Dyna-Style Safety Augmented Reinforcement Learning: Staying Safe in the Face of Uncertainty

Artur Eisele[†,1] **Bernd Frauenknecht**[†] **Friedrich Solowjow** **Sebastian Trimpe**

Institute for Data Science in Mechanical Engineering, RWTH Aachen University

Abstract

Safety is a key problem in reinforcement learning, especially during exploration. While safety filters are promising to address this issue, they are ill-suited for high-dimensional systems with unknown dynamics. We propose *Dyna-style Safety Augmented Reinforcement Learning (Dyna-SAuR)*, a novel algorithm that learns a control policy as well as a safety filter requiring minimal domain knowledge by leveraging a learned uncertainty-aware dynamics model. A novel filter learning formulation requires the filter to avoid failures and areas of high model uncertainty. As joint deployment of the control policy and filter allows to safely collect data and improve the dynamics model, certain areas grow and filter conservatism decreases over time. Dyna-SAuR reduces failures compared to established safe RL baselines by two orders of magnitude on goal-reaching CartPole and MuJoCo Walker.

1 Introduction

Safety during training is a major challenge in reinforcement learning (RL). Safe RL methods typically treat safety as a soft constraint [1, 4] or use safety filters from control theory [7, 3, 48]. The former ensures safety only in expectation, while the latter requires extensive domain knowledge and struggles with high-dimensional problems. While recent results suggest safety filters can be obtained in a data driven way using RL [30], limitations such as access to a simulator and scaling of the method remain.

We introduce *Dyna-style Safety Augmented Reinforcement Learning* (Dyna-SAuR), a model-based reinforcement learning (MBRL) method that concurrently learns a control policy and a safety filter leveraging an uncertainty-aware dynamics model. Dyna-SAuR enables safe exploration, solely relying on a notion of failure and initial environment data.

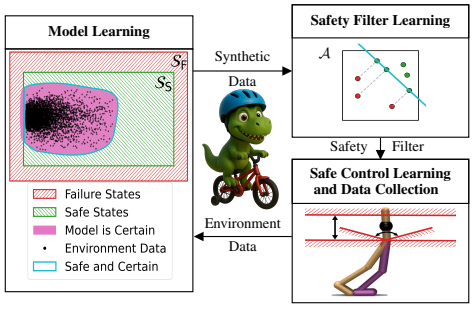


Figure 1: Dyna-SAuR mechanism. *An uncertainty-aware dynamics model is used to train a safety filter that avoids both failures and uncertain regions of the model. The filter is leveraged to safely learn a control policy in the environment. The collected interaction data is used to improve the dynamics model, which expands the certain area and reduces filter conservatism in the next iteration.*

Main Idea Evaluating safety involves identifying if subsequent failures occur, which can be assessed leveraging a learned dynamics model in synthetic rollouts without risk. However, the model is only reliable where it accurately models the environment, as illustrated in Figure 1. We train a safety filter to avoid both unsafe and uncertain regions. Deploying this filter enables safe interaction of a control

[†]Equal contribution. [1]Corresponding author: `{first name}.{surname} [at] dsme.rwth-aachen.de`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

policy with the environment within the model's certain area. The interaction data expands the certain area. This yields a less restrictive safety filter in the next iteration, granting more freedom to control.

To build such an architecture, we combine the notions of safety and certainty within the learned model, propose a compact RL formulation of the safety filter learning problem, and provide practical guidelines to scale the method up. In empirical evaluations on goal-reaching CartPole and MuJoCo Walker, Dyna-SAuR matches or exceeds the performance of established safe RL baselines, while reducing failures during training by at least two orders of magnitude.

2 Background

First, we introduce the necessary background on MBRL and safe learning to then specify the technical challenges we address for learning the safety filter in a model-based fashion leveraging RL.

2.1 Model-based Reinforcement Learning

We consider sequential decision-making in an environment modelled as a discrete-time Markov decision process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma\}$, with states $S_t \in \mathcal{S} \subseteq \mathbb{R}^{n_\mathcal{S}}$ and actions $A_t \in \mathcal{A} = [-1, 1]^{n_\mathcal{A}}$. The latter can be readily achieved through normalization. Rewards $R_t \in \mathbb{R}$ are generated by the reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, while the environment dynamics $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ propagate the system state, with Δ indicating a stochastic mapping. Starting from an initial state $S_0 \sim \rho_0$, the goal of RL is to find an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ within the set of policies Π that maximizes the expected return, i.e., the expected sum of rewards discounted by the factor $0 \leq \gamma < 1$,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]. \tag{1}$$

Where needed, we use S_t, A_t, and R_t to refer to random variables and distinguish them from their realizations s_t, a_t, and r_t. Further, the state-action occupancy of a policy π

$$\rho_\pi(s, a) := \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{P}_\pi \left[S_t = s, A_t = a \right] \tag{2}$$

describes the probability of being in a particular state-action pair under this policy.

To reduce costly and dangerous data collection in the environment, model-based RL trains dynamics models $\hat{p}(\cdot|S_t, A_t)$ to approximate the environment dynamics [18, 27]. In Dyna-style architectures [42, 28], $\hat{p}$ is used to simulate interactions to train model-free RL algorithms. However, $\hat{p}$ is typically unreliable outside the training data distribution. Therefore, models such as the probabilistic ensemble (PE) [29] capture epistemic uncertainty, where ensemble disagreement indicates prediction inaccuracy. This uncertainty is used to define a sufficiently certain set $\mathcal{E} \subseteq (\mathcal{S} \times \mathcal{A})$ where the environment is modeled accurately [22]. Following [23], we leverage the sufficiently certain set

$$\mathcal{E} := \{(s_t, a_t) \in \mathcal{S} \times \mathcal{A} \mid \mathbb{H}(\hat{S}_{t+1}) \leq \lambda_1\}, \tag{3}$$

where $\mathbb{H}(\hat{S}_{t+1})$ is the entropy of the model prediction that remains due to epistemic uncertainty. Low entropy corresponds to a good approximation of the environment dynamics p, thus predictions are considered trustworthy below a threshold λ_1. We refer to [23, 24] for a detailed presentation of the Infoprop state $\hat{S}_{t+1}$ and to Appendix A.1 for details on model learning and rollout generation.

2.2 Safe Learning

We focus on training-time safety [11], minimizing failures during learning rather than enforcing safety only at deployment of the converged agent. Failures are defined as entering a set of failure states $\mathcal{S}_\mathrm{F} \subseteq \mathcal{S}$, such that the complement $\mathcal{S}_\mathrm{S} := \mathcal{S} \setminus \mathcal{S}_\mathrm{F}$ defines the set of safe states[2]. However, some safe states inevitably lead to failure. Therefore, we aim to stay within a finite horizon[3] viability kernel $\mathcal{S}_\mathrm{V} \subseteq \mathcal{S}_\mathrm{S}$ that we define inspired by probabilistic viability theory [34, 5].

[2]We use terminology consistent with viability theory, differing from some works in safe RL, as discussed in Appendix E.2.

[3]Since we consider unbounded process noise, we cannot make infinite horizon statements [26].

Definition 2.1. (**Finite-Horizon Viability**) A state within the finite horizon viability kernel

$$\mathcal{S}_{\mathrm{V}} \coloneqq \left\{ s \in \mathcal{S} \mid \exists \pi \in \Pi : \mathbb{P}_{\pi} \left[\forall t \leq T,\, S_t \in \mathcal{S}_{\mathrm{S}} \mid S_0 = s \right] \geq 1 - \delta \right\} \tag{4}$$

is viable. That is, there exists a policy such that the system remains within the set of safe states with high probability $1 - \delta$ for a considered finite time horizon $T \in \mathbb{N}$ when starting from s.

The unviability kernel $\mathcal{S}_{\mathrm{U}} := \mathcal{S} \setminus \mathcal{S}_{\mathrm{V}}$ includes all states for which failure within horizon T is unavoidable with high probability, irrespective of the applied controls.

Definition 2.2. (**Viable Action**) For $s_t \in \mathcal{S}_{\mathrm{V}}$, a viable action a_t^{V} preserves viability of the next state.

Definition 2.3. (**Viable Policy**) The set of viable policies Π_{V} consists of all policies that select only viable actions and thus preserve viability with high probability $1 - \delta$ for a horizon T

$$\Pi_{\mathrm{V}} \coloneqq \left\{ \pi \in \Pi \mid \forall s \in \mathcal{S}_{\mathrm{V}} : \mathbb{P}_{\pi} \left[\forall t \leq T,\, S_t \in \mathcal{S}_{\mathrm{S}} \mid S_0 = s \right] \geq 1 - \delta \right\}. \tag{5}$$

A safety filter $\kappa : \Pi \to \Pi_{\mathrm{V}}$ avoids failures by projecting the set of policies into the set of viable policies. This is achieved by mapping actions of the policy in Π to the closest viable action. For control-affine dynamics p, the discriminator between viable and unviable actions is a state-dependent hyperplane $w_t^\top a \geq b_t$ with a normal vector $w_t \in \mathbb{R}^{n_{\mathcal{A}}}$ and an offset $b_t \in \mathbb{R}$ [30]. Thus,

$$a_t^{\mathrm{V}} = \kappa^{\pi}(s_t) := \arg\min_{a \in \mathcal{A}} \|a - \pi(s_t)\|_2 \text{ s.t. } w_t^\top a \geq b_t \tag{6}$$

describes the safety filter problem for a control policy $\pi \in \Pi$. However, obtaining these state-dependent hyperplanes using classical methods [3, 7] requires substantial domain knowledge and scales poorly with the system dimension [13, 48]. Thus, Lavanakul et al. [30] formulate the idea of training a filter policy $\mu(s_t)$ that parametrizes the hyperplane via a function h, resulting in the filter

$$a_t = \kappa_\mu^\pi(s_t) := \arg\min_{a \in \mathcal{A}} \|a - \pi(s_t)\|_2 \text{ s.t. } w_t^\top a \geq b_t \text{ with } (w_t, b_t) = h(\mu(s_t)). \tag{7}$$

The learned filter $\kappa_\mu^\pi : \Pi \to \mathrm{Img}(\kappa_\mu^\pi)$ with image $\mathrm{Img}(\kappa_\mu^\pi) \subseteq \Pi$ does not necessarily project into Π_{V} for all μ in the set of filter policies $\mu \in \mathrm{M}$. Instead, a suitable $\mu \in \mathrm{M}$ yielding a valid safety filter with $\mathrm{Img}(\kappa_\mu^\pi) \subseteq \Pi_{\mathrm{V}}$ needs to be found by carefully designing the learning problem.

3 Problem Statement

Following our main idea (cf. Figure 1), Dyna-SAuR requires finding the filter policy μ and the control policy π. Both of them leverage a learned uncertainty-aware dynamics model $\hat{p}$ with a sufficiently certain subset $\mathcal{E}$ defined in (3) and the set of viable policies $\Pi_{\mathrm{V}}(\mathcal{E}) \subseteq \Pi_{\mathrm{V}}$ that additionally remain within $\mathcal{E}$. Before defining $\Pi_{\mathrm{V}}(\mathcal{E})$, we formulate the **Filter Policy Learning Problem**[4]

$$\mu^* = \arg\max_{\mu \in \mathrm{M}} \{ \mathrm{Img}(\kappa_\mu^\pi) \subseteq \Pi_{\mathrm{V}}(\mathcal{E}) \}. \tag{8}$$

That is, finding a filter policy μ^* that projects into the largest possible subset of $\Pi_{\mathrm{V}}(\mathcal{E})$. Projecting into $\Pi_{\mathrm{V}}(\mathcal{E})$ makes κ_μ^π a valid safety filter, while the largest subset yields the least restrictive filter.

Given a filter policy μ, we aim to find the control policy π^* that maximizes the expected return of the filtered policy κ_μ^π, resulting in the **Control Policy Learning Problem**

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{\kappa_\mu^\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right]. \tag{9}$$

To address the filter policy learning problem (8) via MBRL, we answer the following questions:

(i) How can the concept of remaining certain be integrated into the viability kernel?
(ii) What is a unique representation of the hyperplane for an efficient action space design?
(iii) What is an informative data distribution to learn μ in the face of a finite sampling budget?

Subsequently, we combine recent results in model-based data generation [23], actor-critic learning [38], and filtered policy learning [36, 32] to address control policy learning (9). We limit the input to the learning algorithm to initial data $\mathcal{D}_0^{\hat{p},p}$ from the environment and the failure set $\mathcal{S}_{\mathrm{F}}$.

The remainder of this paper focuses on the key technical contributions for addressing (8). In particular, Section 4 focuses on question *(i)*, while Section 5 discusses questions *(ii)* and *(iii)*. Section 5 further presents a high level overview of the full algorithm, supporting intuition for Dyna-SAuR. Technical details, especially for addressing (9), are discussed in Appendix A.

[4] See Appendix E.1 for a more detailed discussion of the notation of equation(8).

4 Viability in the Uncertainty-Aware Model

To prevent failures in the environment, μ is trained exclusively in the dynamics model. Thus, we need to address question *(i)* and extend the viability concepts introduced in Section 2.2 to account not only for safety but also certainty under the model dynamics. Based on the sufficiently certain subset $\mathcal{E}$, the certain safe set $\mathcal{S}_{\mathrm{S}}(\mathcal{E})$ comprises state-action pairs where the model is certain and the state is safe

$$\mathcal{S}_{\mathrm{S}}(\mathcal{E}) := \left\{(s_t, a_t) \in \mathcal{S} \times \mathcal{A} \mid \mathbb{H}(\hat{S}_{t+1}) \leq \lambda_1, s_t \in \mathcal{S}_{\mathrm{S}}\right\}. \tag{10}$$

Consequently, the finite-horizon certain viability kernel

$$\mathcal{S}_{\mathrm{V}}(\mathcal{E}) := \left\{s \in \mathcal{S} \mid \exists \pi \in \Pi : \mathbb{P}_\pi \left[\forall t \leq T, (\hat{S}_t, A_t) \in \mathcal{S}_{\mathrm{S}}(\mathcal{E}) \mid S_0 = s\right] \geq 1 - \delta\right\} \tag{11}$$

comprises all states from which a policy exists that keeps the model in the certain safe set for a given time horizon with high probability. Finally, the set of certain viable policies

$$\Pi_{\mathrm{V}}(\mathcal{E}) \coloneqq \left\{\pi \in \Pi \mid \forall s \in \mathcal{S}_{\mathrm{V}}(\mathcal{E}) : \mathbb{P}_\pi \left[\forall t \leq T, (\hat{S}_t, A_t) \in \mathcal{S}_{\mathrm{S}}(\mathcal{E}) \mid S_0 = s\right] \geq 1 - \delta\right\} \tag{12}$$

allows the system to stay within the certain viability kernel with high probability. Thus, using the model $\hat{p}$ for filter training restricts (8) to the subspace $\Pi_{\mathrm{V}}(\mathcal{E}) \subseteq \Pi_{\mathrm{V}}$, which becomes less restrictive as $\mathcal{E}$ grows. Safety transfers to the environment only insofar as $\hat{p}$ approximates p well everywhere within $\mathcal{E}$. We evaluate this approximation empirically in Appendix C.3.

5 Dyna-SAuR: Dyna-style Safety Augmented Reinforcement Learning

In the following, we introduce the Dyna-SAuR architecture, which at its core addresses the two learning problems (8) and (9). In the following, we focus on the filter policy learning problem (8) as addressing the corresponding technical challenges constitutes our main technical contribution. The solution to (9) then builds on recent results in RL as discussed in Appendix A.

Algorithm 1 Dyna-SAuR

input $\mathcal{D}_0^{\hat{p},p}$, randomly initialized $\hat{p}_0$, π_0 and μ_0
1: **for** $j \in \{1, \ldots, J\}$ Dyna-SAuR iterations **do**
2: Train dynamics model $\hat{p}_j$ using data from $\mathcal{D}_{j-1}^{\hat{p},p}$ (Algorithm 3)
3: **while** Filter evaluation not successful **do**
4: Train filter policy μ_j using $\hat{p}_j$ and π_{j-1} (Algorithm 4)
5: Evaluate filter policy μ_j (Algorithm 5)
6: **end while**
7: Train control policy π_j using $\hat{p}_j$ and μ_j and collect env. data $\mathcal{D}_j^{\hat{p},p}$ (Algorithm 6)
8: **end for**
output $\hat{p}_J, \mu_J, \pi_J$

Algorithm 1 outlines the iterative training process.[5] First, the dynamics model $\hat{p}$ is trained on initial environment data $\mathcal{D}_0^{\hat{p},p}$ (line 2). Second, we learn a filter policy μ exclusively from model-based rollouts (line 4) until it passes an evaluation indicating successful safety filtering (line 5). Third, the control policy π, filtered with μ, is trained using both model data and environment interactions (line 7). Observed environment transitions are added to $\mathcal{D}_j^{\hat{p},p}$ (line 7). Retraining $\hat{p}$ on this expanded data (line 2) in iteration $j+1$ enlarges $\mathcal{E}$, enabling a less restrictive μ. At each iteration j, both policies are retrained from scratch, to avoid local optima [35]. Next, we discuss learning the filter policy μ and refer to Appendix A for a detailed description of the full algorithm.

5.1 Safety Filter MDP

To cast the filter objective (8) as a reinforcement learning problem, we define the safety filter MDP $\mathcal{M}^{\mathrm{SF}} = (\mathcal{S}, \mathcal{U}, p^{\mathrm{SF}}, r^{\mathrm{SF}}, \rho_0^{\mathrm{SF}}, \gamma_{\mathrm{SF}})$, which shares only the state space $\mathcal{S}$ with the control MDP $\mathcal{M}$ from Section 2.1. The filter policy $\mu : \mathcal{S} \to \mathcal{U}$ outputs hyperplane actions $u_t \in \mathcal{U}$, which a bijection h maps to w_t and b_t. We discuss the benefits of this design in Section 5.2. As depicted in Figure 2, the dynamics $p^{\mathrm{SF}} : \mathcal{S} \times \mathcal{U} \to \Delta(\mathcal{S})$ are composed of any control policy $\pi \in \Pi$, the environment dynamics p approximated by $\hat{p}$, the transform h, and the safety filter κ. The reward function r^{SF} encourages the least restrictive projection $\Pi \to \Pi_{\mathrm{V}}(\mathcal{E})$ as discussed in Section 5.3. As μ is trained on model-based rollouts, its initial state distribution ρ_0^{SF} does not need to match ρ_0 and is instead tailored to be more informative as presented in Section 5.4. Finally, the discount factor γ_{SF} defines the effective horizon T in (12) by inducing a random stopping time with expectation $\mathbb{E}[T] = \frac{1}{1-\gamma_{\mathrm{SF}}}$.

[5] We use iteration index j as a subscript for learnable objects, denoting parameters at convergence after the j^{th} iteration.

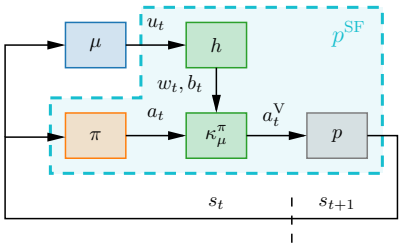


Figure 2: Safety Filter MDP Dynamics p^{SF}. *At state s_t, the control policy π proposes action a_t. The safety filter κ (7), parametrized by the filter policy μ via the bijective transform h and the hyperplane action u_t, outputs a viable action a_t^{V} applied to dynamics p.*

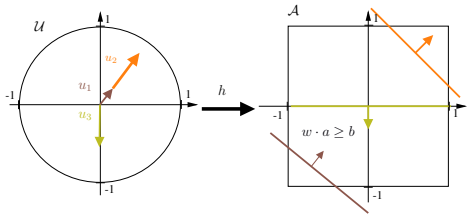


Figure 3: Hyperplane Action Space $\mathcal{U}$. *Example mappings between hyperplane actions $u \in \mathcal{U}$ and their corresponding discriminating hyperplanes in the control action space $\mathcal{A}$. Larger norms $\|u\|_2$ of the hyperplane action result in more restrictive hyperplanes.*

5.2 Efficient Filter Action Space Formulation

Addressing question *(ii)*, we present a novel, expressive design of $\mathcal{U}$. Lavanakul et al. [30] propose learning hyperplane parameters directly as $\tilde{u}_t = [w_t^\top, b_t]^\top$, which causes several problems. First, this formulation is unbounded with $\tilde{\mathcal{U}} = \mathbb{R}^{n_\mathcal{A}+1}$ and overparametrized since $c\, w_t^\top \cdot a \geq c\, b_t$ yields identical hyperplanes for any $c \in \mathbb{R}_{>0}$. Second, $\tilde{\mathcal{U}}$ does not enforce hyperplane intersection with the bounded control action space $\mathcal{A}$. These issues create a large, ambiguous search space.

Instead, we define the space of hyperplane actions as the unit hyperball in $\mathbb{R}^{n_\mathcal{A}}$

$$\mathcal{U} := \{u \in \mathbb{R}^{n_\mathcal{A}} \,|\, 0 < \|u\|_2 < 1\}, \tag{13}$$

resulting in a bounded search space. Subsequently, we obtain the hyperplane parameters via the bijective function $h : \mathcal{U} \to \mathbb{R}^{n_\mathcal{A}} \times \mathbb{R}$ that enforces intersection with $\mathcal{A}$.

Theorem 5.1. *The function h mapping each $u \in \mathcal{U}$ to a discriminating hyperplane intersecting the interior of $\mathcal{A} = [-1, 1]^{n_\mathcal{A}}$ is bijective. It is defined as $h(u_t) = (w_t, b_t)$ with*

$$w_t = \frac{u_t}{\|u_t\|_2}, \text{ and } b_t = (2\|u_t\|_2 - 1)\|w_t\|_1. \tag{14}$$

Proof. See Appendix B.1, Theorem B.2

Figure 3 illustrates the transformation[6]. The normal vector w_t points in the direction of u_t, where actions a_t in the positive direction of w_t are classified as viable. The offset b_t is computed using the distance from the origin to the farthest vertex of $\mathcal{A}$ along w_t, given by $\|w_t\|_1$ as shown in Lemma B.1. We scale this distance by $(2\|u_t\|_2 - 1) \in (-1, 1)$ by construction in (13). Thus, $\|u_t\|_2$ indicates restrictiveness: values near 0 classify most actions viable, while values close to 1 reject most. This yields a minimal search space for μ, improving the scalability of data-driven safety filter learning.

5.3 Filter Policy Learning as an RL Problem

As we aim for a general reward function that enforces staying within $\mathcal{S}_\mathrm{S}(\mathcal{E})$ under p^{SF}, we define

$$r^{\mathrm{SF}}(s_t, u_t) = \begin{cases} 1, & \text{if } (s_t, a_t^{\mathrm{V}}) \in \mathcal{S}_\mathrm{S}(\mathcal{E}), \\ -\frac{1}{1-\gamma_{\mathrm{SF}}}, & \text{otherwise} \end{cases} \tag{15}$$

where a_t^{V} results from u_t via the learned filter equation (7). Further, we terminate whenever $\mathcal{S}_\mathrm{S}(\mathcal{E})$ is left. As a consequence, the safety filter action value function

$$Q^{\mathrm{SF}}(s_t, u_t) := \mathbb{E}_{u\sim\mu, s\sim p^{\mathrm{SF}}}\left[\sum_{k=0}^{\infty} \gamma_{\mathrm{SF}}^{t+k} r_{t+k+1}^{\mathrm{SF}} \middle| s_t, u_t\right] \tag{16}$$

[6]Interactive demo of the mapping between hyperplane action and the coresponding hyperplane for a two-dimensional control action space: `https://data-science-in-mechanical-engineering.github.io/Dyna-SAuR/#demo`.

is bounded with $Q^{\mathrm{SF}}(s_t, u_t) \in [-\frac{1}{1-\gamma_{\mathrm{SF}}}, \frac{1}{1-\gamma_{\mathrm{SF}}}]$ for all state-filter action pairs (s_t, u_t). Here, a value of $-\frac{1}{1-\gamma_{\mathrm{SF}}}$ corresponds to failing in the next transition, and $\frac{1}{1-\gamma_{\mathrm{SF}}}$ corresponds to not failing at all. In particular, safety filter action values provide a lower bound on the expected time to failure

$$\mathbb{E}_{\mu}\left[T_{\mathrm{F}}|s\right] \geq \log_{\gamma_{\mathrm{SF}}}\left(\frac{1 - Q^{\mathrm{SF}}(s, \mu(s))(1-\gamma_{\mathrm{SF}})}{2}\right) \tag{17}$$

under μ as derived in Lemma B.3 of Appendix B.2. Thus, maximizing Q^{SF} enforces the system to stay within $\mathcal{S}_{\mathrm{S}}(\mathcal{E})$ for as long as possible, which closely reflects (12).

To obtain a least restrictive projection $\Pi \to \Pi_{\mathrm{V}}(\mathcal{E})$, we leverage the fact that $\|u_t\|_2$ indicates filter restrictiveness, with lower norms corresponding to more viable actions. Thus, maximizing

$$\mu(s_t) \leftarrow \arg\max_{\mu \in \mathrm{M}} Q^{\mathrm{SF}}\left(s_t, \mu(s_t)\right) - c\|\mu(s_t)\|_2 \tag{18}$$

with a regularization $\|\mu(s_t)\|_2$ scaled by c, closely resembles the filter policy learning problem (8).

5.4 Addressing Stochasticity via Data Shaping

Solving the RL objective (18) in practice is challenging due to the randomness in the dynamics p^{SF} depicted in Figure 2. While the filter components h and κ are deterministic, and the control MDP dynamics p typically introduce mild stochasticity, allowing any $\pi \in \Pi$ makes p^{SF} highly stochastic. Thus, areas in $\mathcal{S} \times \mathcal{U}$ need to be heavily sampled to obtain reliable value estimates $Q^{\mathrm{SF}}(s_t, u_t)$ and to train an effective filter policy. Learning an expressive Q^{SF} everywhere, akin to Hamilton-Jacobi reachability methods, is infeasible for systems of the considered complexity [48]. Instead, we leverage two principles that helped scale dynamic programming to modern deep RL: large-scale, parallelized data generation, and concentrating data and representational capacity on relevant regions of $\mathcal{S} \times \mathcal{U}$. We leverage accurate long-horizon rollouts using the model $\hat{p}$ [23] to cheaply generate large amounts of data, and actor-critic algorithms tailored for large datasets [38]. Addressing question *(iii)*, we present heuristics for concentrating sampling budget in meaningful areas of $\mathcal{S} \times \mathcal{U}$.

Generally, RL methods with function approximation minimize the mean squared value error [43]

$$\xi := \int_{\mathcal{S}} \int_{\mathcal{U}} \rho(s, u) \left(q^{\mathrm{SF}}(s, u) - Q^{\mathrm{SF}}(s, u)\right)^2 \mathrm{d}u \, \mathrm{d}s \tag{19}$$

where q^{SF} is the true filter action value function and $\rho(s, u)$ is the state-action distribution used for training. Equation (19) shows that Q^{SF} is generally more accurate in high-density regions of $\rho(s, u)$. This distribution depends on ρ_0^{SF}, μ, and p^{SF}. We shape $\rho(s, u)$ by oversampling informative states in the initial state distribution ρ_0^{SF} and finding a suitable representation for the stochasticity in p^{SF}.

We use three sources for starting states ρ_0^{SF} in model-based rollouts. First, ρ_0^{SF} must include the environment distribution ρ_0, approximated from $\mathcal{D}_0^{\hat{p},p}$, to mimic rollouts in the environment. Second, failed rollouts τ_{F} that left $\mathcal{S}_{\mathrm{S}}(\mathcal{E})$ should be overrepresented, since μ does not yet provide a valid safety filter in these areas and requires more training. Third, rollouts τ_{G} yielding high returns concerning the control reward r should be overrepresented, since an overly conservative filter would be especially harmful in these areas. Thus, we want to make sure that the filter policy RL objective (18) converges concerning the regularization loss $\|u_t\|_2$ and finds a least restrictive projection. This yields

$$\rho_0^{\mathrm{SF}} = \nu_1 \rho_0 + \nu_2 \mathbf{U}(\mathcal{T}_{\mathrm{F}}) + \nu_3 \mathbf{U}(\mathcal{T}_{\mathrm{G}}) \tag{20}$$

with $\sum_{i=1}^{3} \nu_i = 1$ and $\nu_i \geq 0$, which is the mixture distribution over ρ_0 and uniformly sampling states from the trajectory sets $\tau_{\mathrm{F}} \in \mathcal{T}_{\mathrm{F}}$ and $\tau_{\mathrm{G}} \in \mathcal{T}_{\mathrm{G}}$.

Next, we face the challenge of approximating the stochasticity of p^{SF} induced by $\pi \in \Pi$. Since Π typically is a continuous and unbounded search space, sampling policies at random is unlikely to yield satisfactory results given a finite sampling budget. Thus, instead of considering the policy space directly, we focus on the state occupancy corresponding to Π. We define this occupancy $\rho_{\pi \in \Pi}(s, a) := \mathbb{E}_{\pi \in \Pi}[\rho_{\pi}(s, a)]$ as the expected occupancy over all $\pi \in \Pi$. Further, the state occupancy of certain viable policies $\rho_{\pi \in \Pi_{\mathrm{V}}(\mathcal{E})}(s, a) := \mathbb{E}_{\pi \in \Pi_{\mathrm{V}}(\mathcal{E})}[\rho_{\pi}(s, a)]$ has no probability mass outside of $\mathcal{S}_{\mathrm{S}}(\mathcal{E})$.

A filter κ_{μ}^{π} projecting any policy $\pi \in \Pi$ to a certain viable policy in $\Pi_{\mathrm{V}}(\mathcal{E})$, also represents a mapping between the corresponding state occupancies, such that

$$\kappa_{\mu}^{\pi} : \Pi \to \Pi_{\mathrm{V}}(\mathcal{E}) \iff \kappa_{\mu}^{\pi} : \rho_{\pi \in \Pi} \to \rho_{\pi \in \Pi_{\mathrm{V}}(\mathcal{E})}. \tag{21}$$

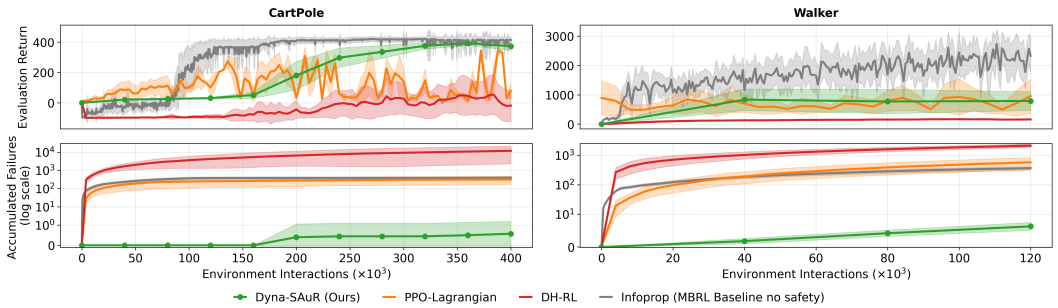


Figure 5: Control Return and Accumulated Failures (log scale). *Performance is measured after each Dyna-SAuR iteration, as the agent is retrained from scratch. The results demonstrate that Dyna-SAuR matches or exceeds the performance of safe RL algorithms (PPO-Lagrangian, DH-RL) while significantly reducing the number of failures compared to all approaches.*

Thus, for learning the filter policy, it suffices to find a projection into $\rho_{\pi\in\Pi_{\mathrm{V}}(\mathcal{E})}$ that maps all probability mass into $\mathcal{S}_{\mathrm{S}}(\mathcal{E})$, regardless of whether the specific control actions originate from a policy within Π.

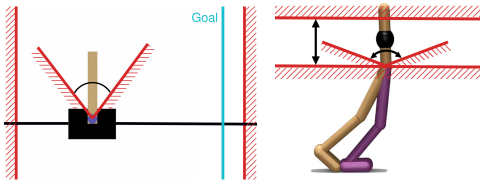
This insight allows us to approximate $\rho_{\pi\in\Pi}$, the filter policy needs to project into $\rho_{\pi\in\Pi_{\mathrm{V}}(\mathcal{E})}$ using

$$\rho_{\pi\in\Pi}(s,a) \approx \nu_4 \rho_{\pi_{j-1}+\sigma_2\epsilon}(s,a) + \nu_5 \rho_{\sigma_3\epsilon}(s,a), \tag{22}$$

with $\nu_4 + \nu_5 = 1$ and $\nu_4, \nu_5 \geq 0$. This represents a weighted sum of the state occupancies corresponding to the current control policy π_{j-1} with exploration noise ϵ and the pure noise ϵ, with scaling factors $\sigma_2, \sigma_3 \in \mathbb{R}$. We use temporally correlated pink noise $\epsilon \sim \mathrm{PN}$ to balance local and global exploration [19]. This design yields a diverse approximation of $\rho_{\pi\in\Pi}$ while emphasizing the current best policy π_{j-1}, ensuring the resulting filter performs well for greedy control actions.

6 Experiments and Discussion

We evaluate Dyna-SAuR on common RL benchmarks that we extend with constraints that must not be violated, as depicted in Figure 4.[7] We consider a constraint-violating state unsafe, while Dyna-SAuR needs to learn the distinction between viable and unviable actions without direct access to the environment dynamics. Specifically, we present results on a goal-reaching CartPole task ($\mathcal{S} \subseteq \mathbb{R}^4, \mathcal{A} \subseteq \mathbb{R}$) and MuJoCo Walker ($\mathcal{S} \subseteq \mathbb{R}^{17}, \mathcal{A} \subseteq \mathbb{R}^6$).



(a) CartPole (b) Walker

Figure 4: Safe Learning Environments with Constraints (red) and Goal (blue).

Goal reaching CartPole extends the standard task [47] with a goal, such that the CartPole needs to leave its equilibrium to reach the goal state at the physical limits without constraint violation. Further, Walker [46] is a relatively high dimensional problem with discontinuous contact dynamics that is beyond the scaling capabilities of most viability-based safety filter approaches. Thus, we compare Dyna-SAuR to scalable established safe RL baselines, where it yields comparable or superior control performance while reducing accumulated failures during training by at least two orders of magnitude.

6.1 Experimental Setup

We compare Dyna-SAuR to *(a)* PPO-Lagrangian [37], an established model-free safe RL approach that minimizes failures in expectation; *(b)* DH-RL [30], a safe RL method that learns a safety filter policy through interaction with a simulator; and *(c)* Infoprop-Dyna [23], a recent MBRL method without safety mechanisms. Dyna-SAuR is an extension to DH-RL, as it learns a filter policy with action space $\mathcal{U}$ (Section 5.2) from a model, and to Infoprop-Dyna as it extends MBRL to safe RL.

[7] All code is provided under `https://github.com/Data-Science-in-Mechanical-Engineering/Dyna-SAuR`.

We provide the initial dataset $\mathcal{D}_0^{\hat{p},p}$ to both Dyna-SAuR and Infoprop-Dyna. Since PPO-Lagrangian and DH-RL cannot directly operate on offline data, we warmstart them with the same number of online environment interactions as stored in $\mathcal{D}_0^{\hat{p},p}$ before reporting performance and counting safety violations. Appendix D provides a detailed discussion of the experimental setup, including the generation of $\mathcal{D}_0^{\hat{p},p}$. Initial data collection is decoupled from Dyna-SAuR: the data can be replaced by logged data from any behavior policy, and its generating controllers are not used during safety filter training. All experiments are averaged over 10 random seeds with solid lines representing the mean and shaded regions the 99% confidence interval.

6.2 Control Performance and Safety

Figure 5 reports control returns (top row) and accumulated failures (bottom row) over environment interactions for both tasks. Since each Dyna-SAuR iteration comprises retraining the policy from scratch, only the final evaluation return after each iteration is shown as a green dot to maintain readability. All failures incurred during training are included. Dyna-SAuR shows the strongest control performance of all safe RL approaches on goal-reaching CartPole and performs on par with PPO-Lagrangian on Walker. Infoprop-Dyna consistently outperforms the safe RL approaches concerning control performance, but does not account for safety. Dyna-SAuR reduces the amount of accumulated failures by at least two orders of magnitude compared to the other methods.

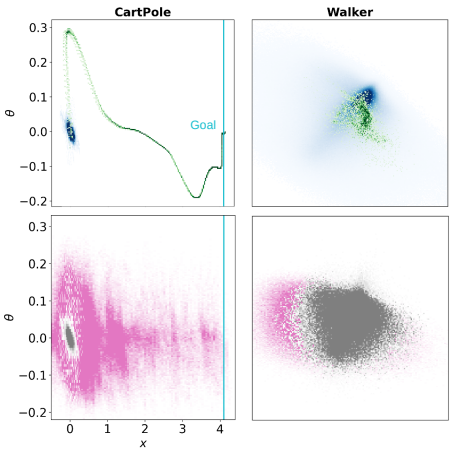


Figure 6: Exploration Throughout Training. *Top row: Initial data $\mathcal{D}_0^{\hat{p},p}$ (blue) vs. data of the final filtered policy $\kappa_{\mu_J}^{\pi_J}$ (green); Bottom row: Initial certain set $\mathcal{E}_0$ (grey) vs. final certain set $\mathcal{E}_J$ (pink). Dyna-SAuR explores the environment beyond initial data.*

Figure 6 investigates the exploration behavior of Dyna-SAuR. The top row compares the initial data set $\mathcal{D}_0^{\hat{p},p}$ to trajectories under the final filtered policy $\kappa_{\mu_J}^{\pi_J}$, in blue and green, respectively. Similarly, the bottom row compares the initial certain set $\mathcal{E}_0$ in grey to the final certain set $\mathcal{E}_J$ in pink.

For goal-reaching CartPole, we plot the subspace $x \times \theta$, indicating a large disconnect between initial data and the final policy. In particular, the initial data and accurate set are far from the goal state. Consequently, $\mathcal{E}$ needed to expand substantially, to learn higher performing policies, which explains the gradual increase in control performance in Figure 5. Since Dyna-SAuR accumulates at most one failure per seed on goal-reaching CartPole, the gradual expansion of $\mathcal{E}$ yielding less restrictive μ appears to work as intended.

Given the 17-dimensional state space, we perform a principal component analysis to compare the distributions in Walker. Both the differences between initial data and final policy as well as between the initial and final certain set are substantial, while less pronounced than in CartPole. Combined with the strong but stagnant control performance in Figure 5, this suggests that efficiently expanding $\mathcal{E}$ in high-dimensional spaces is a limitation of the current approach.

6.3 Ablation of Filter Policy Design Decisions

We present ablation results of the design decisions presented in Section 5 for goal-reaching CartPole in Figure 7 with results for Walker in Appendix C.1.

First, the predominant importance of formulating the action filter space as presented in Section 5.2 is illustrated when compared to learning w, b directly as

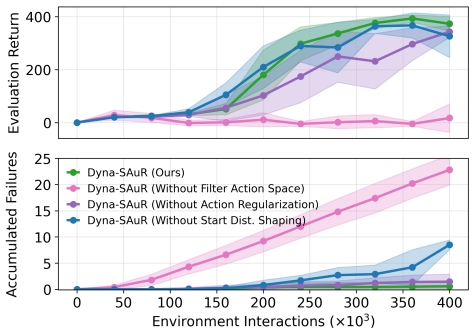


Figure 7: Ablation of Dyna-SAuR Design Choices. *Only the combination of the compact filter action space (Section 5.2), the regularization loss (Section 5.3), and the distribution shaping (Section 5.4), yields strong results concerning performance and safety.*

proposed in [30], shown in pink. The method is incapable of learning a reliable safety filter that allows for high control returns. Second, removing the regularization loss on u from the objective (18) introduced in Section 5.3, shown in purple, still yields a good filter but shows weaker control performance. This indicates a valid but restrictive projection into $\Pi_{\mathrm{V}}(\mathcal{E})$, as expected. Third, removing the heuristics for shaping ρ_0^{SF} presented in Section 5.4 and starting from ρ_0, instead, yields the blue curve. In this case, we observe an increased amount of failures, indicating the importance of overrepresenting $\mathcal{T}_{\mathrm{F}}$, and slightly weaker control performance as $\mathcal{T}_{\mathrm{G}}$ is not represented.

We provide an analysis of the learned safety filter in Appendix C.2 and visualize the Dyna-SAuR training process in a video[8] in the supplementary material.

7 Related Work

Besides penalizing constraint violations in the reward function [33], safety in RL is frequently addressed through constrained Markov decision processes (CMDPs) [2], where the expected cumulative safety cost is constrained by a budget [48]. While widely adopted [1, 44, 37, 41, 6, 54, 52, 4], CMDPs address safety in expectation over trajectories, which is insufficient for many applications such as collision avoidance [55], where even a single constraint violation can be catastrophic.

State-wise constrained MDPs (SCMDPs) make stricter statements by formulating a cost budget per state [55] instead of trajectory wide budgets. Methods addressing state-wise safety of the deployed agent after training often require minimal prior knowledge [14, 10, 53]. Conversely, approaches addressing training-time safety typically rely on strong assumptions, such as known or stable dynamics and prior safe controllers [9, 21, 49, 51, 45].

Viability-based approaches yield rigorous safety filters providing hard state-wise safety guarantees, but often face computational limits or demand substantial domain knowledge [48]. Methods based on Hamilton–Jacobi reachability, control barrier functions, or Lyapunov functions [8, 39, 20, 15, 31] generally require nominal models or a given subset of viable states.

Data-driven approximations to viability-based safety filters reduce these requirements but face limitations, including the inability to handle discontinuous dynamics [50], the need for unfiltered environment interaction, or accurate sampling models [40, 25, 30, 17].

To the best of our knowledge, Dyna-SAuR is the first RL method that approximates viability-based filters for nonlinear and unstable systems, without access to the environment dynamics or a subset of viable states. Further, it provides training-time safety for systems with state-action dimensionalities beyond the reach of existing viability-based approaches.

8 Concluding Remarks

Safe exploration is a fundamental concern in RL, yet traditional safety filters require substantial domain knowledge and struggle with high-dimensional problems. Conversely, RL excels at learning control for high-dimensional systems that are hard to model from first principles. We propose Dyna-SAuR, a data-driven approach to filter synthesis that jointly trains control and filter policies using a learned uncertainty-aware dynamics model. Inspired by viability theory, we present a compact RL formulation that approximates a rigorous safety filter matching the predictive capabilities of the model. Specifically, we combine the notion of viability and certainty, introduce a novel filter parametrization, and provide scaling guidelines. Using the model, we limit required domain knowledge to a notion of failure and an initial dataset. Dyna-SAuR matches or exceeds established safe RL control performance while reducing failures by at least two orders of magnitude.

Providing a novel perspective on viability-based safety filtering, Dyna-SAuR also raises several interesting directions for future research, addressing current limitations of the method. Among these are improving the exploration capabilities in high-dimensional problems, e.g., through active exploration of the dynamics model [12], improving the compute efficiency by leveraging knowledge of the control and filter policies of prior iterations instead of retraining them from scratch, or extending the underlying theory beyond control-affine systems. Nonetheless, we believe Dyna-SAuR represents a significant step towards safer, more reliable RL agents.

[8] Video of training: `https://data-science-in-mechanical-engineering.github.io/Dyna-SAuR/`.

Acknowledgement

We thank Lukas Kesper, Devdutt Subhasish, and Pierre-François Massiani for the valuable discussions on the work presented in this paper. This work was funded in part by the German Federal Ministry of Research, Technology and Space (BMFTR) under grant number 02P22A110. The responsibility for the content of this publication lies with the authors. The authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (project number p0022301). This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities (www.nhr-verein.de/unsere-partner). We acknowledge EuroHPC Joint Undertaking for awarding us access to MareNostrum5 as BSC, Spain. Friedrich Solowjow is supported by the KI-Starter grant by the state of NRW. This work is funded in part under the Excellence Strategy of the Federal Government and the Länder (G:(DE-82)EXS-SF-OPSF854). (G:(DE-82)EXS-SF-OPSF854).

References

[1] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, 2017.

[2] E. Altman. *Constrained Markov Decision Processes: Stochastic Modeling*. Routledge, 2021.

[3] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *European Control Conference*, 2019.

[4] Y. As, B. Sukhija, L. Treven, C. Sferrazza, S. Coros, and A. Krause. ActSafe: Active exploration with safety constraints for reinforcement learning. In *International Conference on Learning Representations*, 2025.

[5] J.-P. Aubin, A. M. Bayen, and P. Saint-Pierre. Viability and capturability. In *Viability Theory: New Directions*. Springer, second edition, 2011.

[6] Q. Bai, A. S. Bedi, and V. Aggarwal. Achieving zero constraint violation for constrained reinforcement learning via conservative natural policy gradient primal-dual algorithm. In *AAAI Conference on Artificial Intelligence*, 2023.

[7] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton–Jacobi reachability: A brief overview and recent advances. In *IEEE Conference on Decision and Control*, 2017.

[8] O. Bastani and S. Li. Safe reinforcement learning via statistical model predictive shielding. In *Robotics: Science and Systems*, 2021.

[9] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, 2017.

[10] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg. Conservative safety critics for exploration. In *International Conference on Learning Representations*, 2021.

[11] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 2022.

[12] M. Buisson-Fenet, F. Solowjow, and S. Trimpe. Actively learning Gaussian process dynamics. In *Learning for Dynamics and Control Conference*, 2020.

[13] M. Chen and C. J. Tomlin. Hamilton–Jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[14] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *AAAI Conference on Artificial Intelligence*, 2019.

[15] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath. Reinforcement learning for safety-critical control under model uncertainty, using control Lyapunov functions and control barrier functions. In *Robotics: Science and Systems*, 2020.

[16] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 2018.

[17] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. arXiv preprint arXiv:1801.08757, 2018.

[18] M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

[19] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *International Conference on Learning Representations*, 2023.

[20] Y. Emam, G. Notomista, P. Glotfelter, Z. Kira, and M. Egerstedt. Safe reinforcement learning using robust control barrier functions. *IEEE Robotics and Automation Letters*, 2022.

[21] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula, and C. J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 2019.

[22] B. Frauenknecht, A. Eisele, D. Subhasish, F. Solowjow, and S. Trimpe. Trust the model where it trusts itself: Model-based actor-critic with uncertainty-aware rollout adaption. In *International Conference on Machine Learning*, 2024.

[23] B. Frauenknecht, D. Subhasish, F. Solowjow, and S. Trimpe. On rollouts in model-based reinforcement learning. In *International Conference on Learning Representations*, 2025.

[24] B. Frauenknecht, D. Subhasish, A. Eisele, F. Solowjow, and S. Trimpe. All models are wrong, knowing where is useful: On model uncertainty in reinforcement learning. arXiv preprint arXiv:2606.01363, 2026.

[25] M. Ganai, Z. Gong, C. Yu, S. L. Herbert, and S. Gao. Iterative reachability estimation for safe reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[26] Y. Gao, K. H. Johansson, and L. Xie. Computing probabilistic controlled invariant sets. *IEEE Transactions on Automatic Control*, 2021.

[27] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025.

[28] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 2019.

[29] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 2017.

[30] W. Lavanakul, J. Choi, K. Sreenath, and C. Tomlin. Safety filters for black-box dynamical systems by learning discriminating hyperplanes. In *Learning for Dynamics and Control Conference*, 2024.

[31] Y. Luo and T. Ma. Learning barrier certificates: Towards safe reinforcement learning with zero training-time violations. In *Advances in Neural Information Processing Systems*, 2021.

[32] H. Markgraf, S. Sawant, H. Krasowski, L. Schäfer, S. Gros, and M. Althoff. Safe reinforcement learning using action projection: Safeguard the policy or the environment? *Transactions on Machine Learning Research*, 2026.

[33] P.-F. Massiani, S. Heim, F. Solowjow, and S. Trimpe. Safe value functions. *IEEE Transactions on Automatic Control*, 2023.

[34] P.-F. Massiani, A. von Rohr, L. Haverbeck, and S. Trimpe. Viability of future actions: Robust safety in reinforcement learning via entropy regularization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2025.

[35] E. Nikishin, M. Schwarzer, P. D'Oro, P.-L. Bacon, and A. Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, 2022.

[36] F. Pizarro Bejarano, L. Brunke, and A. P. Schoellig. Safety filtering while training: Improving the performance and sample efficiency of reinforcement learning agents. *IEEE Robotics and Automation Letters*, 2025.

[37] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. Technical report, OpenAI, 2019.

[38] Y. Seo, C. Sferrazza, H. Geng, M. Nauman, Z.-H. Yin, and P. Abbeel. FastTD3: Simple, fast, and capable reinforcement learning for humanoid control. arXiv preprint arXiv:2505.22642, 2025.

[39] Y. Shao, C. Chen, S. Kousik, and R. Vasudevan. Reachability-based trajectory safeguard (RTS): A safe and fast reinforcement learning safety layer for continuous control. *IEEE Robotics and Automation Letters*, 2021.

[40] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *IEEE International Conference on Robotics and Automation*, 2024.

[41] A. Sootla, A. I. Cowen-Rivers, T. Jafferjee, Z. Wang, D. H. Mguni, J. Wang, and H. Ammar. Saute RL: Almost surely safe reinforcement learning using state augmentation. In *International Conference on Machine Learning*, 2022.

[42] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 1991.

[43] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

[44] C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019.

[45] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg. Recovery RL: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 2021.

[46] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[47] M. Towers, A. Kwiatkowski, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, J. K. Terry, A. Pierré, S. V. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025.

[48] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. D. Ames, and M. N. Zeilinger. Data-driven safety filters: Hamilton–Jacobi reachability, control barrier functions, and predictive methods for uncertain systems. *IEEE Control Systems Magazine*, 2023.

[49] A. Wachi, Y. Sui, Y. Yue, and M. Ono. Safe exploration and optimization of constrained MDPs using Gaussian processes. In *AAAI Conference on Artificial Intelligence*, 2018.

[50] Y. Wang, S. S. Zhan, R. Jiao, Z. Wang, W. Jin, Z. Yang, Z. Wang, C. Huang, and Q. Zhu. Enforcing hard constraints with soft barriers: Safe reinforcement learning in unknown stochastic environments. In *International Conference on Machine Learning*, 2023.

[51] T. Wei and C. Liu. Safe control with neural network dynamic models. In *Learning for Dynamics and Control Conference*, 2022.

[52] L. Zhang, L. Shen, L. Yang, S. Chen, X. Wang, B. Yuan, and D. Tao. Penalized proximal policy optimization for safe reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022.

[53] L. Zhang, Q. Zhang, L. Shen, B. Yuan, X. Wang, and D. Tao. Evaluating model-free reinforcement learning toward safety-critical tasks. In *AAAI Conference on Artificial Intelligence*, 2023.

[54] Y. Zhang, Q. Vuong, and K. W. Ross. First order constrained optimization in policy space. In *Advances in Neural Information Processing Systems*, 2020.

[55] W. Zhao, T. He, R. Chen, T. Wei, and C. Liu. State-wise safe reinforcement learning: A survey. In *International Joint Conference on Artificial Intelligence*, 2023.

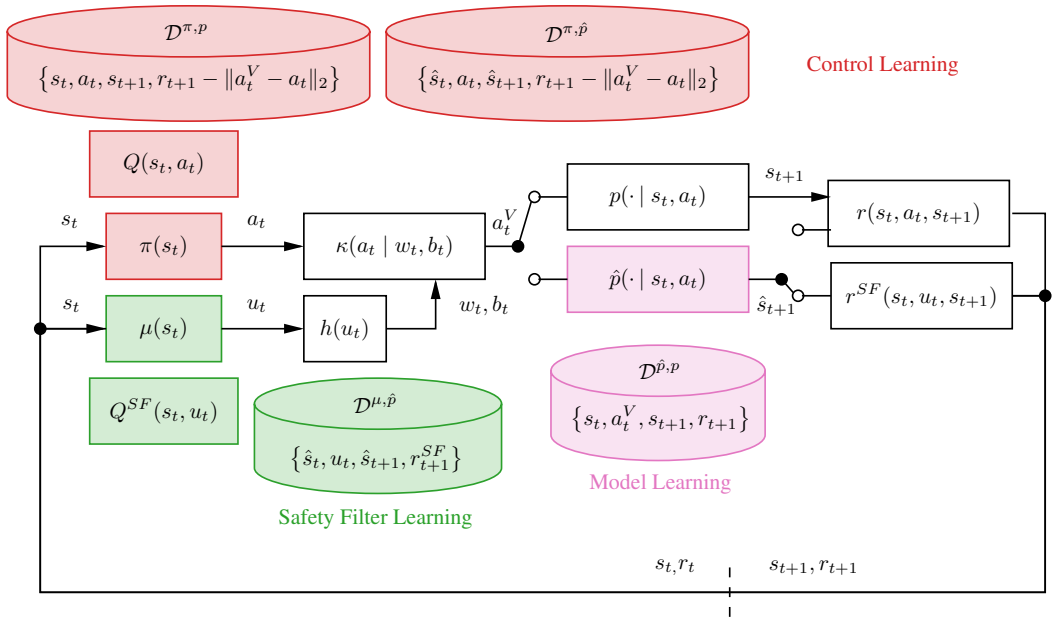


Figure 8: Overview of the three learning problems in Dyna-SAuR and their interactions. *During filter learning, the control policy π is fixed. The filter replay buffer $\mathcal{D}^{\mu,\hat{p}}$ is populated with model transitions $(\hat{s}_t, u_t, \hat{s}_{t+1}, r_{t+1}^{\text{SF}})$, where the filter policy μ transforms potentially unsafe actions a_t into safe actions a_t^V while exploring the action space. During control learning, the filter μ is fixed. The control policy π maintains two replay buffers: an environment control buffer $\mathcal{D}^{\pi,p}$ and a model control buffer $\mathcal{D}^{\pi,\hat{p}}$. Model rollouts, in which actions proposed by π are filtered by μ, are used to populate $\mathcal{D}^{\pi,\hat{p}}$. In parallel, the control policy interacts with the environment, with its actions filtered by μ, to populate $\mathcal{D}^{\pi,p}$. In both cases, the environment reward is penalized by the magnitude of filtering, $\|a_t^V - a_t\|_2$. The unfiltered action a_t is stored in both control buffers. Transitions from the environment control buffer are additionally used to populate the model learning buffer $\mathcal{D}^{\hat{p},p}$, without reward penalization and using the filtered action a_t^V, as this is the action applied to the true dynamics p.*

A Detailed Algorithm Description of Dyna-SAuR

In this section, we provide a more detailed overview of how the three learning problems depicted in Figure 8 interact with each other. We begin with model learning in Section A.1, then continue with the filter learning problem in Section A.2, and conclude with the (safety-filtered) control learning problem in Section A.3. Algorithm 2 combines all learning problems into the Dyna-SAuR algorithm.

Algorithm 2 Dyna-SAuR

input $\mathcal{D}_0^{\hat{p},p}$, randomly initialized $\hat{p}_0$, π_0 and μ_0
  for $j \in \{1, \ldots, J\}$ Dyna-SAuR iterations **do**
    Train dynamics model $\hat{p}_j$ using data from $\mathcal{D}_{j-1}^{\hat{p},p}$ (Algorithm 3)
    repeat
      Train filter policy μ_j using $\hat{p}_j$ and π_{j-1}
      (Algorithm 4)
      Evaluate filter policy μ_j (Algorithm 5)
    until average evaluation episode length does not improve
    Train control policy π_j using $\hat{p}_j$ and μ_j and collect env. data $\mathcal{D}_j^{\hat{p},p}$ (Algorithm 6)
  end for
output $\hat{p}_J, \mu_J, \pi_J$

A.1 Probabilistic Ensemble Model Learning

Algorithm 3 Dynamics Model Learning

input $\mathcal{D}_{j-1}^{\hat{p},p}, \hat{p}_\omega$
  for model training steps **do**
    Sample mini-batch of N^{Model} transitions from $\mathcal{D}^{\hat{p},p}$
    Sample ensemble member $\hat{p}_{\omega_e}$ with $e \sim \mathbf{U}(\{1,\dots,E\})$
    $\omega_e \leftarrow \arg\min_{\omega_e} -\frac{1}{N^{\text{Model}}} \sum \log \hat{p}_{\omega_e}(s_{t+1}|s_t, a_t^{\text{V}})$
  end for
output $\hat{p}_\omega$

At the beginning of iteration j of Dyna-SAuR, we use the model learning buffer $\mathcal{D}_{j-1}^{\hat{p},p}$ to train a probabilistic ensemble model [29, 16] by minimizing the negative log-likelihood, as described in Algorithm 3. Probabilistic ensembles (PE) have members each predicting a normal distribution over the next state. The idea is that ensemble members converge to identical predictions within the data support, while the members' predictions disagree in out-of-distribution settings. This disagreement increases the predictive uncertainty of the overall PE model in the form of epistemic uncertainty.

Recent work on probabilistic ensemble models [23] restricts model rollouts to a sufficiently certain set $\mathcal{E}_j \subseteq (\mathcal{S} \times \mathcal{A})$ in which the environment dynamics are modeled accurately. Infoprop [23] separates aleatoric from epistemic uncertainty in model rollouts. At each step, it draws a realization of the aleatoric noise process, and the remaining uncertainty about this realization, which stems from epistemic uncertainty, is quantified by the entropy $\mathbb{H}(\hat{S}_{t+1})$ of the Infoprop state $\hat{S}_{t+1}$[9]. Based on this quantity, Infoprop restricts model rollouts to a sufficiently certain set $\mathcal{E}_j \subseteq \mathcal{S} \times \mathcal{A}$, in which the model is assumed to be sufficiently accurate:

$$\mathcal{E}_j := \{(s_t, a_t) \in \mathcal{S} \times \mathcal{A} \mid \mathbb{H}(\hat{S}_{t+1}) \leq \lambda_1\}, \tag{23}$$

i.e., the entropy of the predicted next Infoprop state is at most λ_1. The model of iteration j is trained on $\mathcal{D}_{j-1}^{\hat{p},p}$, which contains data collected within $\mathcal{E}_{j-1}$.

In addition, the accumulated information loss is thresholded for each trajectory $\hat{\tau}$

$$\hat{\tau} = (\hat{s}_0, a_0, \hat{s}_1, \dots, a_{T-1}, \hat{s}_T) \tag{24}$$

by λ_2. This defines sufficiently accurate trajectories of length $t' \in \{1,\dots,T\}$ as

$$\left\{ (\hat{s}_t, a_t)_{t=0}^{t'-1} \in \mathcal{E}_j^{t'} \;\middle|\; \sum_{t=0}^{t'-1} \mathbb{H}(\hat{S}_{t+1}) \leq \lambda_2 \right\}, \tag{25}$$

which we use throughout the paper when referring to model rollouts. The authors of [23] provide heuristics for determining suitable values of λ_1 and λ_2 based on the training data of the current model.

We consider leaving $\mathcal{E}_j$ a failure, forcing the safety filter to stay in the sufficiently certain set. When the accumulation threshold is triggered, we truncate the rollout since longer rollouts are unreliable but we do not consider this a failure.

We proceed with filter learning in the current Dyna-SAuR iteration.

A.2 Filter Learning Problem

A.2.1 Filter Policy Learning

To learn the filter μ_j in iteration j of Dyna-SAuR, we require the current model $\hat{p}_j$ and the control policy π_{j-1} from the previous iteration. The learning procedure for the filter is described in Algorithm 4. We adapt the recent model-free RL algorithm FastTD3 [38], which has demonstrated strong performance by fully exploiting parallelization in TD3. FastTD3 employs massively parallelized environments, large batch sizes, n-step returns, multiple exploration noise scales, and a distributional critic.

Since the distributional critic improves sample efficiency at the cost of increased wall-clock runtime, we adopt all components of FastTD3 except the distributional critic. This design choice is motivated

[9] We use $\hat{S}_{t+1}$ for the Infoprop state $\tilde{S}_{t+1}$ in the Infoprop paper [23].

Algorithm 4 Filter Policy Learning

input $\hat{p}_j, \pi_{j-1}, \mu_\phi, \mu_{\bar{\phi}}, Q^{\mathrm{SF}}_{\theta_1}, Q^{\mathrm{SF}}_{\bar{\theta}_1}, Q^{\mathrm{SF}}_{\theta_2}, Q^{\mathrm{SF}}_{\bar{\theta}_2}, \mathcal{D}^{\mu,\hat{p}} = \emptyset$
  Randomly initialize ϕ, θ_1, θ_2 and $\bar{\phi} \leftarrow \phi, \bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$
  for $\mathrm{RO}^{\mathrm{SF}}$ model-based rollouts **do**
    $\sigma_1 \sim \mathbf{U}(\sigma_{\min}, \sigma_{\max})$ *Sample exploration scale [38]*
    $\beta(s_t)$ is either $\pi_{j-1}(s_t) + \sigma_2 \cdot \epsilon^\beta_t$ or $\sigma_3 \cdot \epsilon^\beta_t$ with $\epsilon^\beta_t \sim \mathrm{PN}^\beta$ *Choose between stochastic performance policy and pink noise process as behavior policy*
    $\hat{s}_0 \sim \rho^{\mathrm{SF}}_0$ *Sample initial state from modified starting state distribution introduced in Section 5.4*
    while $t < T$ and $\mathbb{H}(\hat{S}_t) \leq \lambda_1$ and $\sum_{t'=0}^{t} \mathbb{H}(\hat{S}_{t'}) \leq \lambda_2$ and $\hat{s}_t \in \mathcal{S}_{\mathrm{S}}$ **do**
      $a_t \sim \beta(\hat{s}_t)$ *Sample potentially unsafe action from behavior policy*
      $u_t \sim \mu_\phi(\hat{s}_t) + \sigma_1 \cdot \epsilon^\mu_t$ with $\epsilon^\mu_t \sim \mathrm{PN}^\mu$ *Sample hyperplane action exploring with pink noise*
      $u_t \leftarrow u_t \,/\, \max(1, \|u_t\|_2)$ *Clip to unit Euclidean ball*
      $a^{\mathrm{V}}_t = \kappa(a_t, w_t, b_t)$ with $(w_t, b_t) = h(u_t)$ *Potentially filter behavior policy according to* (7)
      $\hat{s}_{t+1} \sim \hat{p}(\cdot|\hat{s}_t, a^{\mathrm{V}}_t)$ *Sample next state via the Infoprop rollout [23]*
      $r^{\mathrm{SF}}_{t+1} = r^{\mathrm{SF}}(\hat{s}_t, u_t)$ *Get next reward according to* (15)
      $\mathcal{D}^{\mu,\hat{p}} \leftarrow \mathcal{D}^{\mu,\hat{p}} \cup \{(\hat{s}_t, u_t, \hat{s}_{t+1}, r^{\mathrm{SF}}_{t+1})\}$ *Store transition in filter replay buffer*
    end while
  end for
  for update steps **do**
    Sample mini-batch of N transitions from $\mathcal{D}^{\mu,\hat{p}}$
    $y = \sum_{k=t}^{t+n-1} \gamma^{k-t} r^{\mathrm{SF}}_{k+1} + \gamma^n \min_{i\in\{1,2\}} Q^{\mathrm{SF}}_{\bar{\theta}_i}(\hat{s}_{t+n}, \mu_{\bar{\phi}}(\hat{s}_{t+n}) + \sigma_4 \cdot \epsilon)$ with $\epsilon \sim \mathcal{N}(0, I)$ *Compute the n-step TD target y*
    $\theta_i \leftarrow \arg\min_{\theta_i} \frac{1}{N} \sum (y - Q^{\mathrm{SF}}_{\theta_i}(\hat{s}_t, u_t))^2$ for $i \in \{1, 2\}$ *Update the Q-functions by minimizing the squared TD error*
    if policy update frequency **then**
      $\phi \leftarrow \arg\max_\phi \frac{1}{N} \sum (Q^{\mathrm{SF}}_{\theta_1}(\hat{s}_t, \mu_\phi(\hat{s}_t)) - c\|\mu_\phi(\hat{s}_t)\|_2)$ *Update policy by maximizing Q-value while penalizing excessive filtering* (18)
      $\bar{\theta}_i \leftarrow \bar{\tau}\theta_i + (1 - \bar{\tau})\bar{\theta}_i$ for $i \in \{1, 2\}$ *Polyak averaging*
      $\bar{\phi} \leftarrow \bar{\tau}\phi + (1 - \bar{\tau})\bar{\phi}$ *Polyak averaging*
    end if
  end for
output μ_ϕ

by the fact that sample complexity is not critical in our setting, as we rely on parallel rollouts from copies of the learned model rather than parallel simulation of the real environment.

Each filter learning phase is initialized from scratch, using newly initialized Q-functions $Q^{\mathrm{SF}}_{\theta_1}$ and $Q^{\mathrm{SF}}_{\theta_2}$, as well as a newly initialized policy μ_ϕ, together with their corresponding target networks $Q^{\mathrm{SF}}_{\bar{\theta}_1}$, $Q^{\mathrm{SF}}_{\bar{\theta}_2}$, and $\mu_{\bar{\phi}}$. The filter replay buffer $\mathcal{D}^{\mu,\hat{p}}$ is also reset at the beginning of each learning phase.

In parallel, we perform $\mathrm{RO}^{\mathrm{SF}}$ parallel model rollouts. For each rollout, a noise scale is sampled uniformly as $\sigma_1 \sim \mathbf{U}(\sigma_{\min}, \sigma_{\max})$ [38]. For each rollout, the behavior policy is chosen with equal probability to be either a pink-noise process or the control policy augmented with pink-noise exploration. Rollouts are terminated if $(\hat{s}_t, a^{\mathrm{V}}_t) \notin \mathcal{S}_{\mathrm{S}}(\mathcal{E}_{j-1})$. Rollouts are truncated if the maximum episode length is reached or if the accumulated information loss is over the threshold as in (25).

Following the diagram in Figure 8 and Algorithm 4, in each model rollout a potentially unsafe action a_t proposed by the behavior policy is sampled to evaluate the exploratory hyperplane action u_t. This action is filtered by solving the quadratic program defined in (7). The resulting filtered action is then applied to the model via an Infoprop rollout [23] to sample the next state $\hat{s}_{t+1}$.

The reward function provides a positive signal when the filter agent keeps the state within the intersection of the safe and certain sets, and otherwise penalizes the agent according to (15). The resulting transition is stored in the filter replay buffer. After each step across all model rollouts, the Q-functions and policy are trained using FastTD3. A large mini-batch is sampled to compute the n-step TD target y, which is then used to update the Q-functions by minimizing the squared TD error. Delayed policy updates are performed, where we introduce the action regularization term in (18) to penalize excessive filtering. As in FastTD3, Polyak averaging is used for updating the target networks.

After a fixed number of steps we start the evaluation of the current filter μ_j.

A.2.2 Filter Evaluation

The goal of the filter evaluation is to determine if the RL filter training is finished. To this end, we perform model rollouts using the greedy filter policy μ and evaluate how long the behavior policy, defined as in the previous section, can be kept safe. Each evaluation model rollout is initialized from

Algorithm 5 Filter Evaluation

input $\hat{p}_j, \pi_{j-1}, \mu_j$
  for number of evaluations **do**
    $\beta(s_t)$ is either $\pi_{j-1}(s_t) + \sigma_2 \cdot \epsilon_t^\beta$ or $\sigma_3 \cdot \epsilon_t^\beta$ with $\epsilon_t^\beta \sim \mathrm{PN}^\beta$ *Choose between stochastic performance policy and pink noise process as behavior policy*
    $\hat{s}_0 \sim \rho_0^{\mathrm{EVAL}}(\cdot)$ *Sample initial state*
    while $t < T$ and $\mathbb{H}(\hat{S}_t) \leq \lambda_1$ and $\sum_{t'=0}^{t} \mathbb{H}(\hat{S}_{t'}) \leq \lambda_2$ and $\hat{s}_t \in \mathcal{S}_{\mathrm{S}}$ **do**
      $a_t \sim \beta(\hat{s}_t)$ *Sample potentially unsafe action from behavior policy*
      $u_t = \mu_j(\hat{s}_t)$ *Sample deterministic hyperplane action*
      $a_t^{\mathrm{V}} = \kappa(a_t, w_t, b_t)$ with $(w_t, b_t) = h(u_t)$ *Potentially filter behavior policy according to* (7)
      $\hat{s}_{t+1} \sim \hat{p}(\cdot|\hat{s}_t, a_t^{\mathrm{V}})$ *Sample next state via the Infoprop rollout [23]*
    end while
  end for
output $\mathcal{T}_{\mathrm{F}}, \mathcal{T}_{\mathrm{G}}$ *Update starting states*

a starting state sampled from $\rho_0^{\mathrm{EVAL}}(\cdot)$, the evaluation starting-state distribution, which is defined as a mixture of the control starting-state distribution ρ_0 and states from $\mathcal{T}_{\mathrm{G}}$.

We compute the average episode length across evaluation rollouts and terminate filter training if this metric does not improve compared to previous evaluations. We adopt this heuristic to make multiple RL filter training runs feasible within a reasonable time frame.

In addition to deciding when to end filter training and proceed to control learning, we update the filter starting-state distribution ρ_0^{SF} after each evaluation. Parts of each terminated trajectory are added to the failure set $\mathcal{T}_{\mathrm{F}}$, while parts of each truncated trajectory are added to $\mathcal{T}_{\mathrm{G}}$. We utilize the concept of maximum time to failure, which describes the maximum number of steps for which a safe state in the unviability kernel can be kept safe before it inevitably becomes unsafe. For most systems, this quantity is infeasible to compute exactly. We therefore design a heuristic that approximates the maximum time to failure using an interval defined by two hyperparameters, $[\mathrm{MTF}_{\mathrm{early}}, \mathrm{MTF}_{\mathrm{late}}]$. This interval is intended to capture the range in which the true maximum time to failure is expected to lie.

Given a terminated trajectory τ_{F} with terminal time T_{f}, we sample $\mathrm{MTF}_{\mathrm{num}}$ states from

$$(\hat{s}_{T_{\mathrm{f}}-\mathrm{MTF}_{\mathrm{late}}}, \ldots, \hat{s}_{T_{\mathrm{f}}-\mathrm{MTF}_{\mathrm{early}}}) \tag{26}$$

and add them to $\mathcal{T}_{\mathrm{F}}$ for each failed rollout. The motivation behind this heuristic is to sample training starting states in the vicinity of failure regions of the filter that can still be corrected. By sampling in this manner, we heuristically select states from the viability kernel.

From the truncated trajectories τ_{G}, we aim to overrepresent those that yield high returns with respect to the control reward r, as an overly conservative safety filter would be particularly harmful in these regions for the subsequent control policy learning problem. At the same time, to ensure that selected starting states are viable, we apply a heuristic that only selects high-return trajectories whose Q-values, used here as proxies for the expected time to failure, exceed a predefined threshold.

After a variable number of training and evaluation rounds, the method proceeds to control policy learning.

A.3 Control Learning Problem

At the start of each control policy RL phase, we are given the filter μ_j and the model $\hat{p}_j$ from the current iteration. The training procedure for the control policy is described in Algorithm 6. Essentially, this procedure follows the Infoprop algorithm, with FastTD3 serving as the model-free RL backbone. Additionally, we incorporate recent insights from [36, 32] by penalizing the control agent for deviations from the safety-filtered action, as illustrated in Figure 8. The penalized reward is given by $\tilde{r}_{t+1} = r_{t+1} - \|a_t - a_t^V\|_2$, where the Euclidean distance measures the deviation between the control agent's proposed action and the safety-filtered action.

When applying this penalty, it is important to store the unfiltered action a_t in both the environment control buffer $\mathcal{D}^{\pi,p}$ and the model control buffer $\mathcal{D}^{\pi,\hat{p}}$. The filtered action a_t^V, which is executed in the environment, is stored in the model learning buffer $\mathcal{D}^{\hat{p},p}$. Training is terminated after a fixed number of environment interactions. The Dyna-SAuR training loop continues with the next iteration of model learning.

Algorithm 6 Control Policy Learning and Environment Data Collection

input $p, \hat{p}_j, \mu_j, \pi_\psi, \pi_{\bar{\psi}}, Q_{\vartheta_1}, Q_{\bar{\vartheta}_1}, Q_{\vartheta_2}, Q_{\bar{\vartheta}_2}, \mathcal{D}^{\pi,p} = \emptyset, \mathcal{D}^{\pi,\hat{p}} = \emptyset, \mathcal{D}^{\hat{p},p}_{j-1}$

Randomly initialize $\psi, \vartheta_1, \vartheta_2$ and $\bar{\psi} \leftarrow \psi, \bar{\vartheta}_1 \leftarrow \vartheta_1, \bar{\vartheta}_2 \leftarrow \vartheta_2$

$s_0 \sim \rho_0$

for environment steps **do**

- $a_t \sim \pi_\psi(s_t) + \sigma_2 \cdot \epsilon^\beta_t$ with $\epsilon^\beta_t \sim \mathrm{PN}^\beta$ — *Sample exploratory control action*
- $u_t = \mu_j(s_t)$ — *Sample filter policy action*
- $a^{\mathrm{V}}_t = \kappa(a_t, w_t, b_t)$ with $(w_t, b_t) = h(u_t)$ — *Filter control action*
- $s_{t+1} \sim p(\cdot|s_t, a^{\mathrm{V}}_t)$ — *Sample next state of environment by executing filtered action*
- $r_{t+1} = r(s_t, a^{\mathrm{V}}_t, s_{t+1})$ — *Determine control reward*
- $\tilde{r}_{t+1} = r_{t+1} - \|a^{\mathrm{V}}_t - a_t\|_2$ — *Penalize filtering*
- $\mathcal{D}^{\hat{p},p} \leftarrow \mathcal{D}^{\hat{p},p} \cup \{(s_t, a^{\mathrm{V}}_t, s_{t+1}, r_{t+1})\}$ — *Add filtered action to model learning buffer*
- $\mathcal{D}^{\pi,p} \leftarrow \mathcal{D}^{\pi,p} \cup \{(s_t, a_t, s_{t+1}, \tilde{r}_{t+1})\}$ — *Add unfiltered action to environment control buffer*
- **if** model-based rollout frequency **then**
  - **for** number of model-based rollouts **do**
    - $\sigma_1 \sim \mathbf{U}(\sigma_{\min}, \sigma_{\max})$ — *Sample exploration scale [38]*
    - $\hat{s}_0 \sim \mathcal{D}^{\pi,p}$
    - **while** $t < T$ and $\mathbb{H}(\hat{S}_t) \leq \lambda_1$ and $\sum_{t'=0}^{t} \mathbb{H}(\hat{S}_{t'}) \leq \lambda_2$ **do**
      - $a_t \sim \pi_\psi(\hat{s}_t) + \sigma_1 \cdot \epsilon^\pi_t$ with $\epsilon^\pi_t \sim \mathrm{PN}^\pi$ — *Sample exploratory control action*
      - $u_t = \mu_j(\hat{s}_t)$ — *Sample deterministic hyperplane action*
      - $a^{\mathrm{V}}_t = \kappa(a_t, w_t, b_t)$ with $(w_t, b_t) = h(u_t)$ — *Potentially filter behavior policy according to* (7)
      - $\hat{s}_{t+1} \sim \hat{p}(\cdot|\hat{s}_t, a^{\mathrm{V}}_t)$ — *Sample next state via the Infoprop rollout [23]*
      - $r_{t+1} = r(\hat{s}_t, a^{\mathrm{V}}_t, \hat{s}_{t+1})$ — *Determine control reward*
      - $\tilde{r}_{t+1} = r_{t+1} - \|a^{\mathrm{V}}_t - a_t\|_2$ — *Penalize filtering*
      - $\mathcal{D}^{\pi,\hat{p}} \leftarrow \mathcal{D}^{\pi,\hat{p}} \cup \{(\hat{s}_t, a_t, \hat{s}_{t+1}, \tilde{r}_{t+1})\}$ — *Add unfiltered action to model control buffer*
    - **end while**
  - **end for**
- **end if**
- **for** update steps **do**
  - Sample mini-batch of N transitions from $\mathcal{D}^{\pi,\hat{p}} \cup \mathcal{D}^{\pi,p}$
  - $y = \tilde{r}_{t+1} + \gamma \min_{i \in \{1,2\}} Q_{\bar{\vartheta}_i}(\hat{s}_{t+1}, \pi_{\bar{\psi}}(\hat{s}_{t+1}) + \sigma_4 \cdot \epsilon)$ with $\epsilon \sim \mathcal{N}(0, I)$ — *Compute the TD target y*
  - $\vartheta_i \leftarrow \arg\min_{\vartheta_i} \frac{1}{N} \sum (y - Q_{\vartheta_i}(\hat{s}_t, a_t))^2$ for $i \in \{1, 2\}$ — *Update the Q-functions by minimizing the squared TD error*
  - **if** policy update frequency **then**
    - $\psi \leftarrow \arg\max_\psi \frac{1}{N} \sum Q_{\vartheta_1}(\hat{s}_t, \pi_\psi(\hat{s}_t))$ — *Update policy by maximizing Q-value*
    - $\bar{\vartheta}_i \leftarrow \bar{\tau}\vartheta_i + (1 - \bar{\tau})\bar{\vartheta}_i$ for $i \in \{1, 2\}$ — *Polyak averaging*
    - $\bar{\psi} \leftarrow \bar{\tau}\psi + (1 - \bar{\tau})\bar{\psi}$ — *Polyak averaging*
  - **end if**
- **end for**

end for

output $\pi_\psi, \mathcal{D}^{\hat{p},p}_j$

B Proofs

B.1 Bijection Between Parameter Space and Discriminating Hyperplanes

Lemma B.1 (Maximum Offset of a Hyperplane Intersecting the Interior of a Hypercube)**.** *Let $w \in \mathbb{R}^{n_\mathcal{A}}$ be a normalized normal vector, i.e., $\|w\|_2 = 1$. A hyperplane $\{a \in \mathbb{R}^{n_\mathcal{A}} \mid w^\top a = b\}$ intersects the interior of a hypercube $\mathcal{A} = [-1, 1]^{n_\mathcal{A}}$ if and only if*

$$|b| < \|w\|_1.$$

Proof. A hyperplane intersects the interior $(-1, 1)^{n_\mathcal{A}}$ if and only if there exists $a \in (-1, 1)^{n_\mathcal{A}}$ with $w^\top a = b$, which holds if and only if

$$b \in \left(\inf_{a \in (-1,1)^{n_\mathcal{A}}} w^\top a, \sup_{a \in (-1,1)^{n_\mathcal{A}}} w^\top a \right).$$

Since $w^\top a$ is linear and $w \neq 0$, the image of the convex open set $(-1, 1)^{n_\mathcal{A}}$ is an open interval, whose infimum and supremum equal the minimum and maximum over the closed hypercube. Therefore:

$$\sup_{a \in (-1,1)^{n_\mathcal{A}}} w^\top a = \max_{a \in [-1,1]^{n_\mathcal{A}}} w^\top a = \sum_{i=1}^{n_\mathcal{A}} |w_i| = \|w\|_1.$$

By symmetry, the infimum is $-\|w\|_1$. The hyperplane intersects the interior if and only if $b \in (-\|w\|_1, \|w\|_1)$, i.e.,

$$|b| < \|w\|_1.$$

□

Theorem B.2 (Bijection)**.** *The function h mapping each $u \in \mathcal{U}$ to a discriminating hyperplane (w, b) with $\|w\|_2 = 1$ intersecting the interior of $\mathcal{A} = [-1, 1]^{n_\mathcal{A}}$ is bijective. It is defined as $h(u_t) = (w_t, b_t)$ with*

$$w_t = \frac{u_t}{\|u_t\|_2}, \text{ and } b_t = (2 \cdot \|u_t\|_2 - 1)\|w_t\|_1.$$

Proof. Well-definedness. For any $u \in \mathcal{U} = \{u \in \mathbb{R}^{n_\mathcal{A}} \mid 0 < \|u\|_2 < 1\}$, the hyperplane $\{a \in \mathbb{R}^{n_\mathcal{A}} \mid w^\top a = b\}$, with $h(u) = (w, b)$ intersects the interior of the action space $\mathcal{A} = [-1, 1]^{n_\mathcal{A}}$.

By construction, w is a normalized normal vector with $\|w\|_2 = 1$. Since $0 < \|u\|_2 < 1$, we have

$$2\|u\|_2 - 1 \in (-1, 1),$$

and therefore

$$|b| = |2\|u\|_2 - 1| \, \|w\|_1 < \|w\|_1.$$

By Lemma B.1, these conditions are sufficient for the hyperplane to intersect $(-1, 1)^{n_\mathcal{A}}$. Given u, the construction uniquely determines both the normalized normal vector w and the offset b.

Injectivity. Suppose $h(u) = h(v)$ for $u, v \in \mathcal{U}$, i.e., $(w_u, b_u) = (w_v, b_v)$. From the equality of normal vectors:

$$\frac{u}{\|u\|_2} = \frac{v}{\|v\|_2},$$

it follows that $v = \lambda u$ for some $\lambda > 0$. From the equality of offsets and $\|w_u\|_1 = \|w_v\|_1 > 0$:

$$(2\|u\|_2 - 1)\|w_u\|_1 = (2\|v\|_2 - 1)\|w_v\|_1 \implies \|u\|_2 = \|v\|_2.$$

Combined with $v = \lambda u$, this gives $\lambda = 1$, hence $u = v$. Thus h is injective.

Surjectivity. Let (w, b) be a hyperplane intersecting $(-1, 1)^{n_\mathcal{A}}$ with normalized normal vector, i.e., $\|w\|_2 = 1$. By Lemma B.1 the absolute offset must be strictly smaller than the maximal offset $|b| < \|w\|_1$. Define

$$l = \frac{1}{2}\left(\frac{b}{\|w\|_1} + 1\right), \qquad u = w\, l.$$

Then $0 < l = \|u\|_2 < 1$ and $u \in \mathcal{U}$. Substituting u into h recovers (w, b) by construction. Hence, h is surjective.

Therefore, h is bijective. □

B.2 Expected Time to Failure

Lemma B.3 (Expected Time to Failure)**.** *The filter action value function $Q^{\mathrm{SF}}(s,u)$ of a filter policy $\mu(s)$, with $\gamma_{\mathrm{SF}} \in (0,1)$, for a reward function*

$$r^{\mathrm{SF}}(s_t,u_t) = \begin{cases} 1, & \text{if } (s_t, a_t^{\mathrm{V}}) \in \mathcal{S}_{\mathrm{S}}(\mathcal{E}), \\ -\frac{1}{1-\gamma_{\mathrm{SF}}}, & \text{otherwise} \end{cases}$$

lower bounds the expected time to failure T_{F} under the filter policy μ in the following form:

$$\mathbb{E}_{\mu}\left[T_{\mathrm{F}}|s\right] \geq \log_{\gamma_{\mathrm{SF}}}\left(\frac{1-Q^{\mathrm{SF}}(s,\mu(s))(1-\gamma_{\mathrm{SF}})}{2}\right)$$

Proof. The safety filter return G^{SF} for a trajectory terminating after T_{F} steps is given by

$$\begin{aligned} G^{\mathrm{SF}} &= \sum_{t=0}^{T_{\mathrm{F}}-1} \gamma_{\mathrm{SF}}^{t} \cdot 1 + \gamma_{\mathrm{SF}}^{T_{\mathrm{F}}}\left(-\frac{1}{1-\gamma_{\mathrm{SF}}}\right) \\ &= \frac{1-\gamma_{\mathrm{SF}}^{T_{\mathrm{F}}}}{1-\gamma_{\mathrm{SF}}} - \frac{\gamma_{\mathrm{SF}}^{T_{\mathrm{F}}}}{1-\gamma_{\mathrm{SF}}} \\ &= \frac{1-2\gamma_{\mathrm{SF}}^{T_{\mathrm{F}}}}{1-\gamma_{\mathrm{SF}}}. \end{aligned}$$

The state value function $V^{\mathrm{SF}}(s) = Q^{\mathrm{SF}}(s,\mu(s))$ [43] encodes the expected return under the policy μ

$$\begin{aligned} Q^{\mathrm{SF}}(s,\mu(s)) &= \mathbb{E}_{\mu}\left[G^{\mathrm{SF}} \mid S=s, U=\mu(s)\right] \\ &= V^{\mathrm{SF}}(s) \\ &= \mathbb{E}_{\mu}\left[G^{\mathrm{SF}} \mid s\right] \\ &= \mathbb{E}_{\mu}\left[\frac{1-2\gamma_{\mathrm{SF}}^{T_{\mathrm{F}}}}{1-\gamma_{\mathrm{SF}}}\Big| s\right] \\ &\overset{\mathrm{a}}{\leq} \frac{1-2\gamma_{\mathrm{SF}}^{\mathbb{E}_{\mu}[T_{\mathrm{F}}|s]}}{1-\gamma_{\mathrm{SF}}} \end{aligned} \tag{27}$$

[a] By Jensen's inequality applied to the concave function

Rearranging (27) yields the desired result. □

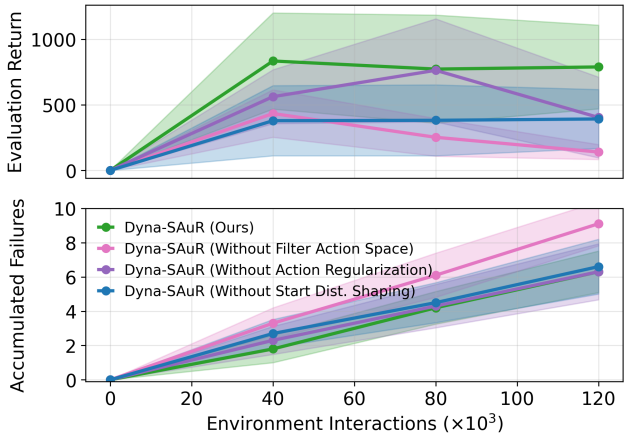


Figure 9: Ablation of Dyna-SAuR design choices for Walker. *Removing the starting state distribution shaping introduced in Section 5.4 or the action regularization introduced in Section 5.3 leads to reduced control performance and increased accumulated failures. In contrast, incorrect parameterization of the hyperplane defining action introduced in Section 5.2 prevents effective learning, indicating its central role in the method.*

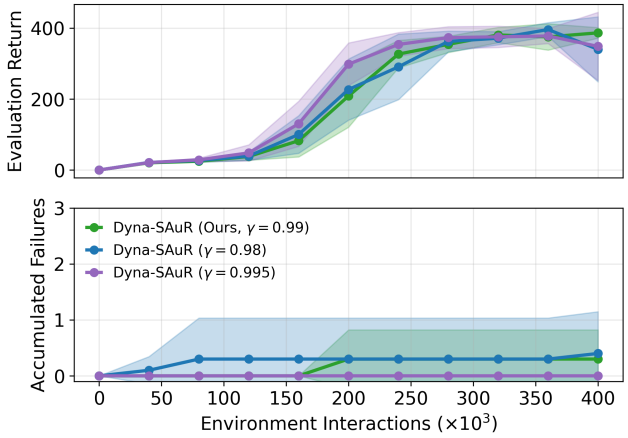


Figure 10: Ablation of the discount factor γ of Dyna-SAuR on CartPole. *The main results use* $\gamma = 0.99$ *without tuning. Decreasing (*$\gamma = 0.98$*) or increasing (*$\gamma = 0.995$*) the discount factor has no significant effect on the evaluation return or the accumulated failures.*

C Additional Results

C.1 Ablation

As visualized in Figure 9, the Walker ablation exhibits trends similar to those observed for CartPole with Dyna-SAuR depicted in Figure 7. When removing the efficient action-space parameterization of the safety filter, we observe the highest number of accumulated failures and the lowest final return. This indicates that the proposed parameterization reduces the search space and improves the effectiveness of learning safety filters with RL. When the action regularization is removed, the safety filter RL agent tends to learn overly restrictive safety filters, which leads to a lower final return. When the high-return starting states as well as the failure-preventing starting states are removed, i.e., without starting state distribution shaping, the final return is also reduced.

The main results of the paper use a discount factor of $\gamma = 0.99$, which we did not tune. To assess its influence, we ran Dyna-SAuR on CartPole with $\gamma \in \{0.98, 0.99, 0.995\}$ for 10 seeds each, with otherwise identical code and hyperparameters. The $\gamma = 0.99$ runs of this ablation were repeated with the same code version as the other values and therefore differ slightly from the main results.

As shown in Figure 10, changing γ has no significant effect on the final return in CartPole or the accumulated safety violations.

C.2 Safety Filter Analysis

Finally, we observe the safety filter μ_J after the final training iteration J for goal-reaching CartPole in Figure 11.[10] The top row shows the CartPole at the beginning, in the middle, and at the end of a trajectory in the environment. The bottom row shows μ_J in the (x, θ) subspace, with velocities given below each frame specifying the current state indicated by the black dot. Filtering the 1-dimensional action, namely applying a force in the interval $[-1, 1]$ to the cart, is achieved by reducing the interval. This is visualized via color-coding, where blue and orange indicate that only force to the right or left is considered viable, respectively, while white indicates that all actions are considered viable. We observe a clear correlation between the pole angle and the set of viable actions, which aligns with the task dynamics. Further, we see a broad set of viable actions around the upright position at low velocities, which shrinks as deflections and velocities increase.

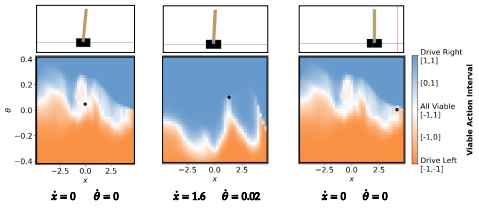


Figure 11: Final safety filter μ_J in goal-reaching CartPole. *The filter is less restrictive around the upper equilibrium and for low velocities, indicated by the comparatively large white areas, and becomes more restrictive as deflections and velocities increase.*

C.3 Model Accuracy

In Figure 12, we investigate whether the sufficiently certain set $\mathcal{E}_j$ identifies the region in which the learned dynamics model approximates the environment accurately. We consider the dynamics models $\hat{p}_1$ to $\hat{p}_5$ learned in the first five iterations of a Dyna-SAuR training run on goal-reaching CartPole.

Since the state space of CartPole is four-dimensional, we evaluate two two-dimensional slices, each on a regular grid of 81×81 states: cart position x and velocity $\dot{x}$ with the pole upright at rest ($\theta = \dot{\theta} = 0$), and pole angle θ and angular velocity $\dot{\theta}$ with the cart at rest in the center ($x = \dot{x} = 0$). The slices extend to the failure thresholds in x and θ and to the 99.5% quantile of the velocities observed during training. In every grid state, we apply the same 32 actions, sampled uniformly from the action space, to both the environment p and the model $\hat{p}_j$. For each state-action pair, we determine whether it lies in $\mathcal{E}_j$, with λ_1 chosen as during training, and compute the Euclidean distance between the true next state and the mean prediction of $\hat{p}_j$.

Within $\mathcal{E}_j$, the environment dynamics are approximated accurately by all models. Every evaluated state-action pair in $\mathcal{E}_j$ has a single-step error below 0.01. At the same time, the accurately modeled region is considerably larger than $\mathcal{E}_j$, so $\mathcal{E}_j$ is a conservative estimate of where the model can be trusted. As additional data is collected, $\mathcal{E}_j$ expands. Restricting model rollouts to $\mathcal{E}_j$ thus ensures that the filter and control policies are trained on accurate model data, while the growth of $\mathcal{E}_j$ allows for an increasingly less restrictive filter, consistent with the rising returns in Figure 5.

[10] Video of the filter learning process: `https://data-science-in-mechanical-engineering.github.io/Dyna-SAuR/`

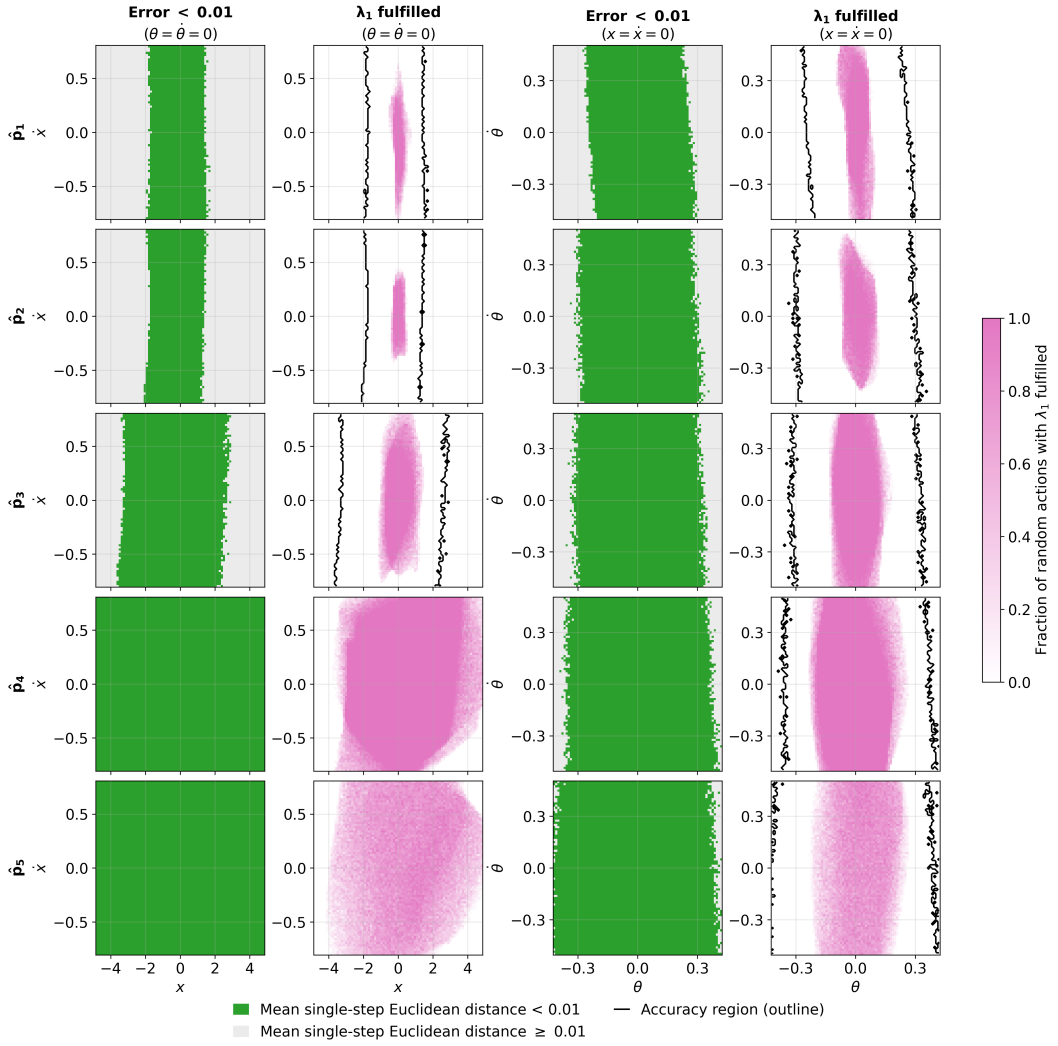


Figure 12: Single-step accuracy of the Dyna-SAuR dynamics models $\hat{p}_1$ to $\hat{p}_5$ (rows) on goal-reaching CartPole, evaluated on two slices of the state space: cart position and velocity with the pole upright at rest (left) and pole angle and angular velocity with the cart at rest (right). Green regions mark states where the Euclidean distance between the predicted and the true next state, averaged over 32 uniformly sampled actions, is below 0.01. The black line outlines this region. Pink shows the fraction of these actions for which $(s, a) \in \mathcal{E}_j$. In all iterations, the sufficiently certain set $\mathcal{E}_j$ lies within the accurately modeled region, i.e., as long as model rollouts stay in $\mathcal{E}_j$, the environment dynamics are approximated accurately.

D Implementation Details

D.1 Initial Data

Since safety without any prior knowledge is an ill-defined problem [4], we assume the availability of some initial data to train an initial dynamics model. In our experiments, this initial data is generated using a stabilizing controller. Each action produced by the prior controller is perturbed with pink noise and, at every second step, it is replaced by a uniformly sampled action to improve action-space coverage and enable reliable uncertainty estimation.

For CartPole, we design a standard equilibrium-stabilizing Linear Quadratic Regulator (LQR), which is available in our codebase (`https://github.com/Data-Science-in-Mechanical-Engineering/Dyna-SAuR`). For Walker, we train an RL agent to step forward and maintain balance. We use a partially trained version of this agent to generate initial data. Videos of the controllers used to collect initial data are provided in the supplementary material.

The offline data collection phase is explicitly decoupled from the Dyna-SAuR algorithm. Initial data is collected off-policy without safety constraints and may be replaced by logged data from any behavior policy. During safe learning, Dyna-SAuR does not utilize the controllers, which have generated the data.

Dyna-SAuR requires sufficient initial data to obtain a locally accurate dynamics model, which is necessary to derive an initial set of safe policies induced by the learned safety filter. For CartPole, 30,000 environment interactions suffice due to the one-dimensional action space. For Walker, 4,000,000 interactions are required, as the hyperplane-based safety filter must certify model accuracy over halfspaces of the six-dimensional action space at each state. This necessitates dense data coverage of the action space, but only locally in the state space. Despite this data requirement, as shown in Fig. 6, Dyna-SAuR generalizes beyond the support of the initial data distribution and learns a walking gait absent from the offline dataset.

D.2 Safety Constraints

CartPole. The state is $s = (x, \dot{x}, \theta, \dot{\theta})^\top \in \mathcal{S}$, where x is the cart position and θ is the pole angle. A state is considered a failure state if either the cart leaves the track bounds or the pole exceeds the allowed angular range. Formally, the failure set is defined as:

$$\mathcal{S}_{\mathrm{F}}^{\mathrm{CartPole}} = \Big\{ s \in \mathcal{S} \;\Big|\; x < -4.8 \;\vee\; x > 4.8 \;\vee\; \theta < -0.42 \;\vee\; \theta > 0.42 \Big\}.$$

The task objective is to reach the goal position $x = 4.1$, which necessitates fast and agile cart motion. Consequently, the agent is incentivized to operate near the boundary of the admissible pole angle and to approach the track bound aggressively before decelerating within a narrow positional margin. This renders the goal-reaching variant strictly more demanding than the standard balancing task: the agent must simultaneously maximize locomotion speed and satisfy tight state-wise safety constraints, creating an inherent tension between task performance and constraint satisfaction throughout the trajectory.

Walker. A state $s \in \mathcal{S}$ is considered a failure state if the Walker's vertical position z (measured as the z-coordinate of the rendered black marker shown in Fig. 4b) leaves the admissible interval, or if the torso angle θ exceeds its permitted range. Formally, the failure set is defined as:

$$\mathcal{S}_{\mathrm{F}}^{\mathrm{Walker}} = \Big\{ s \in \mathcal{S} \;\Big|\; z < 0.8 \;\vee\; z > 1.2 \;\vee\; \theta < -0.9 \;\vee\; \theta > 0.9 \Big\}.$$

We use the original task objective. However, the upper bound on z restricts the admissible jumping height relative to the unconstrained Walker task, thereby reducing the achievable cumulative return compared to the standard benchmark. Crucially, the constraint induces a non-trivial temporal credit assignment problem. Once the agent is airborne, no corrective action can prevent a constraint violation that results from a prior locomotion decision. Safe behavior therefore requires anticipatory control. The agent must regulate its vertical momentum sufficiently early to avoid the failure set $\mathcal{S}_{\mathrm{F}}^{\mathrm{Walker}}$ throughout the trajectory.

D.3 Hyperparameters

All experiments were conducted over 10 independent random seeds, with hyperparameters chosen to ensure a fair comparison across methods. For all environments, all episodes are truncated after 500 steps.

D.3.1 Hyperparameters Dyna-SAuR

The hyperparameters for Dyna-SAuR are shown in Table 1.

D.3.2 Hyperparameters of the Infoprop Baseline

As shown in Table 2, we use the standard Infoprop [23] hyperparameters for the baseline. The only exception is that we increase the retraining interval (measured in environment steps) at which the model is retrained. This change is made to keep model training computationally feasible due to the relatively large amount of initial data in the Walker environment, and to match the retraining interval used by the Dyna-SAuR model.

D.3.3 Hyperparameters PPO-Lagrangian

As shown in Table 3, we use standard PPO-Lagrangian [37] hyperparameters for the benchmark.

D.3.4 Hyperparameters DH-RL

As shown in Table 4, we use standard DH-RL [30] hyperparameters for the benchmark.

D.4 Computation Resources

Parts of the code were adapted from the Infoprop implementation [23], available at `https://github.com/data-science-in-mechanical-engineering/infoprop`, published under the MIT license. Each run was conducted on a single NVIDIA H100 GPU with 8 CPU cores at 2.1 GHz and 128 GB of RAM. Each experimental configuration was evaluated across 10 random seeds and 4 algorithm variants (1 primary and 3 ablations). Experiments on the Walker environment required approximately 24 GPU hours per seed-variant pair, and those on CartPole approximately 8 GPU hours, yielding a total computational budget of 1280 GPU hours for the final experiments reported in this work. A single filter training run takes about 40 minutes for CartPole and about 7 hours for Walker. The baselines required the following training times: PPO-Lag took 3 h for CartPole and 26 h for Walker; DH-RL took 7 h for CartPole and 10 h for Walker; and Infoprop took 14 h for CartPole and 17 h for Walker.

Including all development and benchmark runs, the full budget amounted to 9791 GPU hours. The source code is released under the MIT license.

Table 1: Hyperparameters for Dyna-SAuR on Walker and CartPole environments.

Hyperparameter	**CartPole**	**Walker**
Model Learning		
Ensemble size E	7	7
Number of hidden layers	4	4
Number of hidden neurons	200	200
Learning rate	0.0006	0.0006
Weight decay	0.0007	0.0007
Patience for early stopping	10	10
Batch size N^{Model}	256	256
Safety Filter Learning		
Number of hidden layers actor and critic	2	2
Number of hidden neurons actor and critic	256	256
Number of model rollouts $\mathrm{RO}^{\mathrm{SF}}$	100	500
Minimum noise scale $\sigma_{\min}$	0.001	0.001
Maximum noise scale $\sigma_{\max}$	0.3	0.2
Accurate quantile ζ_1	0.99	0.99
Exceptionally accurate quantile ζ_2	0.01	0.01
Learning rate	0.0003	0.0003
Buffer size filter buffer $\mathcal{D}^{\mu,\hat{p}}$	1,000,000	1,000,000
Batch size N	100,000	100,000
Polyak averaging factor $\bar{\tau}$	0.001	0.001
Policy update frequency	2	2
Action regularization factor c	0.1	1
Pink noise process scale σ_3	0.33	0.33
Policy exploration scale σ_2	0.1	0.1
TD3 smoothing scale σ_4	0.003	0.003
Discount Factor γ^{SF}	0.99	0.99
Factor control starting states ν_1	0.05	0.1
Factor failed rollouts ν_2	0.3	0.45
Factor high return rollouts ν_3	0.65	0.45
Mixing factor ν_4	0.5	0.5
Mixing factor ν_5	0.5	0.5
Number of evaluation rollouts	2000	1000
Maximum time to failure early $\mathrm{MTF}_{\mathrm{early}}$	10	20
Maximum time to failure late $\mathrm{MTF}_{\mathrm{late}}$	50	100
Number of states $\mathrm{MTF}_{\mathrm{num}}$	10	5
Control Policy Learning		
Number of hidden layers actor and critic	2	2
Number of hidden neurons actor and critic	256	256
Number of hidden layers	4	4
Minimum noise scale $\sigma_{\min}$	0.001	0.001
Maximum noise scale $\sigma_{\max}$	0.2	0.2
Accurate quantile ζ_1	0.99	0.99
Exceptionally accurate quantile ζ_2	0.01	0.01
Learning rate	0.0003	0.0003
Polyak averaging factor $\bar{\tau}$	0.001	0.001
Policy update frequency	2	2
Policy exploration scale σ_2	0.1	0.1
TD3 smoothing scale σ_4	0.003	0.003
Discount Factor γ	0.99	0.99
Number of environment interactions per Dyna-SAuR iteration	40,000	40,000

Table 2: Hyperparameters for Infoprop on Walker and CartPole environments.

Hyperparameter	**CartPole**	**Walker**
Model Learning		
Ensemble size E	7	7
Number of hidden layers	4	4
Number of hidden neurons	200	400
Learning rate	0.001	0.0006
Weight decay	0.0002	0.0007
Patience for early stopping	9	9
Retrain interval	40,000	40,000
Model Rollouts		
Accurate quantile ζ_1	0.99	0.99
Exceptionally accurate quantile ζ_2	0.01	0.01
Scaling factor ξ	1	1
Rollout interval	250	250
Rollout batch size	100,000	100,000
SAC Agent		
Number of hidden neurons	1,024	1,024
Number of hidden layers	2	2
Learning rate	0.005	0.0002
SAC target entropy	0	-7
Target update interval	5	4
Update steps G	10	10

Table 3: Hyperparameters for PPO-Lagrangian on Walker and CartPole environments.

Hyperparameter	**CartPole**	**Walker**
Number of hidden neurons	64	128
Number of hidden layers	2	2
Steps per epoch	400	4000
Gamma	0.99	0.99
Lambda	0.97	0.97
Cost gamma	0.97	0.97
Cost lambda	0.97	0.97
Penalty learning rate	0.05	0.05
Target Kullback-Leibler divergence	0.01	0.01
Value function learning rate	0.001	0.001

Table 4: Hyperparameters for DH-RL on Walker and CartPole environments.

Hyperparameter	**CartPole**	**Walker**
Safety Filter PPO Agent		
Number of hidden neurons	64	128
Number of hidden layers	2	2
Steps per epoch	400	4000
Gamma	0.99	0.99
Lambda	0.97	0.97
Target Kullback-Leibler divergence	0.01	0.01
Value function learning rate	0.001	0.001
Policy learning rate	0.0003	0.0003
Environment Steps for Safety Filter Training	30000	4000000
Clip Ratio	0.2	0.2
Control PPO Agent		
Number of hidden neurons	64	128
Number of hidden layers	2	2
Steps per epoch	400	4000
Gamma	0.99	0.99
Lambda	0.97	0.97
Target Kullback-Leibler divergence	0.01	0.01
Value function learning rate	0.001	0.001
Policy learning rate	0.0003	0.0003
Clip Ratio	0.2	0.2

E Definitions and Terminology

E.1 Discussion of the Filter Policy Learning Problem

We write the filter policy learning problem

$$\mu^* = \arg\max_{\mu \in \mathrm{M}} \{\mathrm{Img}(\kappa_\mu^\pi) \subseteq \Pi_\mathrm{V}(\mathcal{E})\}, \tag{28}$$

as a shorthand notation for finding a filter policy $\mu^* \in \mathrm{M}$ such that

$$\mathrm{Img}(\kappa_{\mu^*}^\pi) \subseteq \Pi_\mathrm{V}(\mathcal{E}) \text{ and } \nexists \mu \in \mathrm{M} \text{ with } \mathrm{Img}(\kappa_{\mu^*}^\pi) \subsetneq \mathrm{Img}(\kappa_\mu^\pi). \tag{29}$$

E.2 Safety and Viability

The term "safe states" is used inconsistently across the safety literature. For instance, the authors of [14] employ it to denote what our work terms viable states, whereas the authors of [11] use the term consistently with the definition adopted in this work.

Figure 13 illustrates the terminology adopted in this work. As engineers, we define unsafe states $\mathcal{S}_\mathrm{F}$ as states the agent must never reach, e.g., states in which the robot has fallen and lies on the ground. Such failure states are typically easy to define. The set of safe states $\mathcal{S}_\mathrm{S}$ is the complement of the set of unsafe states $\mathcal{S}_\mathrm{F}$. Consequently, some safe states inevitably lead to failure, e.g., a robot that stumbles such that it cannot recover but does not yet lie on the ground. Therefore, the more informative distinction is between viable states $\mathcal{S}_\mathrm{V}$, from which the system can remain in the set of safe states indefinitely under a suitable control policy, and unviable states $\mathcal{S}_\mathrm{U}$, which are either already unsafe or from which failure is inevitable. Distinguishing viable from unviable states is the non-trivial problem that Dyna-SAuR addresses.

Figure 13 provides a minimal illustrative example of the distinction. The car on the slope is safe but unviable, as it enters $\mathcal{S}_\mathrm{F}$ under any control sequence, since the engine torque is insufficient to climb the slope.

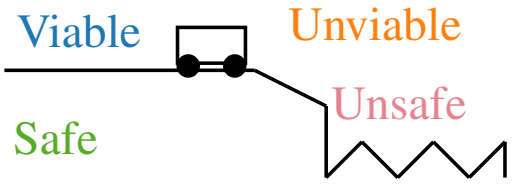


Figure 13: Safety vs. viability terminology illustrated by a car on a sloped track. *From an engineering perspective, unsafe states (red) are defined as the undesired states in which the car has crashed; their complement is the set of safe states (green). On the slope, insufficient torque prevents the car from returning, so regardless of the control sequence applied, the car inevitably enters the unsafe set in finite time. These states and the unsafe states are defined as unviable (orange). Their complement, i.e., states from which a control sequence exists that can keep the system safe, are viable (blue).*

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the core contributions: Dyna-SAuR, a safe RL algorithm requiring only initial environment data and a failure criterion, achieving significant reductions in safety violations during training, empirically validated on safety-constrained variants of CartPole and MuJoCo Walker.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 8.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Complete proofs of all theoretical results are provided in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The code and a README with full reproduction instructions for each experiment in the submission are provided at `https://github.com/Data-Science-in-Mechanical-Engineering/Dyna-SAuR`. Pseudocode and additional algorithmic details are provided in Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code and a README with full reproduction instructions for each experiment in the submission are provided at `https://github.com/Data-Science-in-Mechanical-Engineering/Dyna-SAuR`. Pseudocode and additional algorithmic details are provided in Appendix A.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training details and selected hyperparameters are described in Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All experiments are repeated across ten independent random seeds. Shaded regions in all figures denote 99% confidence intervals computed via the normal approximation across seeds.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Full details of the experimental computational resources are reported in Appendix D.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Dyna-SAuR addresses training-time safety in reinforcement learning, a foundational research problem with potential positive societal implications for deploying reliable autonomous systems. All experimental code is released openly, supporting reproducibility and transparency in accordance with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: Addressing exploration safety, Dyna-SAuR contributes to the general applicability of RL methods. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted in the paper.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: We are not aware of any such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Original authors of third-party code used in this work are credited, and applicable licensing terms are stated, in Appendix D.4.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The codebase is thoroughly documented, with newly contributed components explicitly described.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs play no role in the core methodological development of this work. Their use is strictly limited to spelling correction and text refinement of this work.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.