and Overview

Our overall approach, shown in Fig. 1, ensures a reinforcement learning system's safety by inserting a safety planner in the loop that governs the interaction between a neural controller and its environment. The planner takes as input the neural network controller and an LQR linear controller family as input. A selector ϕ picks up a linear controller from the linear controller family every k steps of a simulated trajectory. Given a linear controller picked by ϕ, the distribution of an initial state set, and characterization of (bounded) noise, the safety planner computes all states reachable by the controller in the next k steps of the trajectory, as well as a lower bound of the probability density of landing in these states. These states comprise a bounded reachable set for the selected controller. Our verification algorithm aims to generate an optimal selector ϕ_{opt} such that it maximizes the overlap between the reachable set of selected linear

policies and the safe state space. A safety shield takes the neural controller, the selected linear controller, and the k-step reachable set of the dynamics as input. It monitors a neural network's action and ensures that the proposed action yields a state within the reachable set of the linear policy, applying the action proposed by the safe linear controller when it does not. Because the selected linear controller's reachable set overlaps with the safe state space, we can provide a high probabilistic, and oftentimes exact, guarantee that the neural controller will not exceed the linear controller's reachable set even in continuous action space.

To illustrate our approach, we consider the application of our technique on a simple but non-trivial, problem - autonomous control of an inverted swinging pendulum, with the safety requirement being that the pendulum remains upright.

2.1 Inverted Swinging Pendulum

The inverted pendulum model we consider is widely used for evaluating reinforcement learning and continuous control tasks. The system is governed by the ODE shown in Fig 2. We assume that the inverted pendulum starts from a uniform distribution of initial states $\mathcal{S}_0$,

$$\{(\theta, \omega) \mid -\frac{\pi}{10} \leq \theta \leq \frac{\pi}{10} \wedge -\frac{\pi}{10} \leq \omega \leq \frac{\pi}{10}\}$$

The global safety property we wish to enforce is that the pendulum never falls, and that the pendulum's velocity should not be too large, residing with a pre-determined interval. We define a set of unsafe states of the pendulum $\mathcal{S}_u$ as

$$\{(\theta, \omega) \mid |\theta| > \frac{\pi}{2} \vee |\omega| > \frac{\pi}{2}|\}$$

The performance objective of the controller, however, is a simple liveness property that aims to keep the pendulum swinging. We can encode the desired liveness and safety property into a single reward function used to train a neural controller:

$$R = \sum_{t=1}^{M} \gamma^t \left(\underbrace{\left|\dot{\theta}_t\right|}_{liveness} + \underbrace{\min\left((\frac{\pi}{2} - |\theta_t|), 0\right) + \min\left((\frac{\pi}{2} - |\omega_t|), 0\right)}_{safety} \right) \qquad (1)$$

$\gamma \in (0, 1]$ is a discount factor generally used in the specification of a reinforcement learning reward function, and t is a time step. $|\dot{\theta}_t|$ defines the position change of the pendulum. Because we want the pendulum to keep swinging, we seek to maximize the cumulative total of this term. $(\frac{\pi}{2} - |\theta_t|)$ and $(\frac{\pi}{2} - |\omega_t|)$ are the L_1 distances to the unsafe region S_u. When the system state is safe, $\min\left((\frac{\pi}{2} - |\theta_t|), 0\right)$ and $\min\left((\frac{\pi}{2} - |\omega_t|), 0\right)$ will be 0; if the safety property is violated, the trajectory will be penalized with a negative reward during neural network training. A deep reinforcement learning algorithm aims to maximize this reward of the resulting controller π_{nn}.
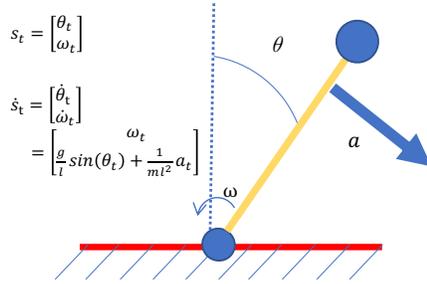
$$s_t = \begin{bmatrix} \theta_t \\ \omega_t \end{bmatrix}$$

$$\dot{s}_t = \begin{bmatrix} \dot{\theta}_t \\ \dot{\omega}_t \end{bmatrix}$$

$$= \begin{bmatrix} \omega_t \\ \frac{g}{l}\sin(\theta_t) + \frac{1}{ml^2}a_t \end{bmatrix}$$

Fig. 2. The pendulum has mass m and length l. A system state is $s = [\theta, \omega]$, where θ is the its angle and ω is its angular velocity. A continuous control action a attempts to maintain the pendulum in an upright position.

Replacing $sin(\theta)$ in $\dot{\omega}_t = \frac{g}{l}\sin(\theta_t) + \frac{a_t}{ml^2}$ with θ, yields a linearized (approximated) pendulum model. For simplicity, the linearized $\dot{s}$ can be written as $\dot{s}_t = As_t + Ba_t$, where

$$A = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}$$

For every Δt time interval, we update the state of the system once, giving a transition function of the form $s_{t+1} = s_t + \Delta t \cdot \dot{s}_t$. We also incorporate stochasticity with a bounded noise term $w \in [\epsilon_l, \epsilon_u]$. Then, the transition function becomes $s_{t+1} = s_t + \Delta t \cdot \dot{s}_t + w$. In fact, although we consider a linearized model here, as long as w is sufficiently large, all possible transition produced by a corresponding non-linear model can also be considered. We set $\Delta t = 0.01$, $\epsilon_l = -1.5 \times 10^{-2}$ and $\epsilon_u = 1.5 \times 10^{-2}$ for the pendulum system that we want to verify.

2.2   Challenge of Distilling Verified Policies

Reward	Verified
Safety rew. only	35
Liveness + safety rew.	0

Table 1. The number of controllers that were verified from a total of 50 distilled under different rewards.

Previous work [3,31] in shield-based DRL verification distill linear controllers by imitating the neural network, with safety being a primary goal of the learnt controller. In practice, however, reinforcement learning tasks need to consider different non-functional properties, in addition to safety. In our example, we want to simultaneously maximize the velocity of the pendulum while also preserving safety. In Table 1, we show that a safety controller distilled from the neural network trained with different rewards impacts their verifiability. We set two different reward functions for the pendulum task in this experiment. In the first setting, we only consider the safety part of the equation 1; the other considers both liveness and safety. For each reward function, we distill 50 different linear controllers and verify them with the verification tool provided in [31]. The Verified column shows the number of controllers among the 50 distilled controllers considered that were verified to be safe. The experiment results show that when both properties are considered, safety verification becomes significantly more challenging, even for problems as simple as our running example.

2.3   Verifying a Linear Controller Family

First, we build a linear controller family with LQR. The linear controller family's goal is to provide a safe controller for the pendulum model. A natural way to help ensure safety is to define controllers with strong stability constraints. We use LQR to generate a batch of linear policies with different Q and R values that penalize changes to state variables and control signals. For our pendulum example, we set the number of linear controllers $|\Pi_l^k| = 3$. Q and R are diagonal matrices. The value of Q's diagonal elements are randomly sampled from $(0, 20]$ and the value of R's diagonal elements are sampled from $(0, 10]$. One instance of Π_l^k is:

$$K_1 = \begin{bmatrix} -22.1 \\ -10.4 \end{bmatrix}, K_2 = \begin{bmatrix} -42.4 \\ -22.4 \end{bmatrix}, K_3 = \begin{bmatrix} -20.1 \\ -5.3 \end{bmatrix}$$

Given a linear controller family Π_l^k, we want to find a combination of these policies that collectively provide a safety guarantee for the pendulum. We synthesize a selector ϕ to build such a combination. First, we fix that the selection interval for one linear controller to be k steps. That is, the selector ϕ identifies one linear controller every k steps. We consider the length of an execution to be M. Thus, we need to select linear policies $\lceil \frac{M}{k} \rceil$ times. If we set $M = 200$ and $k = 100$ in this example, we would need to make 2 selections for a given simulation. Given an instance of $\Pi_l^k = (K_1, K_2, K_3)$, $\phi(\Pi_l^k, t)$ returns a linear controller based on step t, e.g., it may be that when $t = 99, \phi(\Pi_l^k, t) = K_1$ and when $t = 100, \phi(\Pi_l^k, t) = K_2$.
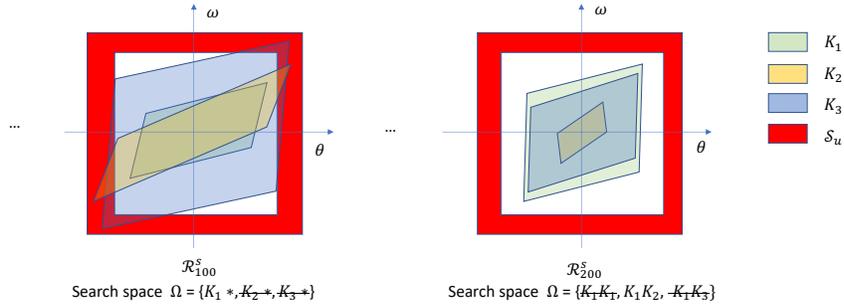
Fig. 3. An illustration indicating how different linear controllers capture different reachable states. At step 100 the reachable sets generated by controllers K_2 and K_3 violate the safety property; at step 200, using the reachable set generated by K_1 for the first 100 steps, we find the reachable set generated by K_2 at step 200 to be a subset of the reachable sets generated by K_1 and K_3, and is thus the preferred controller to use.

The mission of the selector ϕ is to provide a safe controller with high probability. To synthesize the optimal selector ϕ_{opt} that maximizes a safety probability,

we need to analyze the reachability of different combinations of policies. The goal is to provide a lower bound for the percentage of safe states $\hat{p}_t$ in the reachable set $\mathcal{R}_t^s$ at step t. Suppose the reachable set $\mathcal{R}_t^s$ has no overlap with S_u; in this case, $\hat{p}_t$ is 1. The s superscript in $\mathcal{R}_t^s$ signifies the reachable set under stochasticity.[1] The optimal selector is then defined to be:

$$\phi_{opt} = \operatorname*{argmax}_{\phi} \sum_{t=1}^{M} \hat{p}_t \left(\Pi_l^k, \phi \right)$$

The parameterized $\hat{p}_t \left(\Pi_l^k, \phi \right)$ defines the lower bound on the probability of a safe transition at step t when $\mathcal{R}_t^s$ is generated with the controller selected by ϕ from the linear controller family Π_l^k. Because the number of possible selectors is $\left| \Pi_l^k \right|^{\lceil \frac{M}{k} \rceil}$, it is intractable to examine all feasible instances of ϕ as $\lceil \frac{M}{k} \rceil$ and $|\Pi_l^k|$ increase. Thus, we also provide several strategies to reduce the possible search space. For example, in Fig 3, all instances containing K_2 and K_3 are pruned from the search space Ω at step 100, because their reachable sets overlap with S_u, while the reachable set of K_1 does not; $K_1 K_1$, $K_1 K_3$ are pruned from Ω at step 200 because the reachable set of K_2 is a subset of the reachable sets of K_1 and K_3 (more details are provided in Section 4.1). With the instance of Π_l^k as given above, $\phi_{opt} = (K_1, K_2)$, and $\sum_{t=1}^{M} \hat{p}_t \left(\Pi_l^k, \phi \right) = 200$. This instantiation of ϕ_{opt} ensures the safety of the system for $M = 200$ steps, and we say the selector and its corresponding linear controller family are verified.

2.4 Safety Planner with Shield

The optimal selector ϕ_{opt} ensures the stability of the pendulum with respect to the safety property, but does not consider other desirable properties an autonomous controller should exhibit, such as liveness. To combine both optimal safe controller with a DRL-enabled neural controller, our verification methodology also includes a safety planner that integrates these different objectives.

Consider two identical systems that have different noise terms. The first system E' is used for verification. It has noise term $\hat{w}_1 \in [\epsilon_l^1, \epsilon_u^1]$. The second system E is used to model an application. It has noise term $\hat{w}_2 \in [\epsilon_l^2, \epsilon_u^2]$. To be useful, the noise considered by verification environment E' should be greater than E, that is, $\epsilon_l^2 > \epsilon_l^1$ and $\epsilon_u^2 < \epsilon_u^1$.

Assume that at a state transition at step t, the system state is s_t, the action provided by the neural network is a_t, and the action provided by the verified linear controller family is a_t'. Applying actions a_t and a_t' without considering the noise term results in state $\hat{s}_{t+1}$ and state $\hat{s}_{t+1}'$ respectively. To take noise into consideration, denote all the reachable states under E and E' as $\texttt{Region}(\hat{s}_{t+1}, \hat{w}_2)$ and $\texttt{Region}(\hat{s}_{t+1}', \hat{w}_1)$ respectively. As shown in (a) of Fig. 4, when $\texttt{Region}(\hat{s}_{t+1}, \hat{w}_2) \subseteq \texttt{Region}(\hat{s}_{t+1}', \hat{w}_1)$, the action taken by the neural network is under the safety guarantees provided by the optimal safe controller. That

---

[1] If there is overlap between $\mathcal{R}_t^s$ and S_u, we compute $\hat{p}_t$ using the approach described in Appendix Appendix B.4.

(a) No intervention   (b) No intervention   (c)Intervention required
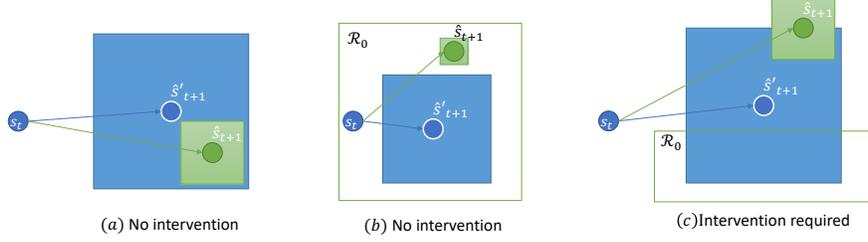
Fig. 4. Three possible cases for a transition. The part shaded green denotes the reachable set of π_{nn} at step $t+1$ and the part shaded blue denotes the reachable set of $\phi(\Pi_l^k, t)$. $\mathcal{R}_0$ is the initial state space of the system. $(a)(b)$ do not require any intervention, while (c) and do.

is, if we take over the neural network after step $t+1$ with our optimal safe controller, we can guarantee that the system will not violate safety properties. In addition, any state in the initial state space $\mathcal{R}_0$ are guaranteed to be safe for at least M steps. Thus, in (b) of Fig. 4, although the reachable set of neural controller resides outside of $\texttt{Region}(\hat{s}'_{t+1}, \hat{w}_1)$, no intervention is required. When $\texttt{Region}(\hat{s}_{t+1}, \hat{w}_2) \nsubseteq \texttt{Region}(\hat{s}'_{t+1}, \hat{w}_1) \cup \mathcal{R}_0$, s_t resides outside the identified safety region. In this case, we need to take action a'_t instead to ensure the s_t will remain safe. In the pendulum example, we set $\hat{w}_1 \in [-1.5 \times 10^{-2}, 1.5 \times 10^{-2}]$ and the $\hat{w}_2 \in [-1.5 \times 10^{-3}, 1.5 \times 10^{-3}]$.

3 Preliminaries

3.1 Markov Decision Process and Reinforcement Learning

A Markov decision process (MDP) is a tuple, $(S, A, R, P, \mathcal{D}_{s_0}, S_u)$, where S is the state space, A is the action space, $R : S \times A \times S \to \mathbb{R}$ is a reward function, $P : S \times A \times S \to [0, 1]$ defines a transition probability function such that $P(s'|s, a)$ is the probability the process transitions to state s' given that the previous state was s and the agent took action a in s; $\mathcal{D}_{s_0} : S \to [0, 1]$ is the initial state distribution that we assume is uniform; and, S_u defines a set of constraints that specify the unsafe state space. Any state in the unsafe state space is prohibited. A controller $\pi : S \to \mathcal{P}(A)$ is a map from states to probability distributions over actions, with $\pi(a|s)$ denoting the probability of selecting action a in state s. The goal of reinforcement learning is to learn a controller π which maximizes a performance measure.

$$J(\pi) \doteq \operatorname*{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, a_t, s_{t+1}\right) \right]$$

Here, $\gamma \in [0, 1)$ is the discount factor, τ denotes a trajectory $(\tau = (s_0, a_0, s_1, \ldots))$, and $\tau \sim \pi$ is shorthand to indicate that the distribution over trajectories depends on $\pi : s_0 \sim \mathcal{D}_{s_0}, a_t \sim \pi\left(\cdot \mid s_t\right), s_{t+1} \sim P\left(\cdot \mid s_t, a_t\right)$.

3.2 Properties and Rewards

Intuitively, an application's safety property requires that the reachable set of possible states produced by the controller's action be in a safe region. For any state of a benchmark, n is the number of state dimension, $n = |S|$. Let $s \in \mathbb{R}^n, L < s < U$, where $L \in \mathbb{R}^n$ and $U \in \mathbb{R}^n$. We encode the safety property as R_{safe}. Now, suppose the system arrives a state $x \in \mathbb{R}^n$; then,

$$R_{safe}(x) = \sum_{i=0}^{n-1} \left(\texttt{min} \left(x_i - L, 0 \right) + \texttt{min} \left(U - x_i, 0 \right) \right)$$

Our desired liveness property requires system states to keep varying. There are some dimensions of a state that describe these state changes, e.g., the second dimension of the pendulum's state is its angular velocity, which is defined in terms of the rate of change of the pendulum's angular rotation (the first dimension in the state). We identify those dimensions $\texttt{Dim}_{live}$ that describe the change of state and encode them as R_{live}. Given a function

$$\texttt{GreaterThan}(a, b) = \begin{cases} 1, a > b \\ 0, else \end{cases}$$

we can define

$$R_{live}(x) = \sum_{i \in \texttt{Dim}_{live}} \texttt{GreaterThan}(|x_i|, T_i)$$

$T \in \mathbb{R}^n$ is a vector of thresholds. If the absolute value of one state dimension is greater than this value, we give a positive reward. Maximizing this reward means that we want as many dimensions as possible to hit the threshold. The definition of liveness reward is slightly different with we saw in the motivation example, because we want to balance the reward contribution from different dimensions.

3.3 Controller Types

We consider two kinds of controllers. We expect the performance controller π_{nn} to be a neural network controller trained by a state-of-art reinforcement learning algorithms. Depending on the algorithm used, π_{nn} can either be deterministic or stochastic. Given a state s of the system, $\pi_{nn}(s_t)$ outputs an action a_t at state s_t. For a deterministic controller, a_t is deterministic. The action a_t produced by a stochastic controller is an action sampled from the distribution predicted by $\pi_{nn}(s_t)$.

A deterministic linear controller family Π_l^k is a set of linear policies $K_i \in \mathbb{R}^{n \times n}$. The superscript k in Π_l^k is a temporal unit. In every k-unit-time interval $[\lfloor \frac{t}{k} \rfloor k, \lfloor \frac{t}{k} \rfloor k + k)$, a selector $\phi(\Pi_l^k, t)$ will choose a linear controller in Π_l^k to predict actions in this time interval.

3.4   Stochastic Transition System

The transition probability function P is modeled as:

$$\begin{cases} \dot{s}_t = tr(s_t, a_t) \\ s_{t+1} = s_t + \dot{s}_t \Delta t + w \end{cases}$$

where $s_t \in \mathbb{R}^n$ is the state vector and $a_t \in \mathbb{R}^{|A|}$ is the action. The function tr computes $\dot{s}_t$, P's derivative at time t when taking action a_t. Stochasticity is introduced by adding a noise (or error) term w to each transition. This noise is subject to a certain distribution D_w. In this paper, we consider bounded noise, that is, $\exists \epsilon_l, \epsilon_u \in \mathbb{R}^n, w \in [\epsilon_l, \epsilon_u]$.

The initial state distribution of a system is bounded by lower bound $\mathcal{L}s_0$ and upper bound $\mathcal{U}_{s_0}$, where s_0 is the initial state. If we now have a linear controller chosen by $\phi\left(\Pi_l^k, 0\right)$, the noise-free state $\hat{s}_1$ at step 1 is

$$\begin{aligned} \hat{s}_1 &= s_0 + \Delta t \cdot \left(As_0 + B \cdot \phi\left(\Pi_l^k, 0\right) s_0\right) \\ &= \left(I + \Delta t \cdot \left(A + B \cdot \phi\left(\Pi_l^k, 0\right)\right)\right) s_0 \\ &= \mathcal{T}_0 s_0 \end{aligned}$$

The state at step 1 after adding the noise term, $s_1 = \hat{s}_1 + w$ where $\hat{s}_1$ is the result after a linear transformation on state s_0. For different steps, $\mathcal{T}_i = I + \Delta t \cdot \left(A + B \cdot \phi\left(\Pi_l^k, i\right)\right)$ is a constant matrix. Thus for step t where $t > 0$, we have $s_{t+1} = \mathcal{T}_t s_t + w$ and thus by induction,

$$\begin{aligned} s_t &= \mathcal{T}_{t-1}\left(\cdots\left(\mathcal{T}_1\left(\mathcal{T}_0 s_0 + w\right) + w\right)\cdots + w\right) + w \\ &= \prod_{i=0}^{t-1} \mathcal{T}_i s_0 + (I + \sum_{i=1}^{t-1}\prod_{j=i}^{t-1} \mathcal{T}_j)w \end{aligned} \tag{2}$$

We denote the stochastic reachable set of a stochastic system at step t as $\mathcal{R}_t^s$, $\mathcal{T}_{s_t} = \prod_{i=0}^{t-1} \mathcal{T}_i$, and $\mathcal{T}_{w_t} = I + \sum_{i=1}^{t-1}\prod_{j=i}^{t-1} \mathcal{T}_j$. We store $\mathcal{R}_t^s$ as a tuple $(\mathcal{T}_{s_t}, \mathcal{T}_{w_t})$.

3.5   Safety Probability of Reachable Sets

We hope to calculate the safety probability of reachable sets on continuous space, thus need to find the probability density function (PDF) for a reachable set $\mathcal{R}_t^s$. Suppose $f_t(s)$ is the PDF of the distribution over the landing property for state s_t. $s_0 \sim \mathcal{D}_{s_0}$, where $\mathcal{D}_{s_0}$ is an uniform distribution, and $w \sim \mathcal{D}_w$. According to Eq. 2 the distribution of reachable states at any given step is the linear combination of $\mathcal{D}_{s_0}$ and $\mathcal{D}_w$.

Given a reachable set $\mathcal{R}_t^s$, we wish to characterize a distribution of its safety probability p_t, a measure that indicates the portion of $\mathcal{R}_t^s$ "surface" that is safe:

$$p_t(s) = \int_{s \in \bar{\mathcal{S}}_u \cap \mathcal{R}_t^s} f_t(s)\ ds$$

For a linear stochastic system, suppose that s_0 is subject to a uniform distribution on $[\mathcal{L}_{s_0}, \mathcal{U}_{s_0}]$. The $\mathcal{L}_{s_0}, \mathcal{U}_{s_0} \in \mathbb{R}^n$ where n is the state dimension number of a system. If $\delta = \mathcal{U}_{s_0} - \mathcal{L}_{s_0}$, we have,

$$f_t(s) \leq \frac{1}{|\det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}$$

If noise also subjects to a uniform distribution, and its distribution is on $[\epsilon'_l, \epsilon'_h]$, then $\delta' = \epsilon'_h - \epsilon'_l$,

$$f_t(s) \leq \min \left(\frac{1}{|\det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}, \frac{1}{|\det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta'_i} \right)$$

We proved this upper bound on $f_t(s)$ in Appendix B.2.

Suppose the upper bound of $f_t(s)$ is U_f, the intersection between the reachable set $\mathcal{R}_t^s$ and the unsafe state set S_u is $S_u \cap \mathcal{R}_t^s$. Then,

$$p_t \geq 1 - U_f \int_{S_u \cap \mathcal{R}_t^s} ds$$

We denote the lower bound $1 - U_f \int_{S_u \cap \mathcal{R}_t^s} ds$ as $\hat{p}_t$.[2] When $\sum_{i=1}^M \hat{p}_t = M$, the system is verified to be safe in M steps.

In our motivating example, $\hat{p}_t$ is parameterized as $\hat{p}_t\left(\Pi_l^k, \phi\right)$, because generating $\mathcal{R}_t^s$ depends on Π_l^k and ϕ. In the following, we use $\hat{p}_t$ to represent $\hat{p}_t\left(\Pi_l^k, \phi\right)$ when it does not cause confusion.

3.6 Safety Shield for Stochastic Transition System

The shield algorithm is defined as a two-stage process. First, in state s_t, the neural network provides an action a_t, yielding state s_{t+1}. We detect whether s_{t+1} resides beyond the verified boundary $\mathcal{B}$ that is known to be safe. Second, if we detect that the action a_t will result in a state s_{t+1} that resides outside the desired boundary $\mathcal{B}$ and $s_{t+1} \notin \mathcal{R}_0$, we need a replacement a'_t for a_t. The new a'_t that provides such a state is supplied by the selected linear controller $K = \phi_{opt}(\Pi_l^k, t)$.

Taking a_t results in $\hat{s}_{t+1}$, and taking a'_t results in $\hat{s}'_{t+1}$. For the verified linear family function, we consider a larger noise term $\hat{w}_1$. Recall $\hat{w}_1 \in [\epsilon_1^l, \epsilon_1^u]$ and $\hat{w}_2 \in [\epsilon_2^l, \epsilon_2^u]$, $\epsilon_1^l < \epsilon_2^l$, $\epsilon_1^u > \epsilon_2^u$. The region $\mathcal{B}$ is defined by $\hat{s}'_{t+1} + \hat{w}_1$. Because the linear controller family is verified to be safe, the states in $\mathcal{B}$ and $\mathcal{R}_0$ are guaranteed to be safe. If either $\mathcal{B}$ or $\mathcal{R}_0$ includes the region defined by $\hat{s}_{t+1} + \hat{w}_2$, states in the region defined by $\hat{s}_{t+1} + \hat{w}_2$ are also guaranteed to be safe. However, if neither $\mathcal{R}_0$ nor $\mathcal{B}$ includes the region defined by $\hat{s}_{t+1} + \hat{w}_2$, because s_{t+1} can be outside the region considered during the controller verification phase, then

---

[2] In Appendix B.3, we provide a computationally efficient method to compute the upper bound of $\int_{S_u \cap \mathcal{R}_t^s} ds$. In Appendix B.4, provide a method to compute the lower bound of p_t.

taking action a_t may break the safety guarantee provided by our shield. Thus, we require the system to take action a_t' in this case. On the contrary, we do not need to replace a_t and can preserve the original behavior of the neural network.

4   Approach

4.1   Verify a Linear Controller Family

Our verification algorithm aims to maximize the safety probability of the reachable state at each step. If $\hat{p}_t(s)$ is always 1, that is $\mathcal{R}_t^s$ only defines safe elements, then its sum is equal to the total number of steps M in a trajectory. In this case, the controller is guaranteed to be safe. We thus seek a selector ϕ_{opt} such that it maximizes the lower bound of $\sum_{t=1}^{M} \hat{p}_t(\Pi_l^k, \phi)$:

$$\phi_{opt} = \mathtt{argmax}_{\phi} \sum_{t=1}^{M} \hat{p}_t(\Pi_l^k, \phi)$$

Suppose that the number of policies in Π_l^k is $\left| \Pi_l^k \right|$. Finding the optimal ϕ_{opt} using a brute-force approach would require traversing all the $\left| \Pi_l^k \right|^{\lceil \frac{M}{k} \rceil}$ possible combinations (i.e., all the possible ϕ) in the worst case. To improve on this, we consider three pruning strategies.

First, we keep track of the largest cumulative lower bound of all controllers $\mathcal{L}_{opt}$ for all visited ϕ. If $\mathcal{L}_{opt}$ reaches M, we can terminate immediately, and return the current ϕ. We say two selectors have the same prefix at step t if all their selections upto step t are identical. For a given step m, $\mathtt{PrefixSame}(\phi, m)$ denotes the set of selectors that have the same prefix as ϕ. Line 13 to line 16 of Algorithm 1 describes the first strategy. If there exists a constant $m < M, s.t. \left(m - \sum_{t=1}^{m} \hat{p}_t(\Pi_l^k, \phi) \right) > (M - \mathcal{L}_{opt})$, $\mathtt{PrefixSame}(\phi, m)$ can be removed from the search space, reducing the number of selectors that need to be considered during verification.

Second, we store all the k-th reachable sets for each k steps. The reachable sets at step t is denoted by $\mathtt{AllRSet}_t$. Consider two reachable sets $\mathcal{R}_k^1 \in \mathtt{AllRSet}_t$ an $\mathcal{R}_k^2 \in \mathtt{AllRSet}_t$, which are generated by different selectors. Suppose $\mathcal{R}_k^1 \subseteq \mathcal{R}_k^2$ and ϕ generates $\mathcal{R}_k^2$. Now, all selectors in $\mathtt{PrefixSame}(\phi, k)$ can be removed from the search space. This is because a smaller reachable set is always safer than a larger one. An example of this strategy is provided in our running example. At step 200, because the reachable set of $K_1 K_2$ is a subset of $K_1 K_1$ and $K_1 K_3$, we can safely remove $K_1 K_1$ and $K_1 K_3$ from consideration in determining the family of safe linear controllers. The second strategy corresponds to line 17 to line 25 in the algorithm. Line 19 and line 22 update $\mathtt{AllRSet}_t$ if any subset relationship is found between the elements of $\mathtt{AllRSet}_t$. Line 23 queries the selector generating the reachable set $\mathcal{R}'$. Line 20 and 24 prune the search space Ω.

Finally, we keep the reachable set $\mathcal{R}_t^s$ generated by ϕ for every k steps; this set is denoted by $\mathtt{PhiRSet}$ in Algorithm 1. $\mathtt{PhiRSet}_t$ is the stochastic reachable set generated by a selector ϕ at step t. $\mathtt{PhiRSet}_t$ is an invariant set if $\exists t' < t$,

$\texttt{PhiRSet}_t \subseteq \texttt{PhiRSet}_{t'}$. At step t, if the cumulative lower boundary $\mathcal{L}$ is t (i.e., the probability of safety violation is 0 upto this step), we can return this bound as M. Meanwhile, we construct a desired selector ϕ_{inv} by keeping the prefix of ϕ before step t, letting the action of the linear controller at step t govern future steps. This strategy is shown in lines 26 to 30. Line 26 computes an optimal ϕ as described above.

The overall structure of the algorithm takes the max simulation step M, the linear controller family Π_l^k, and the search space Ω containing all selectors ϕ as inputs. It returns the optimal selector ϕ_{opt} that maximizes $\sum_{t=1}^{M} \hat{p}_t \left(\Pi_l^k, \phi_{opt} \right)$, and $\mathcal{L}_{opt} = \sum_{t=1}^{M} \hat{p}_t \left(\Pi_l^k, \phi_{opt} \right)$. The outer for-loop at line 2 tries to traverse possible selectors. However, when running, the search space will be pruned, and thus not all the selectors will be visited. Given a selector ϕ, the for-loop at line 5 checks every k steps of the simulation. $\mathcal{L}_{opt}$ is initialized as 0. If we find a better $\mathcal{L}$ at the end of the loop, it is updated at line 33. From lines 7 to 9, the algorithm computes a cumulative p_t from step $k \cdot i$ to step $\texttt{min}(M,m)$, and accumulates the safety probability lower bound $\hat{p}_t$ for step $k \cdot i$ to step $\texttt{min}(M,m)$. Lines 10 to 12 computes $\mathcal{R}_m^s$, and then adds $\mathcal{R}_m^s$ to $\texttt{AllRSet}_m$ and $\texttt{PhiRSet}_m$. Line 10 computes $\mathcal{R}_t^s$ as a tuple $(\mathcal{T}_{s_t}, \mathcal{T}_{w_t})$. The parts of the algorithm involving various optimization strategies have been described above.

4.2   Safe Training with Shield

We use a shield defined in terms of the verified controller family computed by Algorithm 1 to monitor and ensure the safety of a neural network controller, π_{nn}. Algorithm 2 describes the operation of the shield. Given a neural network controller π_{nn}, a linear controller family Π_l^k, synthesized selector ϕ_{opt}, noise used for verification $\hat{w}_1$, the real noise of system $\hat{w}_2$, and the state at step t, s_t, Algorithm 2 returns a shielded action $\hat{a}_t$. $\texttt{Region}$ is a function computing all possible states allowed by given state s and noise w where $w \in [\epsilon_l, \epsilon_u], \texttt{Region}(s, w) = \{s' | s' - s \in [\epsilon_l, \epsilon_u]\}$. In the DRL training process, the neural controller predicts actions (more precisely, the neural network predicts action distributions, and actions are sampled from the distribution) and executes these actions on a system. The neural controller learns from the reward that is associated with different actions during training. To apply the shield during training, before an action is executed, we apply Algorithm 2 to improve the likelihood the chosen action is safe. A particularly important instance of this approach is when $\forall t \leq M, \hat{p}_t = 1$, as formalized by the following theorem.

Theorem 1. *(Soundness of Shield) If* $\sum_{i=1}^{M} \hat{p}_t = M$, $s_0 \in \mathcal{R}_0$, *and* $\forall t < M, a_t$ *is generated by Algorithm 2, then* $\forall t \leq M, s_t \notin S_u$.

5   Experimental Results

We have applied the verification algorithm introduced in the previous section on a variety of stochastic transition systems, whose number of dimensions ranges

---

Algorithm 1: Synthesis algorithm for ϕ_{opt}

Input: M, Π_l^k, Ω
Output: $\phi_{opt}, \mathcal{L}_{opt}$

1 $\mathcal{L}_{opt} \leftarrow 0$;
2 **for** $\phi \in \Omega$ **do**
3     $\mathcal{L} \leftarrow 0$;
4   PhiRSet $\leftarrow \emptyset$;
5   **for** $i \leftarrow 0$ **to** $\lfloor \frac{M}{k} \rfloor$ **do**
6        $m \leftarrow k \cdot (i+1)$;
7   **for** $j \leftarrow ki+1$ **to** $\min(M, m)$ **do**
8           $\mathcal{L} \leftarrow \mathcal{L} + \hat{p}_j(\Pi_l^k, \phi)$;
9   **end**
10   Compute $\mathcal{R}_m^s$;
11   AllRSet$_m \leftarrow$ AllRSet$_m \cup \{\mathcal{R}_m^s\}$;
12   PhiRSet $\leftarrow$ PhiRSet$_m \cup \{\mathcal{R}_m^s\}$;
      `// 1st strategy, keeping the optimal cumulative lower bound.`
13   **if** $m - \mathcal{L} > M - \mathcal{L}_{opt}$ **then**
14          $\Omega \leftarrow \Omega / \texttt{PrefixSame}(\phi, m)$;
15   break;
16   **end**
      `// 2nd strategy, between-selector cutting`
17   **for** $\mathcal{R}' \in$ AllRSet$_m$ **do**
18   **if** $\mathcal{R}' \subset \mathcal{R}_m^s$ **then**
19   AllRSet$_m \leftarrow$ AllRSet$_m / \mathcal{R}_m^s$;
20             $\Omega \leftarrow \Omega / \texttt{PrefixSame}(\phi, m)$;
21   **else if** $\mathcal{R}_m^s \subset \mathcal{R}'$ **then**
22   AllRSet$_m \leftarrow$ AllRSet$/ \mathcal{R}'$;
23   Query the ϕ' computing $\mathcal{R}'$;
24             $\Omega \leftarrow \Omega / \texttt{PrefixSame}(\phi', m)$;
25   **end**
      `// 3rd strategy, invariant of reachable set.`
26   **for** $\mathcal{R}' \in$ PhiRSet **do**
27   **if** $\mathcal{R}_m^s \subseteq \mathcal{R}'$ **then**
28   Compute ϕ_{inv};
29   **return** ϕ_{inv}, M;
30   **end**
31   **end**
32   **end**
33   **if** $\mathcal{L} > \mathcal{L}_{opt}$ **then**
34       $\mathcal{L}_{opt} = \mathcal{L}$, $\phi_{opt} = \phi$;
35   **end**
36   **if** $\mathcal{L} = M$ **then**
37   **return** ϕ_{opt}, M;
38   **end**
39 **end**
40 **return** $\phi_{opt}, \mathcal{L}_{opt}$;

Algorithm 2: Shield algorithm for π_{nn}

Input: $\pi_{nn}, \Pi_l^k, \phi_{opt}, \hat{w}_1, \hat{w}_2, s_t$
Output: $\hat{a}_t$

1 $K \leftarrow \phi_{opt}(\Pi_l^k)$;
2 $a_{nn} \leftarrow \pi_{nn}(s_t)$;
3 $a_{safe} \leftarrow K \cdot s_t$;
4 $\hat{s}_{t+1} = dynamics(s_t, a_{nn})$;
5 $\hat{s}'_{t+1} = dynamics(s_t, a_{safe})$;
6 **if** $(\texttt{Region}(\hat{s}_{t+1}, \hat{w}_2) \subseteq \texttt{Region}(\hat{s}'_{t+1}, \hat{w}_1)) \cup \mathcal{R}_0$ **then**
7 $\quad$ **return** a_{nn};
8 **else**
9 $\quad$ **return** a_{safe};
10 **end**

---

from 2 to 896. We associate various safety constraints with each benchmark, synthesizing a verified linear controller family that seeks to guarantee these properties hold, and use that family to train a neural network with additional performance (a.k.a. liveness) objectives. The resulting system consists of a performance-sensitive neural network that is trained with the awareness of safety constraints coupled with a safety shield represented by the linear controller family. We train the neural network using the Proximal Policy Optimization (PPO)[24]. Our experiments are run on Intel(R) Xeon(R) Gold 6130 CPU supporting 64 threads, but all executions use only a single thread.

Benchmarks Benchmarks Pendulum, Cartpole, Carplatoon, and Helicopter are adapted from [12]; Oscillator comes from [17]. Pendulum and Cartpole are two classical control models. We have discussed Pendulum in detail earlier. Cartpole is a control system for a moving cart with a vertical pole; a safety property requires the cart to move without causing the pole to fall. Carplatoon models 8 vehicles forming a platoon, maintaining a safe relative distance among one another. Oscillator consists of a two-dimensional switched oscillator plus a 16-order filter. The filter smoothens the input signals and has a single output signal; the safety property requires the output to remain below a threshold. Helicopter provides a longitudinal motion model of a helicopter; its safety constraint requires that it operate within a specified region. Each of these benchmarks also has an associated liveness property defined in terms of a performance objective extracted from the application's dynamics.

The other benchmarks in our suite are *stacked* from the first 5 systems, but given different safety properties; perturbations are also added to different stacking elements to yield different behaviors. The prefix number denotes the number of systems stacked. Given a stochastic transition system as defined in Sec. 3.4, we stack the A, B matrices of the linear control system as diagonal elements of a large matrix. For example, for 2-Pendulum, we stack $A, B \in \mathbb{R}^{2 \times 2}$

thus:

$$A' = \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & A \end{bmatrix}, B' = \begin{bmatrix} B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \cdot P_B$$

Here A' and B' are 2-Pendulum's transition matrices and A and B come from the specification of Pendulum as defined in Sec. 2.1. The diagonal elements in the perturbation matrices $P_B \in \mathbb{R}^{4\times4}$ are sampled randomly from the range $[0.95, 1.05]$; all non-diagonal elements are 0. Because we need to compute $B' \cdot P_B \cdot a_t$, P_B perturbs the input action a_t by a scaling factor. Similarly, we also stack safety constraints. For example, the safety constraints of 2-Pendulum have lower bound $L' \in \mathbb{R}^4$ and upper bound $U' \in \mathbb{R}^4$. The Pendulum has lower bound $L \in \mathbb{R}^2$ and upper bound $U \in \mathbb{R}^2$,

$$L' = P_L \odot [L, L], U' = P_U \odot [U, U]$$

$P_L, P_U \in \mathbb{R}^4$ and their elements are sampled from $[0.95, 1.05]$. Meanwhile, we ensure that every element of P_L is smaller than P_U. $\odot$ denotes element-wise multiplication. While we could apply our technique to each component of these stacked systems individually, we evaluate our approach on the high dimensional case directly to demonstrate the scalability of our algorithms.

Verification The verification results for our benchmarks are shown in Table 2. Although our system supports probabilistic guarantees, all the benchmarks in our experiments were able to be fully verified. The probability safety lower bound was used only to prune the search space explored by Algorithm 1. To achieve 100% safety, we iteratively run Algorithm 1 on different linear controller families. Because the linear controller family is generated with randomly sampled Q and R values, the quality of its controllers is hard to guarantee *a priori*. Thus, if we cannot find a verified selector which maximizes the sum of p_t to M after checking at most 100 possible choices for ϕ, we regenerate a linear controller family by sampling new Q and R values and re-running Algorithm 1 on it again. We manually tuned the noise term $\hat{w}_1$ and let it be as large as possible for our experiments. Larger value of $\hat{w}_1$ gives a less restrictive shield and thus allow the system to explore more states. $|\phi|$ is the number of distinct linear controllers in the verified controller family. E.g., if $\phi = (K_1, K_2, K_1)$, $|\phi| = 2$.

The number before $\pm$ in the last three columns signifies the mean of our results - we run the verification algorithm 10 times for each benchmark; the number after $\pm$ is the standard deviation. When iterating on the selector space Ω, selectors with smaller sizes are visited first. Although we provide 10 potential linear controllers for each linear controller family, all of our benchmarks require a much smaller set to generate a safe controller combination. The verification time per Π_l^k column contains the running time of generating an LQR linear controller family with 10 potential controllers and running Algorithm 1 once.

The dimension is not the only fact that affects the verification time. Different safety properties and system dynamics can also impact verification time. For

Benchmarks	state dim	action dim	M	k	$\hat{w}_1$	$|\phi|$	ver. time per $|\Pi_l^k|$	total ver. time
Pendulum	2	1	500	100	1.5e-2	2.00± 0.00	0.54s± 0.05s	0.82s± 0.33s
Cartpole	4	1	500	100	3e-3	1.10± 0.30	0.80s± 0.26s	1.24s± 0.45s
Carplatoon	15	8	1000	100	2e-3	1.80± 1.66	1.07s± 0.25s	3.14s± 4.26s
Oscillator	18	2	1000	100	4e-3	1.80± 0.98	0.69s± 0.08s	0.72s± 0.21s
Helicopter	28	6	1000	100	2e-3	2.50± 1.86	1.36s± 0.42s	2.14s± 0.86s
2-Pendulum	4	2	500	100	1.5e-2	2.10± 0.30	0.75s± 0.20s	1.04s± 0.24s
2-Cartpole	8	2	500	100	3e-3	2.30± 1.27	0.81s± 0.19s	1.26s± 0.69s
2-Carplatoon	30	16	1000	100	2e-3	3.00± 2.32	1.32s± 0.32s	2.44s± 3.53s
2-Oscillator	36	4	1000	100	4e-3	2.30± 0.78	0.83s± 0.14s	0.99s± 0.45s
2-Helicopter	56	12	1000	100	2e-3	2.70± 2.24	4.37s± 3.92s	5.54s± 6.76s
4-Pendulum	8	4	500	100	1.5e-2	1.80± 0.40	0.56s± 0.06s	0.62s± 0.23s
4-Cartpole	16	4	500	100	3e-3	1.80± 0.40	0.97s± 0.20s	1.94s± 1.00s
4-Carplatoon	60	32	1000	100	2e-3	4.80± 2.71	1.17s± 0.20s	1.83s± 0.61s
4-Oscillator	72	8	1000	100	4e-3	2.70± 0.46	1.34s± 0.27s	1.52s± 0.39s
4-Helicopter	112	24	1000	100	2e-3	2.50± 1.86	8.30s± 3.33s	8.28s± 3.99s
8-Pendulum	16	8	500	100	1.5e-2	1.80± 0.40	0.97s± 0.13s	0.94s± 0.47s
8-Cartpole	32	8	500	100	3e-3	6.00± 1.84	1.18s± 0.33s	1.45s± 0.82s
8-Carplatoon	120	64	1000	100	2e-3	6.20± 0.98	4.03s± 0.75s	5.06s± 1.86s
8-Oscillator	144	16	1000	100	5e-3	2.80± 0.40	5.78s± 1.58s	38.60s± 66.35s
8-Helicopter	224	48	1000	100	2e-3	2.60± 2.01	45.03s± 31.45s	70.11s± 92.06s
16-Helicopter	448	16	1000	100	2e-3	2.80± 1.89	164.05s± 36.52s	458.17s± 343.75s
32-Helicopter	896	32	1000	100	2e-3	2.30± 1.27	2115.20s± 1090.29	2962.55s± 2907.33s

Table 2. Verification results. The state dim and action dim columns denote the number of dimensions in the state and action space of the benchmark, respectively. Stacking system benchmarks n-* have n times the number of dimensions as their single system counterpart. M is the number of execution time steps considered. The time interval between choosing a new linear controller is k. We use $\hat{w}_1$ to represent the verified max noise for each step in this system. $|\phi|$ is the number of linear controllers the optimal controller selects; E.g., if $\phi = (K_1, K_2, K_1)$, $|\phi| = 2$. The verification time per $|\Pi_l^k|$ is the time for verifying a single linear controller family, while total ver. time indicates the time to find a controller combination that is guaranteed to be safe (i.e., $\sum_{t=1}^{M} = M$).

example, 8-Carplatoon has 120 dimensions but only requires 4.03 seconds. to verify on average, while the 4-Helicopter benchmark with 112 dimensions requires 8.30 seconds to verify. For Pendulum and its stacked systems, verification time per linear controller family is close to total verification time, implying that there was little need to regenerate new controller instantiations. For more complicated benchmarks such as 8-Oscillator and 8-Helicopter, the linear controller families needed to be regenerated more often, increasing total verification time. Nonetheless, verification times, even for challenging benchmarks like 32-Helicopter with 896 dimensions, required less than 1 hour on average; 8-Helicopter with 224 required 70.11 secs to verify on average.

Effectiveness We train a neural controller using the safety guarantees captured by our verified linear controller family. Table 3 demonstrates the effectiveness of our approach on overall network performance. First, although the LQR controller is verified to be safe, it can nonetheless perform poorly with respect to

performance objectives. However, if we use it as a shield for a performant neural controller, we hope to realize both performance as well as safety benefits.

To explore the validity of this thesis, we train a neural controller for each singly stacked benchmark for 100K steps with and without a shield. For a stacked benchmark B of the form n-B, we train its corresponding controller for $n \cdot 100K$ steps. Since the shielded controller is known to guarantee safety, it can be trained with the liveness reward R_{live} only. Because the neural controller without the shield needs to keep safety in mind, it is trained with reward $R_{safe} + R_{live}$. Intuitively, a training step that satisfies a particular objective gets a positive reward.

We train these networks by simulating 500 trajectories for each benchmark. The comparison between the rewards of the different controllers are summarized in the Table 3. The table presents normalized performance characteristics of the shielded controller relative to the base PPO algorithm without augmentation of a safety planner, and the LQR family that is implemented without performance objectives. Thus, numbers greater than one in the column labeled Shield/PPO indicates that the controller trained in conjunction with the safety planner outperformed the PPO-only trained algorithm. A similar interpretation holds for the column labeled Shield/LQR. While it is not surprising that controllers trained with both safety and performance (Shield) would outperform those that are only aware of safety (LQR), it is notable that the shielded the controller has a higher performance reward than the PPO-trained controller on 15 of the 20 benchmarks considered. The last two columns indicate the number of safety violations encountered - PPO trained networks exhibited a non-negligible number of safety violations on every benchmark; since our verification algorithm was able to generate a provably safe shield for each benchmark, the safety-augmented controller exhibited no violation in any of the benchmarks.

Random Selector Without our algorithm, a random selector is not likely to generate a good linear combination with verifiable safety property. The experiments in Table 4 show that simply randomly picking linear controllers is not a viable strategy. We selected the 8-stacked benchmarks and the same noise term $\hat{w}_1$ as given in Table 2. We randomly generated 100 linear controller families, each with size 10. For each linear controller family, a random selector ϕ randomly picks up a linear controller from the linear controller family every k steps. For each linear controller family, we use the random selector to generate 10 linear combinations and compute $\mathcal{L}$. Here, $\mathcal{L} = \sum_{t=1}^{M} \hat{p}_t \left(\Pi_l^k, \phi \right)$, where ϕ is the random selector. $\mathcal{L}/M$ measures the sum of the safety probability lower bounds generated by a randomly selected combination. Except for the 8-Carplatoon benchmark, the max of $\mathcal{L}/M$ does not achieve 100% safety.

6   Related Work and Conclusions

There has been significant recent interest in exploring techniques to enhance the safety of learning-enabled systems. Recent works [22,25,18,19] have defined suit-

Benchmarks	$\hat{w}_2$	Shield/PPO	Shield/LQR	PPO vio.	Shield vio.
Pendulum	1.5e-3	**2.65**	**8.59**	1437	0
Cartpole	3e-4	**1.36**	**3.54**	959	0
Carplatoon	2e-4	**1.83**	**30.58**	69	0
Oscillator	4e-4	**1.37**	**6.79**	3	0
Helicopter	2e-4	**1.04**	**1.49**	30	0
2-Pendulum	1.5e-3	**1.77**	**2.84**	2375	0
2-Cartpole	3e-4	0.64	**2.59**	1775	0
2-Carplatoon	2e-4	0.76	**11.84**	1137	0
2-Oscillator	4e-4	**1.18**	**3.33**	46	0
2-Helicopter	2e-4	**1.07**	**1.48**	277	0
4-Pendulum	1.5e-3	**2.68**	**2.34**	4736	0
4-Cartpole	3e-4	0.60	**2.21**	3529	0
4-Carplatoon	2e-4	0.60	**15.94**	1863	0
4-Oscillator	4e-4	**2.20**	**3.77**	150	0
4-Helicopter	2e-4	**1.17**	**1.33**	405	0
8-Pendulum	1.5e-3	**1.28**	**1.93**	11305	0
8-Cartpole	3e-4	**1.17**	**2.55**	12680	0
8-Carplatoon	2e-4	0.75	**12.5**	7250	0
8-Oscillator	4e-4	**1.99**	**3.29**	579	0
8-Helicopter	2e-4	**1.33**	**1.24**	1388	0

Table 3. Effectiveness of the safety planner on training safe and performant networks. The $\hat{w}_2$ is the noise size of the simulation environment. It is set to be $\frac{\hat{w}_1}{10}$. Shield/PPO is the ratio of R_{live} gained by the shielded neural controller versus the PPO neural controller. Similarly, Shield/LQR is the ratio of R_{live} gained by the shielded neural controller versus the LQR controller combination used for the shield. Bold-faced entries indicate when the shielded controller outperforms its counterpart. PPO vio. and Shield vio. are the number of safety violations encountered during training.

able verification methodologies that are capable of providing stronger guarantees for open-loop LECs. For closed-loop systems discussed in our work, [1,6,13] focus on specifying controller safety as an additional reward. By changing the reward, these methods seek to increase the safety characteristics of the learnt controller. These approaches are different from ours insofar as we consider provably verifiable methods applied independently of a training-based reward framework. [29,15,16,11,27] verify a neural network directly. However, the complexity of the networks, the amount of computation required, and the approximation introduced during the verification process nake these methods difficult to scale to high dimension problems. Another line of work explores verifiability by applying imitation learning techniques on the subject networks [31,3,5], and use the result as a shield [2] to ensure the safety of a neural network during training [3] or post-deployment [31]. However, since these approaches must align the imitated simple controller, which is heavily biased towards safety considerations, with the neural controller that also takes performance objectives into account, verification

Benchmarks	avg.	min	max
Pendulum	36.94%± 12.38%	3.60%	88.00%
Cartpole	19.52%± 6.53%	0.00%	37.92%
Carplatoon	88.09%± 23.14%	13.59%	100.00%
Oscillator	91.64%± 2.66%	69.10%	96.20%
Helicopter	18.47%± 16.77%	0.00%	66.90%

Table 4. Safety Probability Lower Bound $\mathcal{L}/M$ of a random selector strategy.

is challenging, especially when considering sophisticated performance objectives, or high dimensional systems.

There also exist tools for synthesizing controllers [30,26]. Similar to [30], our verification algorithm supports linear, potentially time-varying, discrete-time systems that are perturbed by a stochastic disturbance, but our algorithm is demonstrably more scalable. In our work, we verify the safety property of linear controller family generated by LQR, and learning other non-functional properties using standard reinforcement learning algorithms supplemented with the verified linear controller family. Reinforcement learning algorithms can generally support complex properties defined with various objectives. For example, [7,28] encode LTL specifications into rewards and train neural controllers with reinforcement learning. However, simply encoding specifications as rewards cannot provide any guarantee on critical properties. In contrast, our work can provide desired verifiable results, exploiting the power of learning other complex properties using reinforcement learning. There also exists approaches that consider falsification methods [9,4,8,10,23] that aims at finding potential unsafe safes in CPS systems. They can work with complex specifications and high dimensions systems. However, they do not provide provably verifiable guarantees.

In this paper, we present a new pipeline that synthesizes a neural network controller with safety guarantee. First, we propose a linear controller family intended to stabilize a system. Then, we verify the linear controller with respect to a safety property. This verified linear controller family protects the safety of network training, and meanwhile, also ensures the deployed controller does not violate safety constraints. Because safety verification is decoupled from the training process, our approach has pleasant scalability characteristics that are sensitive to performance objectives. In addition, because we inject a safety planner into the learning process, the resulting controller is trained with safety considerations in mind, yielding high-quality verified learning-enabled controllers that often outperform their non-verified counterparts. The key insight of our work is that we can decouple properties relevant for learning from those necessary for verification. Base on the verified properties, we use reinforcement learning to learn other properties. We propose to continue studying this interplay as part of future work.

References

1. Achiam, J., Held, D., Tamar, A., Abbeel, P.: Constrained policy optimization. arXiv preprint arXiv:1705.10528 (2017)
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18) (2017), http://arxiv.org/abs/1708.08611
3. Anderson, G., Verma, A., Dillig, I., Chaudhuri, S.: Neurosymbolic reinforcement learning with formally verified exploration. Advances in Neural Information Processing Systems **33** (2020)
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257. Springer (2011)
5. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. CoRR **abs/1805.08328** (2018), http://arxiv.org/abs/1805.08328
6. Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe model-based reinforcement learning with stability guarantees. arXiv preprint arXiv:1705.08551 (2017)
7. Camacho, A., Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI. vol. 19, pp. 6065–6073 (2019)
8. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In: NASA Formal Methods Symposium. pp. 127–142. Springer (2015)
9. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. Journal of Automated Reasoning **63**(4), 1031–1053 (2019)
10. 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)
11. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 157–168 (2019)
12. Fan, C., Mathur, U., Mitra, S., Viswanathan, M.: Controller synthesis made real: Reach-avoid specifications and linear dynamics. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 347–366. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_19, https://doi.org/10.1007/978-3-319-96145-3_19
13. Garcia, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**, 1437–1480 (2015)
14. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning. pp. 1861–1870. PMLR (2018)
15. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–22 (2019)

16. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. CoRR **abs/1811.01828** (2018), http://arxiv.org/abs/1811.01828

17. Jordan, D., Smith, P., Smith, P.: Nonlinear ordinary differential equations: an introduction for scientists and engineers, vol. 10. Oxford University Press on Demand (2007)

18. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 97–117. Springer International Publishing, Cham (2017)

19. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 443–452. Springer International Publishing, Cham (2019)

20. Li, S., Bastani, O.: Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 7166–7172. IEEE (2020)

21. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)

22. Paulsen, B., Wang, J., Wang, C.: Reludiff: Differential verification of deep neural networks. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). pp. 714–726. IEEE (2020)

23. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18 (2017)

24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arxiv 2017. arXiv preprint arXiv:1707.06347

25. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)

26. Soudjani, S.E.Z., Gevaerts, C., Abate, A.: $faust^2$: Formal abstractions of uncountable-state stochastic processes. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 272–286. Springer (2015)

27. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 147–156 (2019)

28. Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Teaching multiple tasks to an rl agent using ltl. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 452–461 (2018)

29. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. arXiv preprint arXiv:2004.05519 (2020)

30. Vinod, A.P., Gleason, J.D., Oishi, M.M.K.: SReachTools: A MATLAB Stochastic Reachability Toolbox, p. 33–38. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3302504.3311809

31. Zhu, H., Xiong, Z., Magill, S., Jagannathan, S.: An inductive synthesis framework for verifiable reinforcement learning. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 686–701 (2019)

Appendix A   Soundness of Shield

Theorem. 1 *(Soundness of Shield) If $\sum_{i=0}^{M} \hat{p}_t = M$, $s_0 \in \mathcal{R}_0$, and $\forall t < M$, a_t is generated by Algorithm 2, then $\forall t \leq M, s_t \notin S_u$.*

Proof. Supposing the initial state of a system is s_0, $\forall s_0 \in \mathcal{R}_0$, based on Algorithm 2, if $\texttt{Region}(\hat{s}_1, \hat{w}_2) \subseteq \texttt{Region}(\hat{s}_1', \hat{w}_1) \cup \mathcal{R}_0$, the action a_{nn} will be executed. Otherwise, the a_{safe} will be executed. This execution will ensure the next state s_1 lands in the reachable set $\mathcal{R}_1^s \cup \mathcal{R}_0$, where $\mathcal{R}_1^s$ is the stochastic reachable set of the selected linear controller. Thus, $s_0 \in \mathcal{R}_0 \implies s_1 \in \mathcal{R}_1^s \cup \mathcal{R}_0$. Similarly, for the the state s_t at step t and state s_{t+1} at step $t + 1$, we have, $\forall t < M, s_t \in \mathcal{R}_t^s \implies s_{t+1} \in \mathcal{R}_{t+1}^s \cup \mathcal{R}_0$. By induction, $\forall t \leq M, s_0 \in \mathcal{R}_0 \implies s_t \in \bigcup_{i=0}^{t} \mathcal{R}_i^s$. $\sum_{i=0}^{M} \hat{p}_t = M \implies \hat{p}_t = 1.0$, thus we know that all the reachable sets before time step M has no overlapping with S_u. $\forall t \leq M, \mathcal{R}_t^s \cap S_u = \emptyset$. Algorithm 2 ensures that $\forall t \leq M, s_t \in \bigcup_{i=0}^{t} \mathcal{R}_i^s$. Thus, $\forall t \leq M, s_t \in \bigcup_{i=0}^{t} \mathcal{R}_i^s, \mathcal{R}_t^s \cap S_u = \emptyset \implies s_t \notin S_u$.

Appendix B   Probabilistic Reachable Analysis

In this section, we will analyze the probabilistic reachability. Instead of merely analyzing which states are reachable, we finally provide a probabilistic lower bound for p_t.

Appendix B.1   Probability Density Function of Reachable Set

We represent the reachable set for the noise-free transition $\mathcal{T}_{s_t} s_0$ as $\mathcal{R}_t$. The reachable set for the stochastic transition at step t is $\mathcal{R}_t^s$. $\mathcal{R}_0 = \mathcal{R}_0^s$. $\mathcal{R}_t \subseteq \mathcal{R}_t^s$. We hope to calculate the safety probability of reachable sets on continuous space, thus need to find the probability density function (PDF) for a reachable set $\mathcal{R}_t^s$. Suppose $f_t(s)$ is the PDF of the distribution s_t is subject to. Since $s_0 \sim \mathcal{D}_{s_0}$ where $\mathcal{D}_{s_0}$ is a uniform distribution, and the stochastic term $w_0 \sim \mathcal{D}_{w_0}$, the reachable distribution of any given step is the linear combination of $\mathcal{D}_{s_0}$ and $\mathcal{D}_{w_0}$.

Assuming there is a mapping r from x to y, $y = r(x)$, and according to the change of variable formula of PDF, the PDF of x is f_x, the PDF of y is

$$f_y(y) = f_x\left(r^{-1}(y)\right) \left| \frac{d}{dy} r^{-1}(y) \right| \tag{3}$$

Theorem 2. *Suppose that X is a random variable taking values in $S \subseteq \mathbb{R}^n$, and that X has a continuous distribution with probability density function f_x. Suppose $Y = r(X)$ where r is a differentiable function from S onto $T \subseteq \mathbb{R}^n$. Then the probability density function g of Y is given by*

$$f_y(y) = f_x(x) \left| \det\left(\frac{dx}{dy} \right) \right|, \quad y \in T \tag{4}$$

Proof. The result follows the multivariate change of variables formula in calculus. If $B \subseteq T$ then

$$\mathbb{P}(Y \in B) = \mathbb{P}[r(X) \in B] = \mathbb{P}\left[X \in r^{-1}(B)\right] = \int_{r^{-1}(B)} f_x(x)dx$$

Using the change of variables $x = r^{-1}(y), dx = \left|\det\left(\frac{dx}{dy}\right)\right| dy$ we have

$$\mathbb{P}(Y \in B) = \int_B f\left[r^{-1}(y)\right] \left|\det\left(\frac{dx}{dy}\right)\right| dy$$

So it follows that g defined in the theorem is a PDF for Y.

Let $s_t = \mathcal{T}_{s_t} s_0$, $w_t = \mathcal{T}_{w_t} w$. Suppose the PDF of the distribution that $\mathcal{T}_t s_0$ subjects to is $g_t(s_t)$ and the PDF of distribution that $\mathcal{T}_{w_t} w$ subjects to is $h_t(w_t)$. The PDF of s_0's distribution and w's distribution is $g_0(s_0)$ and $h_0(w)$ respectively.

Applying the (4) to g_t, because $s_0 = \mathcal{T}_{s_t}^{-1} s_t$, we know that $\frac{ds_0}{ds_t} = \mathcal{T}_{s_t}^{-1}$, thus,

$$g_t(s_t) = \frac{g_0\left(\mathcal{T}_{s_t}^{-1} s_t\right)}{|\det(\mathcal{T}_{s_t})|} \tag{5}$$

Applying the (4) to h_t,

$$h_t(w_t) = \frac{h_0\left((\mathcal{T}_{w_t})^{-1} w_t\right)}{|\det(\mathcal{T}_{w_t})|} \tag{6}$$

In the case that will not cause confusion, we remove the subscripts of s_t and w_t, and write these two equations as

$$g_t(s) = \frac{g_0\left(\mathcal{T}_{s_t}^{-1} s\right)}{|\det(\mathcal{T}_{s_t})|}$$

$$h_t(w) = \frac{h_0\left((\mathcal{T}_{w_t})^{-1} w\right)}{|\det(\mathcal{T}_{w_t})|}$$

Given a reachable set $\mathcal{R}_t^s$, we wish to characterize p_t, a measure of how many states of $\mathcal{R}_t^s$ are safe:

$$p_t = \int_{s \in \bar{\mathcal{S}}_u \cap \mathcal{R}_t^s} f_t(s) \ ds$$

Here, $f_t(s)$ is the PDF of $\mathcal{R}_t^s$ and it depends on $g_t(s)$ and $h_t(w)$. $\bar{\mathcal{S}}_u$ is the state set that satisfies the safety properties. When $\mathcal{R}_t^s \subseteq \bar{\mathcal{S}}_u$, $p_t = 1$
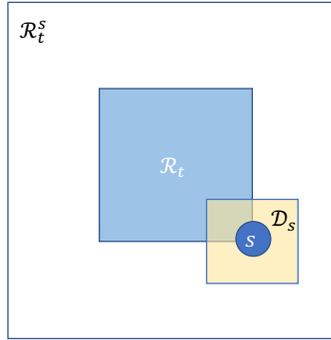

Fig. 5. An illustrating example for Eq. 7 in 2-d case. The $\mathcal{D}_s$ is computed with $\mathcal{D}_w$ and s. It can be seen as the result of translating $\mathcal{D}_w$ with vector s. The PDF of $\mathcal{R}_t$ is $g_t(s')$, the PDF of $\mathcal{D}_s$ is $h_t(s' - s)$. $\mathcal{D}_s$ acts as filter and moves as s changes.

According to the definition in Sec. 3.4, $f_t(s) = g_t(s) + h_t(w)$. The sum of two random variables is distributed as the convolution of their probability densities. Thus, f_t is distributed as the convolution of the distributions g_t and h_t. Given a domain $\mathcal{D}_w = \{w | h_t(w) > 0\}$, $\mathcal{D}_s = \{s' | s \in \mathcal{R}_t^s, (s' - s) \in \mathcal{D}_w\}$; we have that

$$f_t(s) = (g_t * h_t)(s) = \int_{s' \in \mathcal{D}_s} g_t(s') h_t(s' - s) \; ds' \tag{7}$$

Appendix B.2   Upper Bound of Probability Density Function

Now, we consider the upper bound of $f_t(s)$,

Theorem 3. *The s_0 subjects to a uniform distribution on $[\mathcal{L}_{s_0}, \mathcal{U}_{s_0}]$. $\mathcal{L}_{s_0}, \mathcal{U}_{s_0} \in \mathbb{R}^n$; n is the number of state dimensions. Let $\delta = \mathcal{U}_{s_0} - \mathcal{L}_{s_0}$,*

$$f_t(s) \leq \frac{1}{|det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}$$

Proof. The s_0 subjects to a uniform distribution on $[\mathcal{L}_{s_0}, \mathcal{U}_{s_0}]$.

$$g_0(s) = \begin{cases} \frac{1}{\prod_{i=0}^{n-1} \delta_i} & s \in \mathcal{R}_0 \\ \\ 0 & otherwise \end{cases}$$

From eq. 5, we know that

$$g_t(s) = \begin{cases} \frac{1}{|det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i} & s \in \mathcal{R}_t \\ \\ 0 & otherwise \end{cases}$$

Applying linear transformation $\mathcal{T}_t$ to $\mathcal{R}_0$, we get $\mathcal{R}_t$.
$f_t(s) = (g_t * h_t)(s_t) = \int_{s' \in \mathcal{D}_s} g_t(s') h_t(s - s') \; ds'$. Thus,

$$f_t(s) \leq \frac{1}{|det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i} \int_{s' \in \mathcal{D}_s} h_t(s' - s) \; ds'$$

$h_t(s)$ is a PDF, thus $\int_{s \in \mathcal{D}_s} h_t(s' - s) \; ds \leq 1$. We proved that

$$f_t(s) \leq \frac{1}{|det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}$$

Corollary 1. *Suppose the noise on every step subjects to an uniform distribution on $[\epsilon_l', \epsilon_h']$; $\varepsilon_l', \varepsilon_h' \in \mathbb{R}^n$. $\delta' = \epsilon_h' - \epsilon_l'$,*

$$f_t(s) \leq \min \left(\frac{1}{|det(\mathcal{T}_{s_i})| \prod_{i=0}^{n-1} \delta_i}, \frac{1}{|det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta_i'} \right)$$

The Theorem 3 can be extended if the noise subjects to the uniform distribution.

Proof. We assume the noise subjects to the uniform distribution on $[\epsilon'_l, \epsilon'_h]$. According to Eq. 6, we have

$$
h_t(s - s') = \begin{cases} \frac{1}{|det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta'_i} & s' \in \mathcal{D}_s \\[2em] 0 & otherwise \end{cases}
$$

g_t is PDF, its integration is smaller or equal to 1. According to Eq. 7,

$$
f_t(s) \le \frac{1}{|\det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta'_i} \int_{s' \in \mathcal{D}_s} g(s') \, ds' \le \frac{1}{|\det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta'_i}
$$

Merge the conclusion in the Theorem 3,

$$
f_t(s) \le \min \left(\frac{1}{|\det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}, \frac{1}{|\det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta'_i} \right)
$$

Appendix B.3   Compute $\int_{\mathcal{S}_u \cap \mathcal{R}_t^s} ds$

Now we have the upper bound of $f_t(s)$. If we know which part of $\mathcal{R}_t^s$ violates the safety constraints, we can integrate the upper bound of $f_t(s)$ on this unsafe part to compute the upper bound of probability to reach unsafe region at step t. However, the computation of the part of $\mathcal{R}_t^s$ which violates the safety constraints is inefficient as the dimension grows, so we computed the over-approximation $\mathcal{A}(\mathcal{R}_t^s) \subset \mathbb{R}^n$. The $\mathcal{A}(\mathcal{R}_t^s)$ is in the form that $\mathcal{A}(\mathcal{R}_t^s) = \{s | \mathcal{A}_l(\mathcal{R}_t^s) < s < \mathcal{A}_u(\mathcal{R}_t^s)\}$, where $\mathcal{A}_l(\mathcal{R}_t^s), \mathcal{A}_u(\mathcal{R}_t^s) \in \mathbb{R}^n$ are two vectors. ($\mathcal{A}(\mathcal{R}_t^s)$ is an interval domain).

All the safety constraints on our benchmark are defined as a rectangle. Thus our following analysis is based on Assumption 1.

Assumption 1 *Given constant lower bound $L \in \mathbb{R}^n$ and constant upper bound $U \in \mathbb{R}^n$ for safe region. The S_u is in the form of*

$$
S_u = \{s \in \mathbb{R}^n | s > U \lor s < L\}
$$

Theorem 4. $\forall \mathcal{T} \in \mathbb{R}^{n \times n}, s \in \mathbb{R}^n$,

$$
\mathcal{T}^{\ge 0} \mathcal{L}(s) + \mathcal{T}^{<0} \mathcal{U}(s) \le \mathcal{T}s \le \mathcal{T}^{\ge 0} \mathcal{U}(s) + \mathcal{T}^{<0} \mathcal{L}(s)
$$

Where $\mathcal{T}_{ij}^{\ge 0} = max(\mathcal{T}_{ij}, 0)$ and $\mathcal{T}_{ij}^{<0} = min(\mathcal{T}_{ij}, 0)$. $\mathcal{L}(s), \mathcal{U}(s)$ is the upper and lower boundary of s respectively.

Proof. Let $s' = \mathcal{T}s$, $s'_i \in \mathbb{R}$ is the i-th element of $s' \in \mathbb{R}^n$,

$$s'_i = \sum_{j=0}^{n-1} \mathcal{T}_{ij} s_j$$

When $\mathcal{T}_{ij} \geq 0$,

$$\forall s_j \in [\mathcal{L}_j(s), \mathcal{U}_j(s)], s'_i \in [\mathcal{T}_{ij}\mathcal{L}_j(s), \mathcal{T}_{ij}\mathcal{U}_j(s)]$$

When $\mathcal{T}_{ij} < 0$,

$$\forall s_j \in [\mathcal{L}_j(s), \mathcal{U}_j(s)], s'_i \in [\mathcal{T}_{ij}\mathcal{U}_j(s), \mathcal{T}_{ij}\mathcal{L}_j(s)]$$

For $\mathcal{L}(s')$, when $\mathcal{T}_{ij} \geq 0$, $\mathcal{T}_{ij}s_j \geq \mathcal{T}_{ij}\mathcal{L}_j(s)$; when $\mathcal{T}_{ij} < 0$, $\mathcal{T}_{ij}s_j \geq \mathcal{T}_{ij}\mathcal{U}_j(s)$, as a result,

$$\mathcal{T}s \geq \mathcal{T}^{\geq 0}\mathcal{L}(s) + \mathcal{T}^{<0}\mathcal{U}(s)$$

Similarly, for $\mathcal{U}(s')$, when $\mathcal{T}_{ij} \geq 0$, $\mathcal{T}_{ij}s_j \leq \mathcal{T}_{ij}\mathcal{U}_j(s)$; when $\mathcal{T}_{ij} < 0$, $\mathcal{T}_{ij}s_j \leq \mathcal{T}_{ij}\mathcal{L}_j(s)$,

$$\mathcal{T}s \leq \mathcal{T}^{\geq 0}\mathcal{U}(s) + \mathcal{T}^{<0}\mathcal{L}(s)$$

We store $\mathcal{R}_t^s$ as a tuple $(\mathcal{T}_{s_t}, \mathcal{T}_{w_t})$. We can compute its over-approximation $\mathcal{A}(\mathcal{R}_t^s)$ with $\mathcal{T}_{s_t}$ and $\mathcal{T}_{w_t}$. First, we compute the over-approximation for the noise-free reachable set $\mathcal{R}_t$ with $\mathcal{T}_{s_t}$. Supposing the initial state space is $[\mathcal{L}_{s_0}, \mathcal{U}_{s_0}]$, $\mathcal{A}_l(\mathcal{R}_t) = \mathcal{T}_{s_t}^{\geq 0}\mathcal{L}_{s_0} + \mathcal{T}_{s_t}^{<0}\mathcal{U}_{s_0}$, and $\mathcal{A}_u(\mathcal{R}_t) = \mathcal{T}_{s_t}^{\geq 0}\mathcal{U}_{s_0} + \mathcal{T}_{s_t}^{<0}\mathcal{L}_{s_0}$. Supposing the noise is bounded by $[\varepsilon'_l, \varepsilon'_u]$, $\mathcal{W}_t$ is a set containing all the possible states of $\mathcal{T}_{w_t}w$, $\mathcal{A}_l(\mathcal{W}_t) = \mathcal{T}_{w_t}^{\geq 0}\varepsilon'_l + \mathcal{T}_{w_t}^{<0}\varepsilon'_u$, and $\mathcal{A}_u(\mathcal{W}_t) = \mathcal{T}_{w_t}^{\geq 0}\varepsilon'_u + \mathcal{T}_{w_t}^{<0}\varepsilon'_l$. We can compute the $\mathcal{A}(\mathcal{R}_t^s)$ by adding $\mathcal{A}(\mathcal{R}_t)$ and $\mathcal{R}(\mathcal{W}_t)$. Because $\mathcal{A}(\mathcal{R}_t)$ and $\mathcal{R}(\mathcal{W}_t)$ are two intervals,

$$\mathcal{A}_l(\mathcal{R}_t^s) = \mathcal{A}_l(\mathcal{R}_t) + \mathcal{A}_l(\mathcal{W}_t) = \mathcal{T}_{s_t}^{\geq 0}\mathcal{L}_{s_0} + \mathcal{T}_{s_t}^{<0}\mathcal{U}_{s_0} + \mathcal{T}_{w_t}^{\geq 0}\varepsilon'_l + \mathcal{T}_{w_t}^{<0}\varepsilon'_u$$
$$\mathcal{A}_u(\mathcal{R}_t^s) = \mathcal{A}_u(\mathcal{R}_t) + \mathcal{A}_u(\mathcal{W}_t) = \mathcal{T}_{s_t}^{\geq 0}\mathcal{U}_{s_0} + \mathcal{T}_{s_t}^{<0}\mathcal{L}_{s_0} + \mathcal{T}_{w_t}^{\geq 0}\varepsilon'_u + \mathcal{T}_{w_t}^{<0}\varepsilon'_l$$

Theorem 5. *Given lower bound $L \in \mathbb{R}^n$ and upper bound $U \in \mathbb{R}^n$ for safe region, $\mathtt{clip}(\cdot, L, U)$ bounds the input between the L and U.*

$$\mathbf{b_1} = \mathcal{A}_u(\mathcal{R}_t^s) - \mathcal{A}_l(\mathcal{R}_t^s)$$
$$\mathbf{b_2} = \mathtt{clip}(\mathcal{A}_u(\mathcal{R}_t^s), L, U) - \mathtt{clip}(\mathcal{A}_l(\mathcal{R}_t^s), L, U)$$
$$\int_{\mathcal{S}_u \cap \mathcal{R}_t^s} ds \leq \prod_{i=0}^{n-1}(\mathbf{b_1})_i - \prod_{i=0}^{n-1}(\mathbf{b_2})_i$$

A illustrating example is in Fig. 6. $\prod_{i=0}^{n-1}(\mathbf{b_1})_i - \prod_{i=0}^{n-1}(\mathbf{b_2})_i$ represents the area of the yellow frame. We do not compute the $\int_{\mathcal{S}_u \cap \mathcal{R}_t^s} ds$ directly, but compute its upper bound with an over-approximation. Such approximation can be useful when the exact reachable set is expensive to compute in the high dimension case. Also, all the operations can be done with simple matrix operations straightforwardly, which are highly optimized on modern software and hardware.

Appendix B.4   Lower Bound of p_t

With the Theorem 3 or Corollary 1, we can get the upper bound U_f of $f_t(s)$. The intersection between the reachable set $\mathcal{R}_t^s$ and the unsafe state set S_u is $S_u \cap \mathcal{R}_t^s$. Then,

$$p_t \geq 1 - U_f \int_{S_u \cap \mathcal{R}_t^s} ds$$

$$U_f = \frac{1}{|\det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}$$

If the noise subjects to uniform distribution,

$$U_f = \min\left(\frac{1}{|\det(\mathcal{T}_{s_t})| \prod_{i=0}^{n-1} \delta_i}, \frac{1}{|\det(\mathcal{T}_{w_t})| \prod_{i=0}^{n-1} \delta_i'}\right)$$

The Theorem 5 gives us that $\int_{\mathcal{S}_u \cap \mathcal{R}_t^s} ds \leq \prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i$. Thus, we get the Theorem 6.

Theorem 6.

$$p_t \geq 1 - U_f \cdot \left(\prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i\right)$$
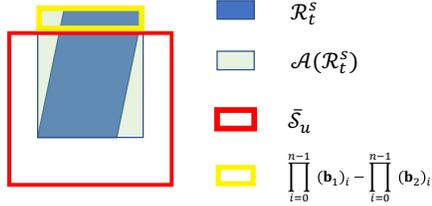

Fig. 6. A demo for Theorem 6. $\mathcal{R}_t^s$ is the reachable set at step t; $\mathcal{A}(\mathcal{R}_t^s)$ is the over-approximation of $\mathcal{R}_s^t$; $\bar{S}_u$ is the safe state space; $\prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i$ is the area of the yellow frame, and it is used as upper bound of $\int_{\mathcal{S}_u \cap \mathcal{R}_t^s} ds$.

In Fig 6, we provide a demo about how the Theorem 6 works. Given a stochastic reachable set $\mathcal{R}_s^t$, we compute the over-approximation $\mathcal{A}(\mathcal{R}_t^s)$. The safety region is defined as $\overline{S}_u$. We can compute the area of $s \in \mathcal{A}(\mathcal{R}_t^s) \cap S_u$ with $\prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i$. Because the probability density function at step t, $f_t(s) \leq U_f$, the cumulative probability for the yellow-wrapped region is upper bounded by $U_f \cdot (\prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i)$. Thus, we can know that the cumulative probability for these safe states is lower bounded by $1 - U_f \cdot (\prod_{i=0}^{n-1} (\mathbf{b_1})_i - \prod_{i=0}^{n-1} (\mathbf{b_2})_i)$.