SafePO: A Benchmark for Safe Policy Optimization

Author Name
Affiliation
email@example.com

Abstract

Safe reinforcement learning (Safe RL) tackles decision-making problems with safety constraints. Despite the influx of attention in this field, there is a lack of commonly recognized safe RL algorithm benchmark. This is because many safe RL methods' code is unavailable, and new methods often come with new testing tasks. As a result, researchers suffer from incorrect implementations, unfair comparisons, and misleading conclusions. This study offers a unified, highly-optimized, and extensible safe RL algorithm benchmark—(`SafePO`[1])—which benchmarks popular safe policy learning algorithms across a list of typical environments. Specifically, we start by standardizing the problem of safe reinforcement learning by solving constrained Markov decision processes (CMDP). Then, we provide implementations for CMDP solutions, covering both constrained policy optimization type methods and Lagrangian type methods. Our implementations in `SafePO` are highly efficient because learners can collect samples in parallel and synchronize their policy gradients on different physical CPU cores. We test them on three safety environment suites and the newly proposed Safety DexterousHands. Based on the benchmark results, we substantiate the correctness of our implementation and derive new insights by disclosing the interplay of different attributes on safety performance.

1 Introduction

Safe policy learning is critical in real-world reinforcement learning applications where dangerous decisions are undesirable. For example, a robot agent should avoid taking actions that irreversibly damage its hardware [Ray *et al.*, 2019; Dulac-Arnold *et al.*, 2019]. Due to its importance, the community has been actively researching safe policy learning (e.g.,[Alshiekh *et al.*, 2018; Stooke *et al.*, 2020; Gu *et al.*, 2021; Yuan *et al.*, 2021; Gronauer, 2022; Yang *et al.*, 2022; Liu *et al.*, 2022]). However, most of the existing work mainly focuses on algorithm design. Among these works, either the authors did not publish the source code (e.g., P3O [Zhang *et al.*, 2022]), or the algorithms were implemented using different frameworks(e.g., PCPO [Yang *et al.*, 2020] in Theano [Al-Rfou *et al.*, 2016], CPPO-PID [Stooke *et al.*, 2020] in PyTorch), with divergent approaches (FOCOPS [Zhang *et al.*, 2020] does not parallelize sample collection while others do), and on separate tasks (FOCOPS is tested solely on MuJoCo-Velocity [Todorov *et al.*, 2012] and CPPO-PID solely on Safety-Gym [Ray *et al.*, 2019]). While there exists safety-starter-agents [Ray *et al.*, 2019] as a publicly available collection of algorithms, it was implemented using TensorFlow1, required old hardware and system, lacked recent updates, and was no longer maintained. As a result, the safe RL community has experienced serious difficulty reproducing the experimental results, comparing algorithms fairly, and deriving correct insights. An open-source, standardized algorithm benchmark for algorithm verification and empirical study is desperately needed.

To address the undesirable research status of safe RL, in this work, we re-implement eight safe policy optimization algorithms using PyTorch in a unified, highly-optimized, and extensible framework, called `SafePO`, and provide support to four environments and 30 safety tasks included (see Appendix B.1 for details). The soundness, correctness, efficiency, and extensibility of `SafePO` are experimentally substantiated in later sections. Based on our work, researchers can now conduct experiments under the same conditions or quickly verify their ideas by implementing only their unique innovations and utilizing all other functionalities directly from `SafePO` framework. Concretely, our contributions can be stated as follows:

- **Comprehensive algorithm implementation**: Based on original papers or public code base, we re-implement eight algorithms (CPO [Achiam *et al.*, 2017], PCPO [Yang *et al.*, 2020], FOCOPS [Zhang *et al.*, 2020], P3O [Zhang *et al.*, 2022], PPO-Lag[2] [Ray *et al.*, 2019], TRPO-Lag [Ray *et al.*, 2019], CPPO-PID [Stooke *et al.*, 2020], and IPO [Liu *et al.*, 2020]

[1] Our code and instructions for its use are included in the supplementary material.

[2] In this paper, we use PPO-Lag, TRPO-Lag, MAPPO-Lag as abbreviations of PPO-Lagrangian, TRPO-Lagrangian, and MAPPO-Lagrangian.

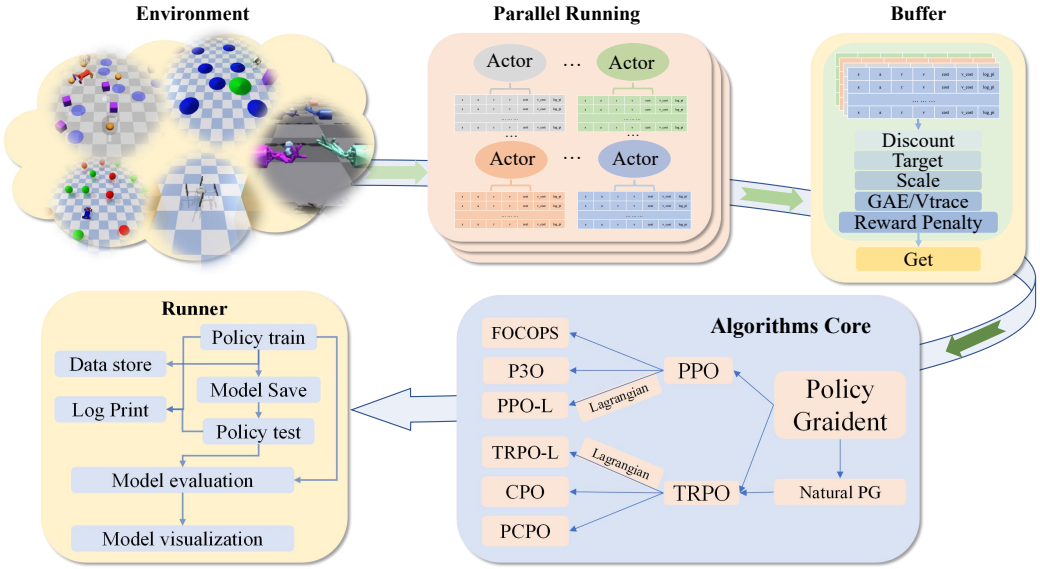

Figure 1: The overall architecture of `SafePO`. By abstraction, we modularize the framework into environments interaction, parallel sample collection, buffer storage, computation, algorithm core update, and visualization and connect them with the main data flow. In the algorithm cores, the algorithms are implemented in a single unified tree-like inheritance structure. `SafePO` allows easy extension and replacement to every module for experimental validation.

), covering major safe policy optimization algorithms. The correctness of our implementation is experimentally substantiated, as we attain comparable results to the original work. We will ensure long-term maintenance of `SafePO` benchmark, keeping adding new algorithms, expanding its scope, and supporting new environments, as our mission is to contribute to safe RL research.

- **Unified, highly-optimized, and extensible framework**: We implement major algorithms and support all popular environments in a single well-designed framework, which is unique and unprecedented. Specifically, `SafePO` is implemented using PyTorch and supports the newest hardware architecture, enabling efficient experiments and fair comparison. Beyond that, we also support multi-core CPU parallelization, substantially accelerating the training process. Finally, we have done maximum abstraction and encapsulation, deriving a similar model structure and update paradigm, thus enabling code reuse, ensuring a clean code style, and making it extremely extensible.

- **Exhaustive environments support and experiments testing**: Currently, `SafePO` supports MuJoCo-Velocity [Todorov *et al.*, 2012], Safety-Gym [Ray *et al.*, 2019], Bullet-Safety-Gym [Gronauer, 2022], and Safety DexterousHands, a novel safe environment first proposed in this work. In these environments, we have done exhaustive experiments with all the algorithms implemented and contributed the results, our observations, and analysis for the reference of the community.

In the following sections, we first present a survey of related work in Section 2 and discuss our contributions. Then in Section 3, we formalize safe reinforcement learning mathematically, with an introduction to the notations we use. After an overview of safe environments and safe policy optimization algorithms in Section 4 and 5, we elaborate on the design of `SafePO` and demonstrate its strengths in Section 6. Section 7 consists of extensive experiments, our observations, analysis, and recommendations of algorithm usage for different tasks. Finally, Section 8 concludes the paper. As abundant auxiliary data and explanations are accompanying the main paper, we place them in Appendix B for further reference.

2 Related Work

Safety Environments. In traditional RL, agents need to explore surrounding environments to learn optimal policies by trial and error. It is typical to train RL agents mostly or entirely in simulation, where safety concerns are minimal. However, we anticipate that challenges in simulating the complexities of the real world (e.g., human-AI collaborative control [Carlson and Demiris, 2010; Bi *et al.*, 2021]) will cause a shift towards training RL agents directly in the real world, where safety concerns are paramount. OpenAI includes safety requirements in the safety-gym [Ray *et al.*, 2019], a suite of high-dimensional continuous control environments for measuring research progress on constrained RL. Safe-control-gym [Yuan *et al.*, 2021] allows for constraint specification and disturbance injection onto a robot's inputs, states, and inertial properties through a portable configuration system. DeepMind also presents a suite of reinforcement learning environments, AI Safety Gridworlds [Leike *et al.*, 2017], illustrating various safety properties of intelligent agents. These problems include safe interruptibility, avoiding side effects, absent supervisor, reward gaming, safe ex-

ploration, as well as robustness to self-modification, distributional shift, and adversaries. Bullet-safety-gym [Gronauer, 2022] proposes a unified and standardized collection of safety environments, including 16 tasks. MuJoCo-Velocity, originally proposed in [Zhang *et al.*, 2020], consists of a series of safety tasks based on MuJoCo environment [Todorov *et al.*, 2012]. Since these safety environment suites are publicly acknowledged and widely used, `SafePO` supports Safety-Gym, Bullet-Safety-Gym, and MuJoCo-Velocity. In addition, we also propose Safety DexterousHands tasks to supplement the current work.

Safe Policy Optimization Algorithms. In the tabular case, CMDPs have been extensively studied for different constraint criteria [Beutler and Ross, 1985; Ross and Varadarajan, 1989; Kallenberg, 1983]. With the rise of deep learning, CMDPs are also moving to more high-dimensional continuous control problems. [Achiam *et al.*, 2017] proposes Constrained Policy Optimization, the first general-purpose policy search algorithm for constrained reinforcement learning with guarantees for near-constraint satisfaction at each iteration. [Yang *et al.*, 2020] utilizes a different two-step approach(i.e., first finds the policy with the maximum return, then projects this policy back into the feasible cost constraint set in terms of the minimum KL divergence.). [Zhang *et al.*, 2020] has adopted a similar idea by directly solving the constrained trust region problem using a primal-dual approach and then projecting the solution back into the parametric policy space. At the same time, researchers have made successful attempts to integrate more with traditional control theory. [Chow *et al.*, 2018; Chow *et al.*, 2019] presents a method for constructing the Lyapunov function, which guarantees constraint satisfaction during training. [Stooke *et al.*, 2020] combines PID control with Lagrangian methods, which dampens cost oscillations resulting in reduced constraint violations. Therefore, to facilitate experimental validation in safe RL research, we provide PyTorch-version re-implementations of 8 major safe policy optimization algorithms in `SafePO` at present and will keep adding more algorithms to expand its coverage.

3 Safe Reinforcement Learning

Safe reinforcement learning is formulated as a constrained Markov decision process (CMDP) [Altman, 1999], defined as $(\mathcal{S}, \mathcal{A}, \mathbb{P}, r, \rho_0, \gamma, \mathcal{C})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0(\cdot) \in \mathcal{P}(\mathcal{S})$ is the initial state distribution ($\mathcal{P}(X)$ denotes the set of probability distributions over a set X), $\gamma \in [0, 1)$ is the discount factor, and $\mathcal{C} = \{(c, b)\}$ is the constraint set, where $c : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a cost function, and b is the corresponding cost limit.

We use $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ to denote a stationary policy and use Π to denote the set of all stationary policies. Let $\tau = \{s_t, a_t, r_{t+1}, c_{t+1}\}_{t\geq 0} \sim \pi$ be a trajectory generated by π, where $s_0 \sim \rho_0(\cdot)$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$, $r_{t+1} = r(s_{t+1}|s_t, a_t)$, and $c_{t+1} = c(s_{t+1}|s_t, a_t)$. The *state value function* of π is defined as $V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1}|s_0 = s]$. The goal of reinforcement learning is to maximize the *expected total reward*, defined as $J(\pi) = \mathbb{E}_{s\sim\rho_0(\cdot)}[V_\pi(s)]$.

Code 1 Velocity constraint in MuJoCo suite.

```
Pre-step:
 next_obs, rew, done, info=env.step(act)
def velocity_cost(info):
    cost=0
    if 'y_velocity' not in info:
        cost=np.abs(info['x_velocity'])
    else:
        cost=np.sqrt(
             info['x_velocity'] ** 2
           + info['y_velocity'] ** 2
        )
    return cost
```

In safe reinforcement learning, safety constraints have to be satisfied in the first place. We define the *cost return function* as $J^c(\pi) = \mathbb{E}_{s\sim\rho_0(\cdot)}[\sum_{t=0}^{\infty} \gamma^t c_{t+1}|s_0 = s]$, and the feasible policy set $\Pi_\mathcal{C}$ as $\Pi_\mathcal{C} = \{\pi \,|\pi \in \Pi, J^c(\pi) \leq b, \forall(c, b) \in \mathcal{C}\}$. The goal of safe reinforcement learning is to learn the optimal policy $\pi_\star$ such that

$$\pi_\star = \arg\max_{\pi\in\Pi_\mathcal{C}} J(\pi). \tag{1}$$

4 Safety Environments

The purpose of this section is to introduce four safety environments that provide interesting tasks for safe reinforcement learning. Among them, three are popular safety environments, MuJoCo-Velocity [Todorov *et al.*, 2012], Safety-Gym [Ray *et al.*, 2019], Bullet-Safety-Gym [Gronauer, 2022], and the other one, called Safety DexterousHands, is newly proposed in this work, which is based on DexterousHands [Chen *et al.*, 2022]. `SafePO` provides support to these environments.

4.1 MuJoCo-Velocity

MuJoCo-Velocity consists of a series of safety tasks based on MuJoCo environment [Todorov *et al.*, 2012]. It is originally proposed in [Zhang *et al.*, 2020]. In these tasks, agents, such as Ant, Half Cheetah, and Humanoid, are trained to move faster for higher rewards while imposing a velocity constraint for safety considerations. Formally, for an agent moving on a two-dimensional plane, the cost is calculated as $c(s, a) = \sqrt{v_x^2 + v_y^2}$; for an agent moving along a straight line, the cost is calculated as $c(s, a) = |v_x|$, where v_x, v_y are the velocities of the agent in the x and y directions respectively. Code 1 shows how we add the cost.

4.2 Safety Gym

Safety Gym [Ray *et al.*, 2019] is a suite of environments for safe reinforcement learning developed by OpenAI. In these environments, an agent must navigate a cluttered environment to achieve a task. The agent receives a reward for completing the task and gets penalized for behaving unsafely. For more

details, please refer to the official documentation[3][4].

4.3 Bullet-Safety-Gym

Bullet-Safety-Gym [Gronauer, 2022] is another suite of environments that provides four agents (Ball, Car, Drone, and Ant) and four tasks (Circle, Collect, Reach, and Run). In general, the agents are rewarded for completing the tasks and penalized for leaving the safety zone, colliding with dangerous objects, exceeding a velocity threshold, etc. For more information, please refer to the official GitHub page[5].

4.4 Newly Proposed: Safety DexterousHands

In this work, we propose a new series of safety tasks based on DexterousHands [Chen *et al.*, 2022]. In these environments, there are two robotic hands (see figure 2 (c) and (d)). At the beginning of each episode, a ball falls randomly around the right hand, and the two hands must collaborate to place the ball in a given position. Since the target is out of the reach of the right hand, and the right hand cannot pass the ball to the left hand directly, a possible solution is that the right-hand grabs the ball and throws it to the left hand; the left-hand catches the ball and puts it to the target.

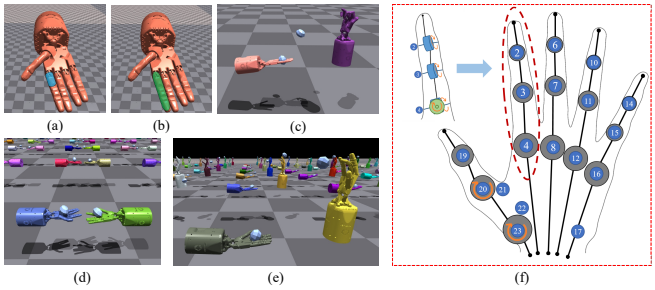

Figure 2: Explanation of Safety DexterousHands. (a) In Safety Joint tasks, the freedom of the highlighted part of the forefinger (corresponding to the freedom of joint ④) is restricted within a given range. (b) In Safety Finger tasks, the freedom of the whole forefinger (corresponding to the freedom of joints ②, ③, and ④) is restricted within a given range. (c) A figure of the ShadowHand-CatchOver2Underarm environment in DexterousHands. (d) A figure of the ShadowHandOver environment in DexterousHands. (e) GPU-based large-scale parallel sample collection and training. (f) Illustration of the joints on a dexterous robotic hand.

For timestep t, let $x_{b,t}$ be the ball's position, and $x_{g,t}$ be the goal's position. We use $d_{p,t}$ to denote the positional distance between the ball and the goal $d_{p,t} = \|x_{b,t} - x_{g,t}\|_2$. Let $d_{a,t}$ denote the angular distance between the object and the goal, and the rotational difference is $d_{r,t} = 2 \arcsin \min\{|d_{a,t}|, 1.0\}$. The reward is defined as follows,

$$r_t = \exp\{-0.2(\alpha d_{p,t} + d_{r,t})\}, \tag{2}$$

where α is a constant balance of positional and rotational rewards.

[3]https://openai.com/blog/safety-gym

[4]https://github.com/openai/safety-gym

[5]https://github.com/SvenGronauer/Bullet-Safety-Gym

Safety Joint. In these tasks, we constrain the freedom of joint ④ of the forefinger (please refer to figure 2 (a) and (f)). Without the constraint, joint ④ has freedom of $[-20°, 20°]$. The safety tasks restrict joint ④ within $[-10°, 10°]$. Let `ang_4` be the angle of joint ④, and the cost is defined as:

$$c_t = \mathbb{I}(\texttt{ang\_4} \notin [-10°, 10°]). \tag{3}$$

Safety Finger. In these tasks, we constrain the freedom of joints ②, ③, and ④ of the forefinger (please refer to figure 2: b and f). Without the constraint, joints ② and ③ have freedom of $[0°, 90°]$ and joint ④ of $[-20°, 20°]$. The safety tasks restrict joints ②, ③, and ④ within $[22.5°, 67.5°]$, $[22.5°, 67.5°]$, and $[-10°, 10°]$ respectively. Let `ang_2`, `ang_3`, `ang_4` be the angles of joints ②, ③, ④, and the cost is defined as:

$$\begin{aligned} c_t = \mathbb{I}\Big(&\texttt{ang\_2} \notin [22.5°, 67.5°], \\ &\text{or } \texttt{ang\_3} \notin [22.5°, 67.5°], \\ &\text{or } \texttt{ang\_4} \notin [-10°, 10°]\Big). \end{aligned} \tag{4}$$

Environments in Safety DexterousHands are based on IsaacGym [Makoviychuk *et al.*, 2021], which allows large-scale parallel sample collection on GPU, thus accelerating the training process substantially. Besides, Safety DexterousHands provide both single-agent and multi-agent settings. To supplement the multi-agent tasks, we also release a collection of multi-agent safe reinforcement learning algorithms accompanying the environments for quick experiments. These algorithms include HAPPO [Kuba *et al.*, 2021], IPPO [de Witt *et al.*, 2020], MACPO [Gu *et al.*, 2021], MAPPO [Yu *et al.*, 2021], and MAPPO-Lag [Gu *et al.*, 2021]. Detailed usage information can be found on supplementary material.

5 Safe Policy Optimization Algorithms

In this section, we present a brief introduction to representative safe policy optimization algorithms. Appendix A contains all the key steps of these algorithms.

CPO. [Achiam *et al.*, 2017] suggests to replace the reward objective $J(\pi)$ and the cost constraint $J^c(\pi)$ with surrogate functions, which evaluate the objective $J(\pi)$ and constraint $J^c(\pi)$ according to the samples collected by π_k. CPO updates policy as follows,

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_{\boldsymbol{\theta}}} \quad \mathbb{E}_{s \sim d^{\rho_0}_{\pi_k}(\cdot), a \sim \pi(\cdot|s)} \left[A_{\pi_k}(s,a)\right] \tag{5}$$

$$\text{s.t. } J^c(\pi_k) + \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\rho_0}_{\pi_k}(\cdot), a \sim \pi(\cdot|s)} \left[A^c_{\pi_k}(s,a)\right] \leq b \tag{6}$$

$$\bar{D}_{\mathrm{KL}}(\pi, \pi_k) = \mathbb{E}_{s \sim d^{\rho_0}_{\pi_k}(\cdot)}[\mathrm{KL}(\pi, \pi_k)[s]] \leq \delta, \tag{7}$$

where π_k is short for $\pi_{\boldsymbol{\theta}_k}$. In practice, CPO introduces convex approximations to replace the term $A_{\pi_k}(s,a)$, $A^c_{\pi_k}(s,a)$, and $\bar{D}_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}, \pi_k)$ for trust-region update.

PCPO. [Yang *et al.*, 2020] is an iterative method containing two steps: the first step performs a local reward improvement. In contrast, the second step reconciles any constraint violation by projecting the policy back onto the constraint set. For the practical implementation, PCPO inherits the key idea from TRPO [Schulman *et al.*, 2015], and CPO [Achiam *et al.*, 2017], where PCPO also uses convex approximations to the reward objective and cost constraint.

Table 1: Performance comparisons between SafePO and other benchmarks. For every environment, we run the original code and SafePO under ten random seeds and report the comparative performance of SafePO to the original work. Note that FOCOPS originally uses *discounted* total cost to update the Lagrangian multiplier, while SafePO follows OpenAI safety-starter-agents to use *undiscounted* version. In this table, we modify the original FOCOPS to compare the undiscounted performance (this is done by removing the exponential discount in line 76 of https://github.com/ymzhang01/focops/blob/main/data_generator.py, to be consistent with line 386 of https://github.com/openai/safety-starter-agents/blob/master/safe_rl/pg/run_agent.py).

	Safety Starter Agents				**FOCOPS-Volecity**				**Bullet-Safety-Gym**			
	Safexp-Point-Goal1-v0		Safexp-Car-Goal1-v0		Humanoid-v3		Ant-v3		BallCircle		CarRun	
	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
CPO	+12.1%	+1.3%	+13.5%	-1.7%	+13.4%	-1.2%	+18.8%	-1.2%	+8.2%	+2.1%	+10.8%	-1.3%
TRPO-Lag	+14.2%	-1.8%	+12.9%	+1.8%	+15.4%	-1.4%	+11.6%	-1.2%	+9.6%	-1.7%	+10.2%	-0.8%
PPO-Lag	+13.4%	-1.1%	+15.7%	+1.3%	+9.8%	-1.1%	+16.1%	-2.3%	-	-	-	-
P3O	-	-	-	-	-	-	-	-	-	-	-	-
PCPO	-	-	-	-	-	-	-	-	+14.2%	+0.5%	+8.9%	-1.1%
FOCOPS	-	-	-	-	+14.7%	-1.2%	+14.2%	+1.1%	-	-	-	-
CPPO-PID	-	-	-	-	-	-	-	-	-	-	-	-
IPO	-	-	-	-	-	-	-	-	-	-	-	-

FOCOPS. [Zhang *et al.*, 2020] finds the current optimal policy by solving a constrained optimization problem in the non-parameterized policy space. Then it projects the non-parameterized policy back into the parametric policy space.

TRPO-Lag and PPO-Lag. The Lagrangian approach solves CMDP (1) as follows,

$$(\pi_\star, \lambda_\star) = \arg\min_{\lambda\geq 0}\max_{\pi\in\Pi_{\boldsymbol{\theta}}} \left\{J(\pi) - \lambda(J^c(\pi) - b)\right\}. \tag{8}$$

TRPO-Lag and PPO-Lag share the same key idea as follows,

$$\pi_{k+1} = \arg\max_{\pi\in\Pi_{\boldsymbol{\theta}}} \left\{J(\pi) - \lambda_k(J^c(\pi) - b)\right\}, \tag{9}$$

$$\lambda_{k+1} = \lambda_k + \eta(b - J^c(\pi_k))_+, \tag{10}$$

where η is step-size. In practice, the two algorithms use different ways to approximate $J(\pi)$ and $J^c(\pi)$. PPO-Lag uses the following clip term to replace $J(\pi)$ in (9),

$$\begin{aligned}\mathcal{L}^r_{\text{clip}}(\pi) = \mathbb{E}_{s\sim d^{\rho_0}_{\pi_k}(\cdot), a\sim\pi_k(\cdot|s)}\Big[\\ \min\Big\{\frac{\pi(a|s)}{\pi_k(a|s)}A_{\pi_k}(s,a), \\ \text{clip}\left(\frac{\pi(a|s)}{\pi_k(a|s)}, 1-\epsilon, 1+\epsilon\right)A_{\pi_k}(s,a)\Big\} \\ \Big].\end{aligned} \tag{11}$$

Similarly, we obtain $\mathcal{L}^c_{\text{clip}}$ with $A_{\pi_k}(s,a)$ replacing $A^c_{\pi_k}(s,a)$, and PPO-Lag replaces $J^c(\pi)$ with $\mathcal{L}^c_{\text{clip}}$. TRPO-Lag approximates $J(\pi)$ and $J^c(\pi)$ following TRPO.

P3O. [Zhang *et al.*, 2022] solves the cumbersome constrained policy iteration via a single minimization of an equivalent unconstrained problem:

$$\begin{aligned}\pi_{k+1} = \arg\min_{\pi\in\Pi_{\boldsymbol{\theta}}} \Big\{ \\ \mathbb{E}_{s\sim d^{\rho_0}_{\pi_k}(\cdot), a\sim\pi_k(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_k(a|s)}A_{\pi_k}(s,a)\right] + \kappa B(\pi,b) \\ \Big\},\end{aligned} \tag{12}$$

where κ is a positive scalar, and the penalty term $B(\pi,b)$ is defined as follows,

$$\begin{aligned}B(\pi,b) = \max\Big\{ \\ 0, \mathbb{E}_{s\sim d^{\rho_0}_{\pi_k}(\cdot), a\sim\pi_k(\cdot|s)}\left[\frac{\pi(a|s)}{\pi_k(a|s)}A^c_{\pi_k}(s,a)\right] \\ + (1-\gamma)\left(J^c(\pi_k) - b\right) \\ \Big\}.\end{aligned} \tag{13}$$

P3O utilizes a simple yet effective penalty approach to eliminate cost constraints and removes the trust-region constraint by the clipped surrogate objective.

IPO. [Liu *et al.*, 2020] considers the objective with logarithmic barrier functions [Nemirovski, 2004] to learn the safe policy. Concretely, IPO considers the following way to update policy,

$$\pi_{k+1} = \arg\max_{\pi\in\Pi_{\boldsymbol{\theta}}} \left\{\mathcal{L}^r_{\text{clip}}(\pi) + \phi(\pi)\right\}, \tag{14}$$

where the clip objective $\mathcal{L}^r_{\text{clip}}(\pi)$ is defined in (11), and $\phi(\pi)$ is the logarithm barrier function with respect to the CMDP problem,

$$\phi(\pi) = \frac{1}{m}\log\Big(b - J^c(\pi)\Big), \tag{15}$$

where $m > 0$ is a hyper-parameter that needs to be tuned.

CPPO-PID. [Stooke *et al.*, 2020] is also a primal-dual policy optimization method (8). Instead of using the stochastic gradient method to update the Lagrange multiplier, CPPO-PID considers the PID control technique [Johnson and Moradi, 2005] to update the Lagrange multiplier λ.

6 SafePO Implementation

In this section, we provide a detailed discussion of the design of SafePO. Features such as strong performance, fast training, extensibility, customization, visualization, and documentation are all presented to demonstrate the advantages and contributions of SafePO.

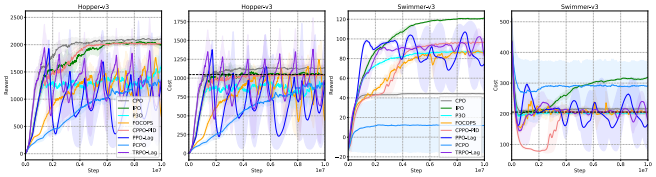

Figure 3: Learning curves for Hopper-v3 and Swimmer-v3 on MuJoCo-Velocity tasks.

Overall Architecture Design. The overall architecture design of `SafePO` is shown in figure 1. We abstract a similar structure of safe policy optimization algorithms and modularize the code into interaction with environments, parallel sample collection, buffer storage, and computation, algorithm core update, and auxiliary functionalities such as visualization and logger. Maximum abstraction and encapsulation take place at the implementation of the core of the algorithm, where each algorithm inherits directly from its base algorithm; thus, unique features have to be implemented, and all other code can be reused. For example, the classes of policy gradient, natural policy gradient, TRPO, and TRPO-Lag form an inheritance chain. In this way, not only is code reused and consistency ensured, but it also effectively unifies algorithms into a single common framework in the most coherent way. Such a tree-like structure is our novelty and contribution to benchmark implementation, as it provides practitioners with a new way to organize related algorithms in a single framework. Moreover, the overall design also leads to its extensibility, which will be discussed later.

Correctness. For a benchmark, it is critical to ensure its correctness and reliability. To achieve this goal, we examine the implementation of `SafePO` carefully. First, every algorithm is implemented strictly according to the original paper. Secondly, for algorithms with a commonly acknowledged open-source code base, we compare our implementation with those line by line to double-check the correctness. For the others, we also try to substantiate their correctness further. As an example, P3O does not have open-source code. We contacted the authors to invite them to inspect our implementation and are responded with approval.

Experimentally, `SafePO` achieves comparable and, most of the time, better performance than the original papers. As shown in the first column in table 1, `SafePO` is consistently stronger than the safety-starter-agent. Moreover, while some algorithm implementations are reported to fail on some tasks (e.g., FOCOPS fails on safety-gym and is admitted officially by the author in a GitHub issue[6]), `SafePO` successfully attains desirable performances which can be referred to Appendix B.2. Therefore, we are confident about the correctness of `SafePO` while also open to suggestions for improvement.

Efficiency. `SafePO` supports multi-core CPU parallelization, substantially accelerating sample collection and training. This enables researchers to conduct more experiments in a limited time. Moreover, `SafePO` supports the newest version of PyTorch and the latest hardware architecture, making it convenient for contemporary practitioners.

[6]https://github.com/ymzhang01/focops/issues/1

Extensibility. `SafePO` enjoys high extensibility thanks to its architecture. New algorithms can be integrated to `SafePO` by inheriting from base algorithms and only implementing their unique features. For example, we integrate PPO by inheriting from policy gradient and only adding the clip ratio variable and rewriting the function that computes the loss of policy π. Similarly, algorithms can be easily added to `SafePO`. Beyond, other parts of `SafePO` are also changeable by simply replacing the existing implementation. It enables users to validate their ideas quickly and allows fair comparisons since all other parts are controlled.

Customization. `SafePO` provides flexible customization through configuration files in YAML format and command line arguments. On average, more than 30 hyperparameters can be specified by users.

Logging and Visualization. Another important functionality of `SafePO` is logging and visualization. Supporting both TensorBoard and WandB, we offer code for visualizing more than 40 parameters and intermediate computation results to inspect the training process. Common parameters and metrics such as KL-divergence, learning rate, SPS (step per second), and max, min, mean, and variance of cost and reward are visualized universally. Special features of algorithms are also reported, such as the Lagrangian multiplier of Lagrangian-based methods, $q = g^T H^{-1} g, r = g^T H^{-1} b, s = b^T H^{-1} b, \nu^*$, and λ^* of CPO, proportional, integral, and derivative of PID-based algorithms, etc. During training, users can inspect the changes of every parameter, collect the log file, and obtain saved checkpoint models. The complete and comprehensive visualization allows easier observation, model selection, and comparison.

Documentation. In addition to its code implementation, `SafePO` comes with extensive documentation in the code base. We include detailed guidance on installation and propose solutions to common issues. Moreover, we provide instructions on simple usage and advanced customization of `SafePO`. Official information concerning maintenance, ethical and responsible use are stated clearly for reference. Overall, the documentation enables a quick start to use `SafePO`.

7 Experiments and Results

To test the performance of our implementation, we run the eight algorithms on 30 tasks (for a complete list of the tasks, please refer to Appendix B.1) contained in the four environment suites and present our experimental results for the reference of the community. Based on these results, we derive key insights, which are analyzed below.

7.1 Experimental Observations

In Appendix B.2, we show the performance of our implementations on various tasks. In the first place, we have to state that we report the *undiscounted* total reward and cost in the figures and tables, which differs from [Zhang *et al.*, 2020][7], but is consistent with OpenAI [Ray *et al.*, 2019]. From comprehensive experiments, we find that CPO guarantees safety

[7]https://github.com/ymzhang01/focops

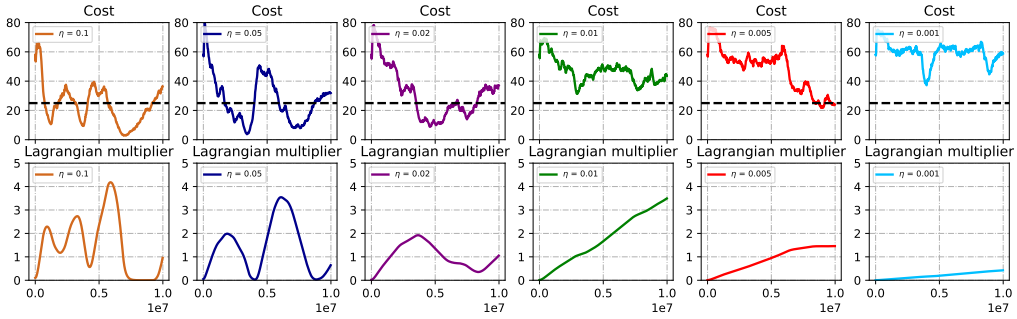

Figure 4: Curves of cost and Lagrangian multiplier on PointGoal1 with TRPO-Lag.

constraints, but does not attain high rewards. IPO is based on the interior-point method and thus needs to be initialized in the feasible set or optimized through gradient clipping, posing a great challenge to parameter tuning. Empirically, it works well in BallReach, AntRun, AntCircle, etc., while exhibiting instability in other environments. FOCOPS and PCPO share similar theoretical properties, but FOCOPS attains more robust performance. From the data of Goal-series tasks in Safety-Gym, IPO, CPO, and PCPO fail to satisfy cost constraints, while PPO-Lag and TRPO-Lag meet safety requirements and achieve high rewards. Based on PID methods, CPPO-PID has low-cost oscillation. As it introduces three parameters, P (Proportional, for response speed), I (Integral, for error minimization), and D (Derivative, for oscillation suppression), CPPO-PID maintains stable cost performance during training. However, we also find that CPPO-PID tends to diverge in some environments, suggesting that we may introduce other effective methods from the control system as a means to bridge reinforcement learning and control theory.

7.2 Comments on MuJoCo-Velocity Tasks

An important observation is that the safety task of MuJoCO-Velocity is uncommon, where a contradictory phenomenon occurs between reward improvement and constrained limited cost for the robots like Hopper-v3 and Swimmer-v3.

Concretely, for example, the reward of Hopper-v3 is mainly determined by the following term [8]

$$r_t(s,a) \propto \frac{\|(x_{t'},y_{t'})-(x_t,y_t)\|_2}{t'-t} \approx \sqrt{v_x^2+v_y^2} = c_t(s,a), \tag{16}$$

where (x_t, y_t) is the two-dimensional coordinate of the robot at time t, and $r_t(s,a)$ denotes the reward with respect to the robot transforms from (x_t, y_t) to $(x_{t'}, y_{t'})$ after taking action a. The relationship (16) shows that the reward function is proportional to the cost function. Recall that the goal of safe reinforcement learning is to find a policy that both attains a high total reward and controls the cumulative cost, which is very difficult to be reconciled between the reward improvement and the bounded cost limit.

[8] We present the details with respect to reward according to the open implementation: https://www.gymlibrary.ml/environments/mujoco/hopper/.

Additionally, from Eq.(16), we also know that the learning curve of reward performance and cost performance can be very similar potentially. The result of Figure 3 is consistent with our analysis, where the reward and cost learning curve shares a similar shape. Besides, from Figure 3, we know the reward increases while the cost also increases, which is undesirable for safety learning.

7.3 Comments on Lagrangian Multiplier

We observe that the Lagrangian multiplier is critical in learning a safe policy. If the policy violates the cost limit, the Lagrangian multiplier increases and guides the policy to update in a safe direction. Otherwise, the Lagrangian multiplier gradually decreases, which allows the agent to improve within the safety region.

Recall (8), the Lagrangian methods (e.g., PPO-Lag or TRPO-Lag) consider the following update rule of λ:

$$\lambda_{k+1} = \{\lambda_k + \eta(\hat{J}_k^c - b)\}_+, \tag{17}$$

where $\hat{J}_k^c$ is an estimator for the cost function, $\eta > 0$ is step-size. Those empirical result shown in Figure 4 is consistent with the update rule of λ in Eq.(17). If the estimated cost function is under the target threshold b, then λ gradually decreases. Otherwise, λ increases, which forces the agent to take safe actions. This implies that the Lagrangian multiplier plays an important role for agent to learn a safe policy. Additionally, from the above discussion, we see that figure 4 effectively visualizes the difficulty of different tasks, which can be measured by the scale of the Lagrangian multiplier.

8 Conclusion

In this study, we standardize the safe policy optimization methods for solving constrained Markov decision processes and introduce a unified, highly-optimized, extensible, and comprehensive algorithm benchmark named `SafePO`. We support three major safety environment suites, propose a new series of environments (Safety DexterousHands), and provide PyTorch-version re-implementations of eight algorithms, which effectively resolve the undesirable research status of safe reinforcement learning. Exhaustive experiments have been conducted, and the results substantiate the correctness and strong performance of `SafePO`. As all benchmark algorithms are model-free on-policy methods, for future work, we will cover model-based, off-policy, and control-based methods to expand the scope of `SafePO`.

References

[Achiam *et al.*, 2017] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[Al-Rfou *et al.*, 2016] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv e-prints*, pages arXiv–1605, 2016.

[Alshiekh *et al.*, 2018] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Altman, 1999] Eitan Altman. *Constrained Markov decision processes*. CRC Press, 1999.

[Beutler and Ross, 1985] Frederick J Beutler and Keith W Ross. Optimal policies for controlled markov chains with a constraint. *Journal of mathematical analysis and applications*, 112(1):236–252, 1985.

[Bi *et al.*, 2021] Zhu Ming Bi, Chaomin Luo, Zhonghua Miao, Bing Zhang, WJ Zhang, and Lihui Wang. Safety assurance mechanisms of collaborative robotic systems in manufacturing. *Robotics and Computer-Integrated Manufacturing*, 67:102022, 2021.

[Carlson and Demiris, 2010] Tom Carlson and Yiannis Demiris. Increasing robotic wheelchair safety with collaborative control: Evidence from secondary task experiments. In *2010 IEEE International Conference on Robotics and Automation*, pages 5582–5587. IEEE, 2010.

[Chen *et al.*, 2022] Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuang Jiang, Stephen Marcus McAleer, Hao Dong, Zongqing Lu, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning. *arXiv preprint arXiv:2206.08686*, 2022.

[Chow *et al.*, 2018] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[Chow *et al.*, 2019] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

[de Witt *et al.*, 2020] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[Dulac-Arnold *et al.*, 2019] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[Gronauer, 2022] Sven Gronauer. Bullet-safety-gym: Aframework for constrained reinforcement learning. 2022.

[Gu *et al.*, 2021] Shangding Gu, Jakub Grudzien Kuba, Munning Wen, Ruiqing Chen, Ziyan Wang, Zheng Tian, Jun Wang, Alois Knoll, and Yaodong Yang. Multi-agent constrained policy optimisation. *arXiv preprint arXiv:2110.02793*, 2021.

[Johnson and Moradi, 2005] Michael A Johnson and Mohammad H Moradi. *PID control*. Springer, 2005.

[Kallenberg, 1983] Lodewijk CM Kallenberg. Linear programming and finite markovian control problems. *MC Tracts*, 1983.

[Kuba *et al.*, 2021] Jakub Grudzien Kuba, Ruiqing Chen, Munning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[Leike *et al.*, 2017] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017.

[Liu *et al.*, 2020] Yongshuai Liu, Jiaxin Ding, and Xin Liu. Ipo: Interior-point policy optimization under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4940–4947, 2020.

[Liu *et al.*, 2022] Zuxin Liu, Zijian Guo, Zhepeng Cen, Huan Zhang, Jie Tan, Bo Li, and Ding Zhao. On the robustness of safe reinforcement learning under observational perturbations. *arXiv preprint arXiv:2205.14691*, 2022.

[Makoviychuk *et al.*, 2021] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[Nemirovski, 2004] Arkadi Nemirovski. Interior point polynomial time methods in convex programming. *Lecture notes*, 42(16):3215–3224, 2004.

[Ray *et al.*, 2019] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

[Ross and Varadarajan, 1989] Keith W Ross and Ravi Varadarajan. Markov decision processes with sample path constraints: the communicating case. *Operations Research*, 37(5):780–790, 1989.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust

region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[Stooke *et al.*, 2020] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[Yang *et al.*, 2020] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-based constrained policy optimization. *arXiv preprint arXiv:2010.03152*, 2020.

[Yang *et al.*, 2022] Long Yang, Jiaming Ji, Juntao Dai, Yu Zhang, Pengfei Li, and Gang Pan. Cup: A conservative update policy algorithm for safe reinforcement learning. *arXiv preprint arXiv:2202.07565*, 2022.

[Yu *et al.*, 2021] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[Yuan *et al.*, 2021] Zhaocong Yuan, Adam W Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P Schoellig. safe-control-gym: a unified benchmark suite for safe learning-based control and reinforcement learning. *arXiv preprint arXiv:2109.06325*, 2021.

[Zhang *et al.*, 2020] Yiming Zhang, Quan Vuong, and Keith Ross. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 33:15338–15349, 2020.

[Zhang *et al.*, 2022] Linrui Zhang, Li Shen, Long Yang, Shixiang Chen, Bo Yuan, Xueqian Wang, Dacheng Tao, et al. Penalized proximal policy optimization for safe reinforcement learning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.