Safe Exploration in Reinforcement Learning by Reachability Analysis over Learned Models*

Yuning Wang[0009−0000−4317−9758] and He Zhu[0000−0001−9606−150X]

Rutgers University, New Brunswick NJ, USA
{yw895,hz375}@cs.rutgers.edu

Abstract. We introduce VELM, a reinforcement learning (RL) framework grounded in verification principles for safe exploration in unknown environments. VELM ensures that an RL agent systematically explores its environment, adhering to safety properties throughout the learning process. VELM learns environment models as symbolic formulas and conducts formal reachability analysis over the learned models for safety verification. An online shielding layer is then constructed to confine the RL agent's exploration solely within a state space verified as safe in the learned model, thereby bolstering the overall safety profile of the RL system. Our experimental results demonstrate the efficacy of VELM across diverse RL environments, highlighting its capacity to significantly reduce safety violations in comparison to existing safe learning techniques, all without compromising the RL agent's reward performance.

Keywords: Controller Synthesis · Reinforcement Learning · Safety Verification · Safe Exploration.

1 Introduction

Deep reinforcement learning (RL) is a promising approach for synthesizing controllers [19] to govern cyber-physical systems like autonomous vehicles. State-of-the-art RL algorithms can autonomously acquire motor skills through trial and error, either in simulated environments or even in unknown terrains, thus circumventing the need for laborious manual engineering. However, during training in most RL algorithms, agents perform a significant number of exploratory steps that can lead to dangerous behavior. In many real-world scenarios where ensuring high assurance is crucial, it becomes imperative for the RL agent to behave safely during environment interactions, even in training scenarios when the agent is not yet optimal [37,43].

To facilitate safe exploration, it is essential to have a mechanism that determines the safety of executing an action in a given environment state. Several existing approaches utilize prior knowledge about system dynamics [6,52,5] to make such assessments. When the environment dynamics are not known a priori, existing safe RL methods utilize learned predictors in the shape of neural

* This work is supported by the National Science Foundation under grant CCF-2007799.

networks [1,15,47,7] to predict the safety implications of particular control action. Training these neural predictors may require numerous potentially unsafe environment interactions.

There are also model-based safe RL techniques that leverage learned environment models in unknown environments to filter out unsafe actions [4,34]. In the recent CRABS framework [34], a barrier certificate and a model for environment dynamics are co-trained in conjunction with a controller. The learned neural barrier certificate serves as a predictive tool to assess whether a control action from the policy aligns with the safety requirement. In cases where it does not, a safeguard policy, trained on the environment model, is executed. While rooted in formal methods concepts, CRABS cannot rigorously verify the accuracy of a learned barrier certificate. This challenge arises from the fact that both the certificate and the underlying environment models are deep neural networks, making formal verification a complex task. Another recent work SPICE [4] uses weakest preconditions [16] to generate, from a learned environment model, a predicate that decides if an action is safe to take at a current environment state concerning a short time horizon H. However, H cannot be extended to cover the entire horizon of an RL task, primarily because of the inherent challenge in constructing precise weakest precondition transformers for neural networks. As a result, although grounded in Hoare logic, SPICE still suffers from notable safety violations in its environment exploration.

We present VELM, a model-based safe reinforcement learning framework that engages in formally **v**erified safe **e**xploration through **l**earned environment **m**odels, covering the entire horizon of an RL task. VELM learns a symbolic environment model linking the system's future states, past states, and the controller's actions. Most non-linear control systems are characterized by dynamics dictated by mathematical equations involving operators such as trigonometric functions (like sine and cosine) and power functions. By leveraging this prior knowledge of common operators that could appear in environment dynamics, VELM searches a symbolic environment model in the space of interpretable mathematical expressions by symbolic regression techniques. Symbolic regression methods have demonstrated remarkable extrapolation capabilities in recent studies and have proven valuable across diverse domains including physics [10,28,30]. More importantly, unlike neural environment models, symbolic environment models are conducive to long-horizon reachability analysis, enabling the computation of the reachable set of a control system across the entire task horizon. VELM leverages this capability to establish a safe exploration regime for verified safe learning.

VELM can be instantiated on top of any model-based reinforcement learning algorithms. It involves a two-step procedure repeated until convergence: (a) interact with the true environment to collect a dataset of environment transitions and learn from the data an environment state transition model F (i.e. a function that maps current state s_t and action a_t to next state s_{t+1}) and (b) derive a controller π from this learned model. In each learning iteration, VELM aims to ensure that the data collection process of using the current controller π to interact with the true environment in step (a) is safe. One way to do so is by

verifying the safety of π according to the learned model. However, conducting reachability analysis of neural networks in a closed-loop control system remains a challenging research problem [27]. Alternatively, VELM considers π as an oracle and derives a much simpler and verification-friendly *time-varying* linear controllers π' to approximate the policy actions executed by π at each time step within the RL task horizon. While alternative methods such as approximating a neural controller as a polynomial function exist [48], our objective is to achieve a balance between expressiveness and verifiability. Time-varying linear controllers provide computational efficiency for reachability analysis, making verification over learned models feasible in a learning loop. VELM solves a constrained optimization problem aimed at optimizing the behavior of π' to closely match that of π while simultaneously ensuring that π' can be formally verified as safe for the learned environment model. Leveraging π' as a reference, VELM computes a safety shield that restricts the neural policy π to explore solely within the state space where π' is verified as safe in the learned model. The shield intervenes whenever the neural policy π proposes a potentially unsafe control action that could result in a next state outside the safe state space. It then substitutes this action with a safe alternative provided by π'. The environment state transition model F is repeatedly updated during the learning process using data safely collected using the shielded neural controller. The computation of the shield is accordingly repeated, leading to a more refined shield with each update to the controller.

While there exists prior work that explored shielding for safe RL, they require a calibrated piecewise linear dynamics model [52,5] or an abstract model of the agent's safe behavior [24], whereas VELM automatically learns a dynamics model and a safe shielding policy. Adapting these techniques to learned environment models that evolve across training iterations is challenging, given the inherent difficulty of approximating nonlinear models as piecewise linear functions. Compared with SPICE [4], VELM is computationally efficient as it only computes a shield once for a policy while SPICE requires calling a QP (Quadratic Programming) procedure at every timestep.

Across a suite of challenging continuous control benchmarks, VELM exhibits reward performance comparable to fully neural approaches and significantly fewer safety violations during training compared to state-of-the-art safe RL techniques.

In summary, this paper makes the following contributions:

- We propose a novel approach for model-based safe reinforcement learning. Our approach learns an environment model as a symbolic formula and constructs a shielding layer to confine an RL agent to explore within a state space formally verified as safe for the learned model, thereby enhancing the overall safety profile of the RL system.
- We present VELM as an efficient instantiation of this approach. The experiment results show that VELM offers much greater safety than prior model-based safe RL approaches without suffering a loss in reward performance.

2 Problem Setup

Safety Specification. We define a safety specification as a logical formula specifying the safe states of a control system.

Definition 1 (Safety Specification). *A safety specification φ is a quantifier-free Boolean combination of linear inequalities over the environment state variables x:*

$$\langle \varphi \rangle ::= \langle P \rangle \mid \varphi \wedge \varphi \mid \varphi \vee \varphi;$$

$$\langle P \rangle ::= \mathcal{A} \cdot x \leq b \text{ where } \mathcal{A} \in \mathbb{R}^{|x|},\ b \in \mathbb{R};$$

A state $s \in S$ satisfies a safety specification φ, denoted as $s \models \varphi$, iff $\varphi(s)$ is true.
MDP. We formalize an RL system as a Markov decision process (MDP). Specifically, an MDP is a structure $M[\cdot] = (S, A, P, R, S_0, H, \cdot)$ where S is an infinite set of *continuous real-vector* environment states which are valuations of the state variables $x_1, x_2, \ldots, x_n$ of dimension n ($S \subseteq \mathbb{R}^n$), A is a set of *continuous real-vector* control actions which are valuations of the action variables $u_1, u_2, \ldots, u_m$ of dimension m. $R : S \times A \to \mathbb{R}$ is a reward function that returns the immediate reward after the transition from an environment state $s \in S$ with action $a \in A$. $P(s_{t+1} \mid s_t, a_t)$ is an (unknown) probabilistic state transition function where $s_{t+1}, s_t \in S$ and $a_t \in A$ and t is a time step index. S_0 is a set of initial states. H is the time horizon of the control task (i.e. the maximum number of timesteps of a trajectory). An MDP $M[\cdot]$ is parameterized with an (unknown) controller.
Controller (Policy). A controller is a stochastic function $\pi : S \to A$ mapping states to distributions over actions. We explicitly model the deployment of a (learned) controller π in $M[\cdot]$ as a closed-loop system $M[\pi]$. $M[\pi]$ generates trajectories (or rollouts) $\zeta = s_0, a_0, s_1, a_1, \ldots, a_{H-1}, s_H$ where $s_0 \in S_0$, each $a_t \sim \pi(s_t)$, and each $s_{t+1} \sim P(s_t, a_t)$. Given a discount factor $0 \leq \beta < 1$, the long-term reward of a policy π is $R(\pi) = \mathbb{E}_{(\zeta = s_0, a_0, \ldots, s_H) \sim M[\pi]}[\sum_{t=0}^{H} \beta^t R(s_i, a_i)]$.
Problem Formulation. The goal of reinforcement learning is to find a policy $\pi^* = \arg\max_\pi R(\pi)$. To achieve this goal, the learning process of (model-free or model-based) reinforcement learning algorithms progressively refines and optimizes policies $\pi_0, \pi_1, \ldots, \pi_T$ over successive iterations. At each iteration, the current policy is evaluated, and adjustments are made to improve its performance. This learning process continues until the policy converges to the optimal policy π^*. Given a bound δ, we define safe exploration as a learning process $\pi_0, \pi_1, \ldots, \pi_T$ such that

$$\pi_T = \pi^* \text{ and } \forall 1 \leq j \leq T,\ 0 \leq t \leq H.\ P_{\zeta \sim \pi_j, s_t \in \zeta}(\neg \varphi(s_t)) < \delta \tag{1}$$

Essentially, the end goal is for the final policy π_T in the sequence to optimize long-term rewards, while each intermediate policy (excluding π_0) is constrained to a limited probability δ of unsafe behavior. This definition does not place safety constraints on π_0 as the environment dynamics is not known and hence π_0 can exhibit arbitrary (unsafe) behavior.

Algorithm 1 VELM: **V**erified **E**xploration based on **L**earned Models.

1: **procedure** VELM(M, φ)
2: Initialize an empty dataset D and a random NN policy $\pi_{\texttt{NN}}$
3: **for** epoch in $0, \ldots, T$ **do**
4: **if** epoch $= 0$ **then**
5: $\pi_{\mathcal{S}} \leftarrow \lambda s.\lambda t.\ \pi_{\texttt{NN}}(s)$
6: **else**
7: $\pi_{\mathcal{S}} \leftarrow \textsc{Shield}(\hat{M},\ \pi_{\texttt{NN}},\ \varphi)$ ▷ Algorithm 2
8: Unroll real rollouts $\{(s_t, a_t, s_{t+1})\}$ in the real environment M under π_S
9: $D \leftarrow D \cup \{(s_t, a_t, s_{t+1})\}$
10: $\hat{M} \leftarrow \textsc{LearnModel}(D)$
11: Optimize $\pi_{\texttt{NN}}$ using the learned environment $\hat{M}$ via any RL algorithm

3 Verified Exploration through Learned Models

The Main Algorithm. Our overall framework, Verified Exploration through Learned Models (VELM), employs a learned environment model to facilitate safety analysis during the training phase. Akin to existing model-based safe RL techniques [12,5,34,26,4], VELM utilizes the learned environment model to delineate safety regions for the underlying control policy. While VELM can also be applied to safe model-based planning, a policy is in general more efficient than a planner. The primary training procedure is outlined in Algorithm 1. It operates within an *unknown* environment M and takes as input a safety property φ. The algorithm concurrently learns an environment model represented as an MDP $\hat{M}$ and a stochastic neural control policy $\pi_{\texttt{NN}}$[1]. The algorithm maintains a dataset D comprising observed environment transitions, each of which is a tuple of future and past states along with the controller's actions (s_t, a_t, s_{t+1}). This dataset is acquired by interaction with the real environment M (Line 9). Subsequently, VELM utilizes this dataset to learn a symbolic environment model $\hat{M}$ (Line 10) and optimizes the neural policy $\pi_{\texttt{NN}}$ on this learned model via any model-free RL algorithm of the user's choice (Line 11). Notably, VELM uses a shielded policy π_S for exploring the real environment to construct D. π_S takes a state s at a timestep t as input and generates a safe action for the RL agent to execute at t. This is necessary because directly executing the neural controller $\pi_{\texttt{NN}}$ in the real environment M could result in safety violations. The shield policy π_S is constructed in Line 7 via the Shield procedure (Algorithm 2). This procedure leverages reachability analysis on the learned environment model $\hat{M}$ to establish a safe exploration regime covering the entire task horizon. π_S constrains $\pi_{\texttt{NN}}$ to only explore the real environment within the established safe region.

In the following, we describe in detail the procedures to learn symbolic environment models and construct shielded policies for verified safe exploration.

[1] VELM integrates a stochastic policy for exploring the environment to seek high-reward signals. This is not a strict requirement and VELM can also integrate any deterministic policy learning algorithms.

$$\alpha ::= \alpha \ + \ \alpha \mid \alpha \ - \ \alpha \mid \alpha \ \times \ \alpha \mid \alpha \ / \ \alpha \mid \ \sin(\alpha) \mid \ \cos(\alpha) \mid x \mid n$$

Fig. 1: Context-free grammar for defining state-transition functions.

3.1 Symbolic Environment Models

The LearnModel procedure (Line 10 in Algorithm 1) follows the conventional model-based RL framework [25] to learn an environment MDP model $\hat{M}[\cdot] = (S, A, F, R, S_0, H, \cdot)$ where $F : S \times A \to S$ is learned using the dataset D to approximate the unknown probabilistic state transition P in the real environment[2]. VELM distinguishes itself from existing methods by learning a *symbolic* environment state transition function F instead of a deep neural network model.

Given a dataset $D = \{(s_t, a_t, s_{t+1})\}$ of real environment state transitions, the LearnModel procedure learns an approximate model f of the environment's dynamics to fit D:

$$f = \underset{f \in \mathcal{F}_\alpha}{\operatorname{argmax}} \, \mathbb{E}_{(s_t, a_t, s_{t+1}) \in D} \| f(s_t, a_t) - s_{t+1} \| \tag{2}$$

where $\mathcal{F}_\alpha$ is a family of expressions that can be articulated using the grammar outlined in Fig. 1. This grammar accommodates common mathematical operators such as trigonometric functions. The metavariables x and n represent state variables and constants respectively. The symbolic function f establishes the relation between the next state s_{t+1} and the system's past state s_t, as well as the controller's action a_t.

Why symbolic environment models? First, we observe that the dynamics of non-linear control systems often follow mathematical equations. Second, symbolic environment models are suitable for long-horizon reachability analysis to verify the safety of a control system. In contrast, performing reachability analysis over neural network models suffers from large accumulation errors arising from over-approximation [27].

To infer a symbolic formula f to fit D in Equation 2, the LearnModel procedure employs off-the-shelf symbolic regression techniques [10]. Symbolic regression is a machine learning approach that can learn the governing formulas of data. As demonstrated in recent studies [10,28,30], symbolic regression exhibits excellent extrapolation capabilities and has already proved useful in a variety of domains such as physics. VELM uses it to search over the space of mathematical expression by manipulating the operators, constants, and variables in the grammar depicted in Fig. 1.

Nondeterministic Environment Model. It is important to note that VELM does not directly use the deterministic function f as the state transition function F for learned models $\hat{M}[\cdot] = (S, A, F, R, S_0, H, \cdot)$. In cases where control environments are stochastic (common in RL tasks), deterministic state transition functions are not adequate. For stochastic environments, we aim to bound the

[2] If the real reward function is unknown, an approximate reward function $R : S \times A \to \mathbb{R}$ can also be learned from data [25] by recording in $D = \{(s_t, a_t, s_{t+1}, r_t)\}$ the immediate reward r_t of taking an action a_t. We omit this detail in the paper.

deviation between f and the real environment. We identify ϵ such that for all s_t and a_t, $\|f(s_t, a_t) - s_{t+1}\| \leq \epsilon$ where $s_{t+1} \sim P(\cdot|s_t, a_t)$ is sampled from the true environment transition at s_t by taking action a_t. We then express the state transition function of a learned model $\hat{M}[\cdot]$ as a nondeterministic function:

$$F(s_t, a_t) = f(s_t, a_t) + [-\epsilon, \epsilon]$$

When used for simulation, F generates a next state at time step t by adding an error vector uniformly sampled from $[-\epsilon, \epsilon]$ to the result of $f(s_t, a_t)$. When used for reachability analysis and verification, we consider all possible error terms within $[-\epsilon, \epsilon]$ as an overapproximation to account for the worst-case deviation.

In practice, we estimate ϵ from data and choose the most permissible ϵ such that $\forall(s_t, a_t, s_{t+1}) \in D.\ \|f(s_t, a_t) - s_{t+1}\| \leq \epsilon$. *Given* f, with sufficient data in D, the model learning procedure LearnModel returns a model that is close to the actual environment with high probability $1 - \delta_M$. That is, for all $s \in S, a \in A$,

$$\Pr_{s' \sim P(\cdot|s,a)}\left[s' \notin F(s, a)\right] < \delta_M$$

In this paper, we learn F as a discrete dynamics system model. With an Ordinary Differential Equation solver, we can also leverage symbolic regression to learn a more accurate continuous-time dynamics model. This is left for future work.

Example 1. Consider the classic CartPole environment [8]. The system's state is described by $(x, \dot{x}, \theta, \dot{\theta})$ where x (resp. $\dot{x}$) denotes the position (resp. speed) of the cart along the x-axis and θ (resp. $\dot{\theta}$) is the angle (resp. angular velocity) of the pole with respect to the cart. The goal is to balance the pole straight up and bound the deviation of the cart. VELM learns the following equation to describe the state transition function of the system using the Operon [9] symbolic regression tool where u represents the control action (we ignore ϵ for simplicity):

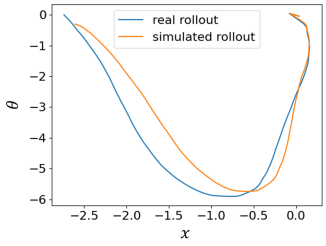

Fig. 2: Executing a random policy on the real CartPole environment and a learned model.

$$x = x + 0.02\dot{x} \qquad \theta = \theta + 0.02\dot{\theta}$$
$$\dot{x} = \dot{x} + 0.019u - (0.001u \cdot \sin(0.999\theta) + 0.001)\sin(\theta) - 0.007\sin(2\theta)$$
$$\dot{\theta} = \dot{\theta} - (0.029u + (0.001\dot{x} + 0.002\dot{\theta}^2 + 0.001\dot{\theta} - 0.02)\sin(\theta))\cos(\theta) + 0.3\cos(\theta - 1.58)$$

Fig. 2 depicts the rollouts in the real environment and simulated in the learned model by executing a random policy from (0,0,0,0). The learned model can reasonably capture real trajectories within a small error bound.

3.2 Shielding for Verified Safe Exploration

With a learned environment model $\hat{M}[\cdot]$, under the assumption of its high-probability approximate accuracy, the verification of a neural controller $\pi_{\texttt{NN}}$ can be directly pursued through reachability analysis over the closed-loop neural

network controlled system $\hat{M}[\pi_{\texttt{NN}}]$ (NNCS). However, the verification of NNCS remains a significant challenge in the research literature [27].

Time-varying Linear Controllers. VELM instead distills a neural controller $\pi_{\texttt{NN}}$ into a time-varying linear controller that is as similar as possible to $\pi_{\texttt{NN}}$. Simultaneously, this process ensures that the safety of the time-varying linear controller can be formally verified concerning the learned model $\hat{M}[\cdot]$ and a safety property φ. Principally, a time-varying linear controller can provide an accurate local approximation of a neural controller at each time step (if the time step is small) and incur a much-reduced verification cost owing to the linearity of the representation. A time-varying linear controller $\pi_\theta(s,t)$ with trainable parameters θ for a time horizon H ($0 \leq t < H$) can be expressed mathematically as:

$$\pi_\theta(s,t) = \theta_k(t)^T \cdot s + \theta_b(t)$$

$\pi_\theta(s,t)$ generates the control input at time t when observing the current environment state s at t. The time-varying nature of the controller is captured by the dependence of the time-varying gain matrix $\theta_k(t)$ and the time-varying bias term $\theta_b(t)$, reflecting the dynamic adjustments in the control strategy over different time instances t. The overall objective of distilling $\pi_{\texttt{NN}}$ into a time-varying linear controller π_θ is:

$$\begin{aligned} \min_\theta \ & \mathbb{E}_{s_0,s_1,\ldots,s_H \sim \hat{M}[\pi_\theta]} \ \|\pi_\theta(s_t,t) - \pi_{\texttt{NN}}(s_t)\|_2 \\ & \text{subject to } \textsc{Verify}(\hat{M}, \pi_\theta, \varphi) = \text{True} \end{aligned} \tag{3}$$

where $\|\cdot\|_2$ is a loss function using the L^2 norm.

Verifying Time-varying Linear Controllers. VELM verifies the safety of a time-varying linear controller π_θ for a learned model $\hat{M}[\cdot]$ using abstract interpretation. While there exist other approaches such as synthesizing barrier certificates for controller verification, the techniques have difficulty handling non-polynomial system dynamics. VELM soundly performs reachability analysis to approximate the set of reachable states of a control system at each timestep:

Definition 2 (Symbolic Rollouts). *Given an environment model $\hat{M}[\pi] = (S, A, F, R, S_0, H, \pi)$ deployed with a controller π, an abstract domain $\mathcal{D}$, an abstract transformer $F^{\mathcal{D}}$ for the state transition function F over $\mathcal{D}$, a symbolic rollout of $M[\pi]$ over $\mathcal{D}$ is $\zeta^{\mathcal{D}} = S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$ where $S_0^{\mathcal{D}} = \alpha(S_0)$ is the abstraction of the initial states S_0 in $\mathcal{D}$ and α is the abstraction function of $\mathcal{D}$. Each symbolic state $S_t^{\mathcal{D}}$ over-approximates the set of reachable states from an initial state in S_0 at timestep t. We have $S_{t+1}^{\mathcal{D}} = F^{\mathcal{D}}\big(S_t^{\mathcal{D}}, A_t^{\mathcal{D}}\big)$ where $A_t^{\mathcal{D}}$ over-approximates the set of actions at t. γ is the concretization function of $\mathcal{D}$ for obtaining the set of concrete states represented by an abstract state $S_t^{\mathcal{D}}$.*

The abstract interpreter $F^{\mathcal{D}}$ in VELM uses Taylor Model (TM) flowpipes as the abstract domain $\mathcal{D}$ to reason about the safety of $\hat{M}[\pi_\theta]$. For reachability analysis of $\hat{M}[\pi_\theta]$ at each timestep t (where $t > 0$), VELM gets the TM flowpipe $S_t^{\mathcal{D}}$ for the reachable set of states of $\hat{M}[\pi_\theta]$ at timestep $t-1$. To obtain a TM representation for the output set of the time-varying linear controller π_θ at timestep t, VELM uses TM arithmetic to evaluate a TM flowpipe $A_t^{\mathcal{D}}$ for

Algorithm 2 Synthesize a shield $\pi_{\mathcal{S}}$ for safe exploration of $\pi_{\texttt{NN}}$. $\pi_{\mathcal{S}}$ intervenes to override potentially unsafe actions by $\pi_{\texttt{NN}}$.

1: **procedure** SHIELD($\hat{M}[\cdot] = \{S, A, F, R, S_0, H, \cdot\}$, $\pi_{\texttt{NN}}$, φ)
2: $\quad\pi_\theta \leftarrow$ APPROXIMATE($\hat{M}[\cdot]$, $\pi_{\texttt{NN}}$, φ) ▷ Algorithm 3
3: $\quad S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_{H-1}^{\mathcal{D}}, S_H^{\mathcal{D}} \leftarrow$ REACHSET($\hat{M}[\pi_\theta]$)
4: $\quad\pi_{\mathcal{S}} \leftarrow \lambda s.\lambda t.$ **let** $a_{\texttt{NN}} = \pi_{\texttt{NN}}(s)$ **in**
5: $\qquad\qquad$ **if** $\exists 0 \leq \boldsymbol{i} \leq t+1.\ F(s, a_{\texttt{NN}}) \subset \gamma(S_{\boldsymbol{i}}^{\mathcal{D}})$ **then** $a_{\texttt{NN}}$
6: $\qquad\qquad$ **else let** $\boldsymbol{i} = \max\left(\{\boldsymbol{i} \mid s \in \gamma(S_{\boldsymbol{i}}^{\mathcal{D}})\}\right)$ **in** $\pi_\theta(s, \boldsymbol{i})$
7: $\quad$ **return** $\pi_{\mathcal{S}}$

$\pi_\theta(s,t) = \theta_k(t)^T \cdot s + \theta_b(t)$ for all states $s \in S_t^{\mathcal{D}}$. The resulting TM representation $A_t^{\mathcal{D}}$ can be viewed as an overapproximation of the controller's output at timestep t. Finally, we use Flow* [11] to construct the TM flowpipe overapproximation $S_{t+1}^{\mathcal{D}}$ for all reachable states at timestep t by reachability analysis over the state transition function $F^{\mathcal{D}}(S_t^{\mathcal{D}}, A_t^{\mathcal{D}})$. To verify $\hat{M}(\pi_\theta)$ against a safety property φ, VELM uses Flow* to check if for each abstract state $S_t^{\mathcal{D}}$ in the symbolic rollout of $\hat{M}(\pi_\theta)$, the concretized states in $\gamma(S_t^{\mathcal{D}})$ does not violate φ.

Verified Shielding. The safety of a distilled controller π_θ does not imply its oracle neural controller $\pi_{\texttt{NN}}$ is safe. For safe exploration using $\pi_{\texttt{NN}}$, VELM constructs a shield for $\pi_{\texttt{NN}}$ based on π_θ. The high-level algorithm for shield synthesis is presented in Algorithm 2.

Given a learned environment model $\hat{M}[\cdot]$, a neural controller $\pi_{\texttt{NN}}$, and a safety specification φ, at Line 2, Algorithm 2 invokes APPROXIMATE to construct a distillation of $\pi_{\texttt{NN}}$ as a time-varying linear controller π_θ. We describe APPROXIMATE in detail in Algorithm 3 and Sec. 3.3. At Line 3, Algorithm 2 uses the symbolic rollout $\zeta^{\mathcal{D}} = S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$ of $\hat{M}[\pi_\theta]$ to derive the reachable set of states of π_θ for the learned environment model $\hat{M}[\cdot]$. As this reachable set of states has been verified safe for $\hat{M}[\pi_\theta]$, the shield constrains $\pi_{\texttt{NN}}$ to only explore within the reachable set $\cup_{0 \leq i \leq H} \gamma(S_i^{\mathcal{D}})$ to remain safe. Algorithm 2 returns a shield $\pi_{\mathcal{S}}$ for $\pi_{\texttt{NN}}$ in the form of a lambda function that takes an environment state s_t at time step t and t as inputs. We show that assuming the learning model soundly approximates the unknown state transition distribution P of the real environment (Sec. 3.1), the shield is provably safe in the following lemma.

Lemma 1. *Assume a learned environment model $\hat{M}[\cdot] = \{S, A, F, R, S_0, H, \cdot\}$ is a sound nondeterministic approximation of the true environment: $\forall s \in S, a \in A.\ s' \sim P(\cdot|s,a) \Rightarrow s' \in F(s,a)$. Given a safety property φ, a neural policy π_{NN}, and its shield $\pi_{\mathcal{S}} =$ SHIELD($\hat{M}[\cdot]$, π_{NN}, φ), for any rollouts $s_0, a_0, s_1, \ldots, s_H$ collected by π_S in the true environment where $s_0 \in S_0$, $a_t = \pi_{\mathcal{S}}(s_t, t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$, we have $s_t \models \varphi$ (i.e. s_t is safe) for all $0 \leq t \leq H$.*

Proof. Since $\pi_{\mathcal{S}} =$ SHIELD($\hat{M}[\cdot]$, $\pi_{\texttt{NN}}$, φ), there exists a π_θ (Line 2 in Algorithm 2) whose symbolic rollouts $S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$ can be verified safe with respect to φ (Line 3 of Algorithm 2). We show that for all $0 \leq t \leq H$, we have $\bigvee_{0 \leq i \leq t} s_t \in \gamma(S_i^{\mathcal{D}})$. This invariant implies that s_t is safe. We prove the invariant

by induction. When $t = 0$, the invariant holds as $s_0 \in \gamma(S_0^{\mathcal{D}})$ by construction. Given an s_t that satisfied the invariant, if $\exists 0 \leq i \leq t+1$. $F(s_t, \pi_{\texttt{NN}}(s_t)) \subset \gamma(S_i^{\mathcal{D}})$ (Line 5), then $a_t = \pi_{\text{NN}}(s_t)$ and by assumption $s_{t+1} \sim P(\cdot|s_t, a_t) \in F(s_t, a_t) \subset \gamma(S_i^{\mathcal{D}})$, which means the invariant holds on s_{t+1} in this case. Otherwise (Line 6), $a_t = \pi_\theta(s_t, i)$ where $i = \max\left(\{i \mid s_t \in \gamma(S_i^{\mathcal{D}})\right)$. Such i must exist as we assume s_t satisfied the invariant. Since $s_{t+1} \sim P(\cdot|s_t, a_t) \in F(s_t, a_t)$ and the soundness of the abstract interpreter $F^{\mathcal{D}}$ ensures that if $s_t \in \gamma(S_i^{\mathcal{D}})$, then $F(s_t, a_t) \subseteq \gamma(S_{i+1}^{\mathcal{D}})$, which means the invariant holds on s_{t+1} in this case as well. By induction, the invariant is true for all $0 \leq t \leq H$.

Theorem 1 (Shield (Algorithm 2) is probabilistically safe). *For a learned environment model $\hat{M}[\cdot] = \{S, A, F, R, S_0, H, \cdot\}$, let δ_M be the probability bound of the model: $\Pr_{s' \sim P(\cdot|s,a)}\left[s' \notin F(s,a)\right] < \delta_M$. Given a safety property φ, a neural policy $\pi_{\texttt{NN}}$, and its shield $\pi_{\mathcal{S}} = \textsc{Shield}(\hat{M}[\cdot], \pi_{\texttt{NN}}, \varphi)$, for any rollouts $s_0, a_0, s_1, \ldots, s_H$ collected by π_S in the true environment where $s_0 \in S_0$, $a_t = \pi_{\mathcal{S}}(s_t, t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$, we have $s_t \models \varphi$ (i.e. s_t is safe) with probability at least $(1-\delta_M)^t$ for all $0 \leq t \leq H$.*

Proof. By Lemma 1, if $s_{t+1} \in F(s_t, a_t)$, then s_{t+1} is safe for all $0 \leq t < H$. By assumption, at each time step, we have $s_{t+1} \in F(s_t, a_t)$ with probability at least $1-\delta_M$. After t time steps, the probability that the assumption is valid is at least $(1-\delta_M)^t$, which means that s_t is safe with probability at least $(1-\delta_M)^t$.

We can relate the probability guarantee in Theorem 1 with our overall objective in Equation 1 by bounding $\delta_M < 1 - (1-\delta)/\exp(H)$. This theorem illustrates that VELM only allows for safety violations when there's an inaccuracy in the environment model. In contrast, existing approaches to safe exploration are susceptible to safety violations stemming from both modeling inaccuracies and actions that are not safe even considering the environment model. For example, SPICE [4] applies weakest precondition generation from safety constraints to a linearization of the learned environment model to determine safe control actions. However, this linearization process introduces substantial approximation errors, compromising the safety of the computed actions on the learned environment model. CRABS [34] uses neural networks for representing environment models and control barrier certificates to identify safe exploration regions. However, a neural control barrier certificate may converge to a suboptimal model and CRABS does not have a procedure to rigorously guarantee its correctness. This may result in delayed or absent intervention for unsafe behaviors.

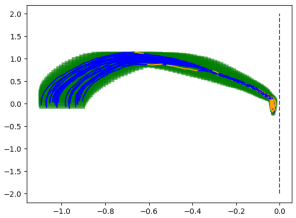

Fig. 3: Executing a shielded neural policy in ACC. The green region denotes the safe space verified on a learned model. The yellow regions denote the control steps where intervention takes place.

Example 2. Consider an adaptive cruise control (ACC) system [5]. The goal is to control an ego car to closely follow a lead car without collision. The lead car can apply acceleration to itself at any time. Fig. 3 shows the rollouts (blue) of a *shielded* neural controller $\pi_{\texttt{NN}}$ in the real environment. The x-axis shows the distance to the lead car while the y-axis shows the relative velocities of the two cars. The rollouts start by accelerating to close the gap with the lead car and subsequently decelerating to prevent a collision. The green region denotes the reachable set of a distilled time-varying linear controller π_θ verified as safe on a learned model of the ACC environment. The yellow regions indicate interventions where π_θ constrains $\pi_{\texttt{NN}}$ to stay within the safe region. Without such intervention, the neural controller alone would fail to decelerate rapidly enough and crash into the lead car (the dashed line on the right side). At times, π_θ needs to intervene well before the final steps to ensure the feasibility of avoiding a crash later.

3.3 Neural Controller Approximation

This section formalizes the APPROXIMATE procedure invoked by Algorithm 2 (Line 2) for distilling a neural controller $\pi_{\texttt{NN}}$ to a time-varying linear controller π_θ that can be verified safe according to a learned environment model.

Minimizing the gap between π_θ and a (fixed) neural controller $\pi_{\texttt{NN}}$ as two functions can be straightforwardly achieved by optimizing θ through gradient descent. However, a binary verification result (true or false) does not offer guidance on how θ should be optimized to ensure that π_θ can be verified safe. Following previous research [45], when facing verification failures, our approach utilizes verification feedback, indicating the extent of safety violations, to guide the optimization process for π_θ. We first formalize the concept of safety violation within the concrete environment state space and then lift it to abstract state spaces.

Definition 3 (State Safety Loss Function). *For a safety specification φ over states $s \in S$, we define a non-negative loss function $\mathcal{L}(s, \varphi)$ such that $\mathcal{L}(s, \varphi) = 0$ iff s satisfies φ, i.e. $s \models \varphi$. We define $\mathcal{L}(s, \varphi)$ recursively, based on the possible shapes of φ (Definition 1):*

- $\mathcal{L}(s, \mathcal{A} \cdot x \leq b) := \max(\mathcal{A} \cdot s - b, 0)$
- $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) := \max(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$
- $\mathcal{L}(s, \varphi_1 \vee \varphi_2) := \min(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$

Notice that $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) = 0$ iff $\mathcal{L}(s, \varphi_1) = 0$ and $\mathcal{L}(s, \varphi_2) = 0$, and similarly $\mathcal{L}(\varphi_1 \vee \varphi_2) = 0$ iff $\mathcal{L}(\varphi_1) = 0$ or $L(\varphi_2) = 0$.

We extend the safety loss definition (Definition 3) to the abstract state space employed in a verification procedure.

Definition 4 (Abstract State Safety Loss Function). *Given an abstract state $S^{\mathcal{D}}$ and a safety specification φ, we define an abstract safety loss function:*

$$\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi) = \max_{s \in \gamma(S^{\mathcal{D}})} \mathcal{L}(s, \varphi)$$

It quantifies the worst-case safety loss of φ across all concrete states encompassed by $S^{\mathcal{D}}$. For an abstract domain $\mathcal{D}$, we typically can approximate the concretization of an abstract state $\gamma(S^{\mathcal{D}})$ using a tight interval $\gamma_I(S^{\mathcal{D}})$. For example, it is straightforward to represent Taylor model flowpipes as intervals in Flow. Based on the potential structure of φ, we redefine $\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi)$ as:*

- $\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \mathcal{A} \cdot x \leq b) := \max_{s \in \gamma_I(S^{\mathcal{D}})} \big(\max(\mathcal{A} \cdot s - b, 0)\big)$
- $\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_1 \wedge \varphi_2) := \max(\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_1), \mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_2))$
- $\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_1 \vee \varphi_2) := \min(\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_1), \mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi_2))$

By definition, we have $\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}, \varphi) = 0 \implies \forall s \in \gamma_I(S^{\mathcal{D}}).\ s \models \varphi$.

We further lift the definition of safety loss over abstract states (Definition 4) to the symbolic rollout of an MDP (Definition 2).

Definition 5 (Symbolic Rollout Safety Loss). *Given an environment model $\hat{M}[\pi_\theta]$ and a safety specification φ, assuming the symbolic rollout of $\hat{M}[\pi_\theta]$ over an abstract domain $\mathcal{D}$ is $\zeta^{\mathcal{D}}_{0:H} = S^{\mathcal{D}}_0, \ldots, S^{\mathcal{D}}_H$, we define an abstract safety loss function to measure the degree to which φ is violated by $\hat{M}[\pi_\theta]$:*

$$\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta],\ \varphi)\ =\ \mathcal{L}_{\mathcal{D}}(\zeta_{0:H},\ \varphi)\ =\ \max_{0 \leq i \leq H}(\mathcal{L}_{\mathcal{D}}(S^{\mathcal{D}}_i, \varphi))$$

Definition 5 enables a quantitative metric for the safety loss of a controller π_θ in the abstract state space of a safety verifier. By definition, we have

$$\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi) = 0 \implies \hat{M}[\pi_\theta] \models \varphi.$$

We rewrite the overall objective of distilling a neural controller $\pi_{\texttt{NN}}$ into a time-varying linear controller π_θ in Equation 3 as:

$$\begin{aligned} &\min_\theta \mathbb{E}_{s_0, s_1, \ldots, s_H \sim \hat{M}[\pi_\theta]}\ \|\pi_\theta(s_t, t) - \pi_{\texttt{NN}}(s_t)\|_2 \\ &\text{subject to } \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi) = 0 \end{aligned} \tag{4}$$

The objective described in Equation 4 frames a constraint optimization problem. To address this, we employ Lagrangian optimization, which provides a principled way to seamlessly incorporate the verification constraint ($\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi) = 0$) into the distillation objective. We introduce a Lagrangian function that incorporates a Lagrange multiplier λ to account for constraint violation and transform Equation 4 into an unconstrained optimization problem:

$$L(\theta, \lambda) = \mathcal{L}_S(\pi_\theta, \pi_{\texttt{NN}}) + \lambda \cdot \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)$$

where $\mathcal{L}_S(\pi_\theta, \pi_{\texttt{NN}}) = \mathbb{E}_{s_0, s_1, \ldots, s_H \sim \hat{M}[\pi_\theta]}\ \|\pi_\theta(s_t, t) - \pi_{\texttt{NN}}(s_t)\|_2$. VELM seeks to optimize the primal parameter θ and the Lagrange multiplier λ to minimize the function $L(\theta, \lambda)$, effectively reducing both the L^2 loss for the distillation objective and safety violations for the verification constraint.

Algorithm 3 outlines the procedure for distilling $\pi_{\texttt{NN}}$ to π_θ. It iteratively performs the following two gradient-based update rules to minimize $L(\theta, \lambda)$:

$$\begin{aligned} \theta &\leftarrow \theta - \eta_\theta \cdot \big(\nabla_\theta \mathcal{L}_S(\pi_\theta, \pi_{\texttt{NN}}) + \lambda \cdot \nabla_\theta \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)\big) \\ \lambda &\leftarrow \lambda + \eta_\lambda \cdot \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi) \end{aligned}$$

where η_θ is a learning rate for θ and η_λ is a learning rate for λ. The Lagrange multiplier λ is increased during the optimization process to penalize deviations

Algorithm 3 Approximate a neural control policy $\pi_{\texttt{NN}}$ with a time-varying linear controller π_θ while ensuring π_θ is formally verified safe for the learned model $\hat{M}$.

1: **procedure** APPROXIMATE($\hat{M}[\cdot] = \{S, A, F, R, S_0, H, \cdot\},\ \pi_{\texttt{NN}},\ \varphi$)
2: Initialize a time-varying linear policy π_θ over H timesteps
3: $\theta \leftarrow$ all parameters in π_θ for optimization
4: **while** `true` **do**
5: $\ell_S \leftarrow \mathcal{L}_S(\pi_\theta, \pi_{\texttt{NN}})$
6: $\ell_{\mathcal{D}} \leftarrow \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)$
7: **if** $\ell_{\mathcal{D}} = 0$ and ℓ_S converges **then**
8: return π_θ
9: $\theta \leftarrow \theta - \eta_\theta \cdot \big(\nabla_\theta \mathcal{L}_S(\pi_\theta, \pi_{\texttt{NN}}) + \lambda \cdot \nabla_\theta \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)\big)$
10: $\lambda \leftarrow \lambda + \eta_\lambda \cdot \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)$

from satisfying the verification constraint. As such, even though the verification procedure may introduce approximation errors, VELM can reduce this error by conducting optimization in the abstract state space [45]. VELM repeats the iterative update until the distillation loss (ℓ_S) converges and the safety violation loss (ℓ_D) converges to 0 (Line 8).

Gradient Estimation for $\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)$. Deriving the gradients of the verification constraint $\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi)$ directly poses a challenge, as it requires the verification procedure to be differentiable, a feature not practical. To address this obstacle, following prior research [45], VELM estimates the gradients of $\mathcal{L}_{\mathcal{D}}$ through random search [36]. In each training iteration, given a closed-loop environment $\hat{M}[\pi_\theta]$, we generate perturbed systems $\hat{M}[\pi_{\theta+\nu\omega}]$ and $\hat{M}[\pi_{\theta-\nu\omega}]$ by introducing sampled Gaussian noise ω to the current controller π_θ's parameters θ in both directions. Here, ν represents a small positive real number. By assessing the abstract safety losses of the symbolic rollouts for $\hat{M}[\pi_{\theta+\nu\omega}]$ and $\hat{M}[\pi_{\theta-\nu\omega}]$, we update θ using a finite difference approximation along an unbiased estimator of the gradient:

$$\nabla_\theta \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_\theta], \varphi) \leftarrow \frac{1}{N} \sum_{k=1}^{N} \frac{\Big(\mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_{\theta+\nu\omega_k}],\ \varphi) - \mathcal{L}_{\mathcal{D}}(\hat{M}[\pi_{\theta-\nu\omega_k}], \varphi)\Big)}{\nu} \omega_k$$

Performance Guarantees. We conclude the technical section by discussing the reward performance of VELM. One important concern is whether shielding a neural control policy hinders the RL algorithm's ability to learn the optimal policy. Previous studies [44,5,4] have established the following regret bound concerning the reward performance of a shielded policy for safe exploration compared to the optimal policy that does not seek to restrict safety violations during the learning process. Let $\pi_S^i = \text{SHIELD}(\hat{M}^i[\cdot], \pi_{\texttt{NN}}^i, \varphi)$ for $1 \leq i \leq T$ be a sequence of policies learned in Algorithm 1 where φ is the safety property, $\hat{M}^i[\cdot]$ and $\pi_{\texttt{NN}}^i$ are the learned environment model and neural controller at the i^{th} iteration. Introduce a safety indicator Z that takes the value 1 when $\pi_S^i(s,t) = \pi_{\texttt{NN}}^i(s)$ and 0 otherwise, and let $\xi = \mathbb{E}[1 - Z]$ be the frequency with which π_S^i intervenes

in neural policy controls. Assume the reward function is Lipschitz on the controller parameter space and let L_R be the corresponding Lipschitz constant. Let β and τ^2 be the bias and variance in the gradient estimate that is incurred due to sampling. Let $\epsilon_{\mathcal{S}}$ be an upper bound on the imprecision incurred by distilling $\pi^i_{\texttt{NN}}$ to a linear time-varying controller. Let ϵ_m be an upper bound for the Kullback-Leibler divergence between the learned environment model and the true environment dynamics at all time steps. Let ϵ_π be an upper bound on the total variational divergence between the policy used to gather data and the policy being trained at all time steps. Set the learning rate η of the RL algorithm for updating $\pi^i_{\texttt{NN}}$ as $\sqrt{\frac{1}{\tau^2}(\frac{1}{T}+\epsilon_{\mathcal{S}})}$. Assuming π^* is the (unknown) the optimal safe control policy, we have the following regret bound [44,5,4] for Algorithm 1: $R(\pi^*) - \mathbb{E}\big[\frac{1}{T}\sum_{i=1}^{T} R(\pi^i_S)\big] = O\big(\sqrt{\frac{1}{\tau^2}(\frac{1}{T}+\epsilon_{\mathcal{S}})} + \beta + L_R \cdot \xi + \epsilon_m + \epsilon_\pi\big)$. VELM does not impose a significant penalty on the agent's reward performance for achieving safety as the regret bound becomes tighter when the frequency of interventions in the decision of the neural controllers ξ decreases during training. As the environment model improves during training (i.e. ϵ_m and ϵ_π decrease), the controller converges to higher rewards. The remaining terms are associated with the standard error by using sampling to approximate policy update gradients.

4 Experiments

In our implementation of VELM [3], we use SAC [23], a state-of-the-art reinforcement learning algorithm, as the base algorithm to optimize neural network controllers. We build the abstract interpreter for reachability analysis of a time-varying linear controller against a learned model on top of Flow* [11] for reasoning about nonlinear state transition functions. We use Operon [9] to learn a symbolic environment model for the LearnModel procedure at Line 10 in Algorithm 1. In the implementation, we invoke the LearnModel procedure only when the existing environment model is invalid for the newly collected trajectories from the real environment. Recall that our learned model is nondeterministic (Sec. 3.1). Given a current state, it outputs a range for the next state. If the actual next state is not within the range, we consider the model invalid (i.e., $\exists t.s_{t+1} \notin F(s_t, a_t)$). This strategy significantly accelerates the learning process.

Baselines. We compared VELM with three baselines: SAC, SPICE [4], and MBPPO-Lagrangian [26]. The SAC baseline acts as an upper bound on reward performance since the agent does not need to explicitly handle safety constraints. The other safe RL baselines are relevant because they are all model-based as VELM. However, they all use neural networks for learning environment state transition dynamics. SPICE applies weakest precondition generation from safety constraints to a linearized form of learned environment models to ascertain safe control actions for shielding. The linearization step may introduce approximation errors. MBPPO-Lagrangian finds a safety-constraint-satisfying policy by

[3] VELM is available at https://github.com/RU-Automated-Reasoning-Group/VELM

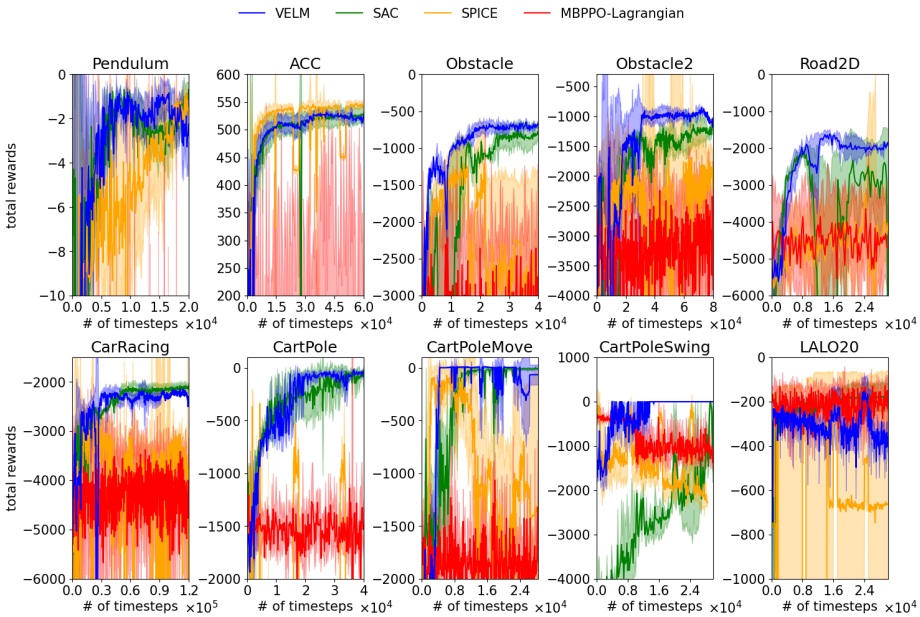

Fig. 4: Rewards for all the tools throughout the training phase. The solid curve represents the mean across 5 random seeds. The shaded area indicates the standard deviation.

using the Lagrangian method to reduce the cumulative safety violations throughout trajectories executed on the learned model. This method does not consider shielding to ensure safe exploration. We also tried to use CRABS [34] as another model-based safe-learning baseline. In addition to a neural environment model, CRABS uses another neural network to learn a control barrier certificate to identify a safe region on the neural environment model for shielding. However, we found that CRABS is excessively time-consuming to execute, completing only an average of 10 episodes within a day. Therefore, we have excluded CRABS in the results presented in this section. In summary, these baselines suffer from safety violations stemming from both (1) environment modeling imprecision and (2) control policies that are not safe even considering the environment model. VELM eliminates the second source of errors. Our experiments aim to answer the question - *How does the performance of VELM compare to representative baseline approaches, considering metrics such as rewards, number of unsafe steps, and overall efficiency?*

Benchmarks. We used the benchmarks considered in related work. Pendulum, ACC, Obstacle, Obstacle2, Road2D, and CarRacing are taken from the SPICE benchmarks [5,4]. In Road2D, an autonomous vehicle is controlled to reach a designated destination while adhering to a specified speed limit. Obstacle and Obstacle2 pose a challenge for a 2D robot to reach a specified goal while avoiding

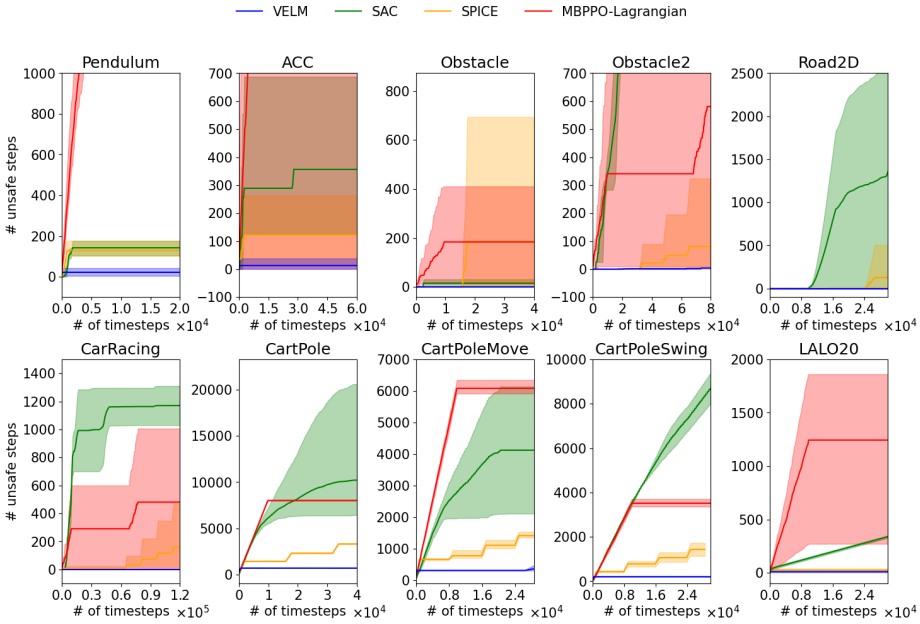

Fig. 5: Cumulative safety violations for all the tools throughout the training phase. The solid curve represents the mean across 5 random seeds. The shaded area indicates the range between the minimum and maximum values.

an obstacle. In Obstacle, the obstruction is positioned to the side, affecting the agent only during exploration, without cutting the shortest path to the goal. In Obstacle2, the obstruction is placed between the initial region and the goal region, requiring the learned controller to navigate around it. In the Pendulum task, the objective is to maintain a pendulum in the upright position. The goal of ACC (adaptive cruise control) is to closely follow a leading vehicle without collision, with the lead car selecting acceleration randomly from a truncated normal distribution at each time step. The CarRacing environment is similar to Obstacle2 but the goal is to reach a goal region on the opposite side of the obstacle and then return to the initial region. This requires the agent to complete a loop around the obstacle to fulfill the objective. Cartpole is from Open AI Gym [8]. The nonlinear benchmarks CartPoleMove and CartPoleSwing are taken from CRABS [34]. The CartPoleMove task is challenging as high-reward policies must carefully explore near the safety boundary. The user-specified safety set is $\{(x, \theta) : |\theta| \leq \theta_{\texttt{max}} = 0.2, |x| \leq 0.9\}$ where x is the cart horizontal position and θ is the pole angle. $\theta_{\texttt{max}}$ corresponds to approximately 11 degrees. The reward function of the task is $r(s, a) = x^2$. Consequently, the optimal policy must delicately move the cart and pole toward the boundary of unsafe regions but remain safe. Similarly, the CartPoleSwing task is also high-risk, high-reward environment. The reward function is $r(s, a) = \theta^2$ and the user-specified safety set

is $\{(x, \theta) : |\theta| \leq \theta_{\texttt{max}} = 1.5, |x| \leq 0.9\}$. So the optimal policy will swing back and forth to some degree close to 90° but prevent the pole from falling. LALO20 is a challenging 7-dimensional nonlinear benchmark modeling a molecular network taken from prior work [48]. This task is difficult because the initial states are situated near the boundary of the unsafe region.

Results. We report the mean reward performance of the learned controllers as well as cumulative safety violations over time during training of each benchmark for VELM and each baseline in Fig. 4 and Fig. 5. The shield intervention rates of VELM and SPICE are listed in Table 1. These results are averaged over 5 random seeds.

Table 1: Comparison of Shield Intervention Rates between VELM and SPICE.

Benchmarks	**VELM**	**SPICE**
Pendulum	0.07	0.00
ACC	0.23	0.78
Obstacle	0.11	0.70
Obstacle2	0.26	0.34
Road2D	0.03	0.18
CarRacing	0.17	0.44
CartPole	0.12	0.81
CartPoleMove	0.16	0.55
CartPoleSwing	0.01	0.78
LALO20	0.49	0.79

Fig. 5 demonstrates that VELM exhibits superior safety performance as it experiences a significantly lower frequency of unsafe steps compared to the baseline methods. Except for the initial controller π_0, the controllers learned by VELM demonstrate nearly zero safety violations when interacting with the real environment in training. SPICE accumulates safety violations more quickly compared to VELM. Overall, VELM achieves a 99.7% reduction in unsafe steps compared to SPICE. SPICE incurs significantly more safety violations in highly nonlinear environments such as CartPole. This suggests that the model linearization step in SPICE introduces significant approximation errors, resulting in either unnecessary interventions or a lack of intervention when there is truly unsafe behavior. This kind of approximation error also limits SPICE to use a bounded-time analysis to determine potential safety violations within the next few time steps (5, as recommended in SPICE [4]). VELM instead can predict the long-term safety of an action far into the future. For example, on CartPole, the average shield intervention rate for SPICE over all the rollouts in the real environment is 81%, while VELM only has an intervention rate of 12%. Similarly, VELM is safer than MBPPO-Lagrangian in every benchmark. As can be seen from Fig. 4, MBPPO-Lagrangian continues to violate the safety property more over time than VELM. Principally, in contrast to VELM, MBPPO-Lagrangian seeks to limit safety violations in expectation and does not assure safety for all visited states.

Fig. 4 also demonstrates that in most cases, VELM attains comparable (or slightly superior) reward performance to SAC. SPICE imposes a substantial penalty on reward performance compared to SAC. This is because SPICE in general exhibits significantly higher shield intervention rates than VELM. As discussed in the performance guarantee analysis in Section 3, frequent shield interventions hinder the RL algorithm from converging to the optimal policy.

Table 2: Training time in seconds for model, network and shield updates

Benchmarks	**Model (s)**	**Network (s)**	**Shield (s)**
Pendulum	6.4	184.0	17.1
ACC	489.9	616.4	57.0
Obstacle	18.6	657.0	33.3
Obstacle2	47.7	661.8	436.5
Road2D	19.8	884.0	62.0
CarRacing	41.2	648.3	209.4
CartPole	21.0	577.5	176.8
CartPoleMove	13.3	302.1	384.5
CartPoleSwing	13.2	311.1	10.5
LALO20	47.8	302.3	808.1

LALO20 is the only benchmark that VELM does not achieve a comparable reward performance to SAC. This is because, in this benchmark, the average shield intervention rate for VELM over all the rollouts in the real environment is relatively high at 49%. However, VELM achieves nearly 0 safety violations during learning. The modest performance penalty is an acceptable trade-off for safety. Although SPICE also achieves almost 0 safety violations on LALO20, its shield intervention rate is 79%, preventing the neural policy from achieving high reward performance.

We present the execution times for each component of VELM across all benchmarks in Table 2, averaged over five random seeds. The Network column in the table reports the time spent training a neural network controller using the base RL algorithm. The Model and Shield columns report the time spent on learning a symbolic environment model and constructing a formally verified shield, respectively. On average, VELM dedicates approximately 9% of its execution time to model learning and 28% to shield construction. This modest overhead is justified by the substantial safety guarantees provided.

5 Related Work

Prior Safe RL works consider constrained Markov decision processes (CMDP), where observed safety violations should be bounded. Lagrangian methods are widely used to solve CMDP with the Lagrangian multiplier controlled adaptively [41] or by PID [40]. Trust region methods [1,51,47] project a current control policy to a feasible safe space around the current policy in each learning iteration. The goal is to bound the number of safety violations under a threshold *in expectation*, while VELM aims to ensure safety for all visited states. Combining these methods with learning a dynamics model can further improve their data efficiency [50,26]. There exist works that learn conservative safety critics to underestimate the long-term safety cost of taking a particular action in a particular state and use the conservative safety critics for safe exploration and

policy optimization [7,46,49]. However, training neural safety critics models may require numerous potentially unsafe environment interactions. VELM instead uses symbolic reachability analysis over learned environment models to identify safe regions of the state space. Other approaches involve pre-training a policy in a simpler environment and fine-tuning it in a more challenging setting [39], or leveraging existing offline data and co-training a recovery policy [42]. Integrating VELM with pretraining and offline data is an interesting avenue for future research.

Another research direction explores Lyapunov functions and barrier certificates. The work in [6] uses Lyapunov functions to identify policy attraction regions where safe operation is guaranteed for discretized deterministic control systems, provided that certain Lipschitz continuity conditions hold. However, this method requires access to system dynamics models. Additionally, a neural network controller may not exhibit Lipschitz continuity with a reasonable coefficient. In [13], it is shown that Lyapunov functions can be co-learned with controllers for discrete action spaces. This work was extended to continuous action spaces by utilizing the Deterministic Policy Gradient theorem [38]. The work by Chow et al. [14] projects control actions to guarantee a decrease in the Lyapunov function after each timestep. In contrast, Donti et al. [17] construct sets of stabilizing actions using a Lyapunov function and then project actions onto this set. A handcrafted barrier function is leveraged in [12] to secure safe exploration in reinforcement learning. A line of research, exemplified by a prior study [48], focuses on verifying an RL controller upon convergence against safety and reachability properties by inferring barrier certificates but does not address safety during training. Combining VELM with such work is promising for future investigation.

Model-based safe reinforcement learning approaches ensure the safety of an RL agent through a model of its environment. When a pre-established model of environmental dynamics is available, a safety shield and a backup controller can be constructed from the model using formal methods to regulate agent behavior [3]. To enforce the safety of a deep neural network controller, the backup controller is run in tandem with the neural controller [2,20,21,22,52,5,31,29,18,24]. Whenever the neural controller is about to leave the provably safe state space governed by the backup controller, the backup controller overrides the potentially unsafe neural actions to enforce the neural controller to stay within the certified safe space. When environment dynamics models are not known a priori, several works [33,32,35,26] maintain a learned environment model and employ various statistical techniques to devise a policy that is likely to be safe according to the model. This gives rise to two sources of unsafe conduct: the policy may exhibit unsafety in relation to the model, or the model could provide an imprecise depiction of the environment. VELM addresses the first source of error by assuring control policies are safe within the confines of an environment model. REVEL [5] involves an iterative learning approach where a neural policy is trained, potentially resulting in unsafety. Subsequently, the learned neural policy is distilled into a piecewise linear policy. Automatic verification is then

applied to certify the piecewise linear policy, a process akin to constructing a barrier function. First, this certification method assumes a calibrated dynamics model, whereas VELM, in contrast, learns the dynamics model. Second, the verification algorithm in REVEL requires a piecewise linear environment model to be manually constructed to approximate the calibrated dynamics model, a condition not practical in VELM (learned environment models evolve across learning iterations in VELM). CRABS [34] iteratively learns a barrier certificate, a dynamics model, and a control policy where the barrier certificate, learned via adversarial training, ensures the policy's safety assuming the learned dynamics model. Yet, formally verifying the correctness of the barrier certificate faces challenges as both the certificate and the underlying environment model are complex, deep neural network models. SPICE [4] determines action safety using weakest preconditions derived from a learned neural environment model within a short time horizon H. However, extending H to cover the entire horizon of an RL task faces challenges due to the difficulty of constructing precise weakest precondition transformers for neural networks and the accumulation of approximation errors inherent in linearizing a neural environment model. Instead, VELM conducts formally verified exploration for RL agents, covering the entire horizon of an RL task through learned environment models.

6 Conclusion

In summary, we present VELM, a novel framework for ensuring verified safe exploration in model-based reinforcement learning. VELM learns environment models as symbolic formulas. Through formal reachability analysis over learned models, VELM constructs an online shielding layer that acts as a safeguard, confining RL agent exploration to a state space verified as safe in the learned model. The results of our experiments in various RL environments, alongside comparisons with state-of-the-art safe RL techniques, highlight the efficacy of VELM in significantly mitigating safety violations during online exploration while maintaining strong learning performance. VELM thus establishes a foundation for building trustworthy and secure RL systems capable of navigating complex environments while adhering to stringent safety constraints.

References

1. Achiam, J., Held, D., Tamar, A., Abbeel, P.: Constrained policy optimization. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, vol. 70 (2017)
2. Akametalu, A.K., Kaynama, S., Fisac, J.F., Zeilinger, M.N., Gillula, J.H., Tomlin, C.J.: Reachability-based safe learning with gaussian processes. In: 53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014 (2014)

3. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018 (2018)
4. Anderson, G., Chaudhuri, S., Dillig, I.: Guiding safe exploration with weakest preconditions. In: The Eleventh International Conference on Learning Representations (2023)
5. Anderson, G., Verma, A., Dillig, I., Chaudhuri, S.: Neurosymbolic reinforcement learning with formally verified exploration. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)
6. Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe model-based reinforcement learning with stability guarantees. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA (2017)
7. Bharadhwaj, H., Kumar, A., Rhinehart, N., Levine, S., Shkurti, F., Garg, A.: Conservative safety critics for exploration. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021 (2021)
8. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
9. Burlacu, B., Kronberger, G., Kommenda, M.: Operon c++: An efficient genetic programming framework for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. GECCO '20 (2020)
10. Cava, W.G.L., Orzechowski, P., Burlacu, B., de França, F.O., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.H.: Contemporary symbolic regression methods and their relative performance. In: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual (2021)
11. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044 (2013)
12. Cheng, R., Orosz, G., Murray, R.M., Burdick, J.W.: End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019 (2019)
13. Chow, Y., Nachum, O., Duéñez-Guzmán, E.A., Ghavamzadeh, M.: A lyapunov-based approach to safe reinforcement learning. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada (2018)
14. Chow, Y., Nachum, O., Faust, A., Duéñez-Guzmán, E.A., Ghavamzadeh, M.: Safe policy learning for continuous control. In: 4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA. Proceedings of Machine Learning Research, vol. 155 (2020)
15. Dalal, G., Dvijotham, K., Vecerík, M., Hester, T., Paduraru, C., Tassa, Y.: Safe exploration in continuous action spaces. CoRR **abs/1801.08757** (2018)
16. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
17. Donti, P.L., Roderick, M., Fazlyab, M., Kolter, J.Z.: Enforcing robust control guarantees within neural network policies. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021 (2021)

18. Fisac, J.F., Akametalu, A.K., Zeilinger, M.N., Kaynama, S., Gillula, J.H., Tomlin, C.J.: A general safety framework for learning-based control in uncertain robotic systems. IEEE Trans. Autom. Control. **64**(7) (2019)
19. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning **11**(3-4) (2018)
20. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018 (2018)
21. Fulton, N., Platzer, A.: Verifiably safe off-model reinforcement learning. In: Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427 (2019)
22. Gillula, J.H., Tomlin, C.J.: Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor. In: IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA (2012)
23. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80 (2018)
24. Hunt, N., Fulton, N., Magliacane, S., Hoang, T.N., Das, S., Solar-Lezama, A.: Verifiably safe exploration for end-to-end reinforcement learning. In: HSCC '21: 24th ACM International Conference on Hybrid Systems: Computation and Control, Nashville, Tennessee, May 19-21, 2021 (2021)
25. Janner, M., Fu, J., Zhang, M., Levine, S.: When to trust your model: Model-based policy optimization. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada (2019)
26. Jayant, A.K., Bhatnagar, S.: Model-based safe deep reinforcement learning via a constrained proximal policy optimization algorithm. In: NeurIPS (2022)
27. Johnson, T.T., Lopez, D.M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T.J., Weimer, J., Lee, I.: ARCH-COMP21 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21), Brussels, Belgium, July 9, 2021. EPiC Series in Computing, vol. 80 (2021)
28. Kamienny, P., d'Ascoli, S., Lample, G., Charton, F.: End-to-end symbolic regression with transformers. In: NeurIPS (2022)
29. Koller, T., Berkenkamp, F., Turchetta, M., Krause, A.: Learning-based model predictive control for safe exploration. In: 57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018 (2018)
30. Kronberger, G., de França, F.O., Burlacu, B., Haider, C., Kommenda, M.: Shape-constrained symbolic regression - improving extrapolation with prior knowledge. Evol. Comput. **30**(1) (2022)

31. 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 2020, Paris, France, May 31 - August 31, 2020 (2020)
32. Li, Y., Li, N., Tseng, H.E., Girard, A., Filev, D.P., Kolmanovsky, I.V.: Safe reinforcement learning using robust action governor. In: Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland. Proceedings of Machine Learning Research, vol. 144 (2021)
33. Liu, Z., Zhou, H., Chen, B., Zhong, S., Hebert, M., Zhao, D.: Safe model-based reinforcement learning with robust cross-entropy method. CoRR **abs/2010.07968** (2020)
34. Luo, Y., Ma, T.: Learning barrier certificates: Towards safe reinforcement learning with zero training-time violations. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual (2021)
35. Ma, Y.J., Shen, A., Bastani, O., Jayaraman, D.: Conservative and adaptive penalty for model-based safe reinforcement learning. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022 (2022)
36. Mania, H., Guy, A., Recht, B.: Simple random search of static linear policies is competitive for reinforcement learning. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada (2018)
37. Moldovan, T.M., Abbeel, P.: Safe exploration in markov decision processes. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012 (2012)
38. Sikchi, H., Zhou, W., Held, D.: Lyapunov barrier policy optimization. CoRR **abs/2103.09230** (2021)
39. Srinivasan, K., Eysenbach, B., Ha, S., Tan, J., Finn, C.: Learning to be safe: Deep RL with a safety critic. CoRR **abs/2010.14603** (2020)
40. Stooke, A., Achiam, J., Abbeel, P.: Responsive safety in reinforcement learning by PID lagrangian methods. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event. Proceedings of Machine Learning Research, vol. 119 (2020)
41. Tessler, C., Mankowitz, D.J., Mannor, S.: Reward constrained policy optimization. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019 (2019)
42. Thananjeyan, B., Balakrishna, A., Nair, S., Luo, M., Srinivasan, K., Hwang, M., Gonzalez, J.E., Ibarz, J., Finn, C., Goldberg, K.: Recovery RL: safe reinforcement learning with learned recovery zones. IEEE Robotics Autom. Lett. **6**(3) (2021)
43. Turchetta, M., Berkenkamp, F., Krause, A.: Safe exploration in finite markov decision processes with gaussian processes. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain (2016)
44. Verma, A., Le, H.M., Yue, Y., Chaudhuri, S.: Imitation-projected programmatic reinforcement learning. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada (2019)
45. Wang, Y., Zhu, H.: Verification-guided programmatic controller synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems - 29th International

Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13994 (2023)
46. Yang, Q., Simão, T.D., Tindemans, S.H., Spaan, M.T.J.: WCSAC: worst-case soft actor critic for safety-constrained reinforcement learning. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021 (2021)
47. Yang, T., Rosca, J., Narasimhan, K., Ramadge, P.J.: Projection-based constrained policy optimization. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020 (2020)
48. Yang, Z., Zhang, L., Zeng, X., Tang, X., Peng, C., Zeng, Z.: Hybrid controller synthesis for nonlinear systems subject to reach-avoid constraints. In: Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part I (2023)
49. Yu, D., Ma, H., Li, S., Chen, J.: Reachability constrained reinforcement learning. In: International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA. Proceedings of Machine Learning Research, vol. 162 (2022)
50. Zanger, M.A., Daaboul, K., Zöllner, J.M.: Safe continuous control with constrained model-based policy optimization. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021 (2021)
51. Zhang, Y., Vuong, Q., Ross, K.W.: First order constrained optimization in policy space. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)
52. 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, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019 (2019)