CRAX: FAST SAFE REINFORCEMENT LEARNING BENCHMARKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Safety is a core concern for deploying reinforcement learning (RL) agents in real-world domains such as robotics and autonomous driving. While benchmarks have been central to progress in RL, existing 3D physics-based safety benchmarks remain computationally slow, limiting large-scale experimentation and rapid prototyping. To address this gap, we propose **CRAX** (**C**onstrained **R**L **A**ccelerated with JA**X**). Built on top of the MuJoCo XLA (MJX) physics engine, CRAX leverages vectorized operations and hardware acceleration, yielding up to 200x faster training over comparable CPU-based safety benchmarks. The benchmark features eight tasks spanning three difficulty levels and multiple agent morphologies. Evaluating seven popular safe RL methods, we find that none dominates across tasks, and that learning safe policies from pixels remains largely unsolved.

1 INTRODUCTION

Although the progress in reinforcement learning (RL; Sutton & Barto, 2018) has been sped up by the use of hardware acceleration, this progress has not yet translated to the research in safe RL (SafeRL; García & Fernández, 2015). Benchmarks are a catalyst driving innovation in research; for instance, ImageNet (Deng et al., 2009) motivated the introduction of CNNs (Krizhevsky et al., 2012), and the Arcade Learning Environment (Bellemare et al., 2013) supported the development of deep Q-networks (Mnih et al., 2015). In a similar trend, accelerated hardware brought a new wave of benchmarks that facilitate research in multiple areas of RL, including goal-conditioned (Bortkiewicz et al., 2025), multi-agent (Rutherford et al., 2024), offline (Jackson et al., 2025), and open-ended (Matthews et al., 2024) RL. Within the SafeRL literature, Safety Gym (Ray et al., 2019) and, more recently, Safety-Gymnasium (Ji et al., 2023) have established a set of common tasks that facilitate comparisons between SafeRL algorithms. Nevertheless, SafeRL research still relies mostly on CPU-based simulations and does not leverage GPU computation.

Effective research in RL, particularly in high-dimensional problems, requires fast data collection, as training RL typically requires a large number of environment interactions. This demand can compromise the research development phase, where we retrain such agents numerous times from scratch, while performing: hyperparameter tuning to ensure a fair comparison, tests in multiple environments to evaluate generalization, repeated runs for statistical significance, and ablation studies to assess individual components of an algorithm. Together, these requirements form a bottleneck for the development of new algorithms. Fast simulation is therefore essential to support research on RL.

By leveraging high-fidelity simulators with hardware acceleration, RL is increasingly closing the gap to real-world applications such as robotics. Simulation platforms such as BRAX (Freeman et al., 2021) and Isaac Lab (Mittal et al., 2025) run on accelerated hardware, allowing researchers to leverage substantial speedups from parallel computing architectures. Such platforms enable large-scale policy training in simulation and, in some cases, transfer to physical robots (Zakka et al., 2025). Nevertheless, this approach still relies closely on reward engineering to specify the behavior desired from the agent. However, in many situations, expressing such behaviors is easier through constraints (Roy et al., 2022), particularly in safety-critical scenarios (Ray et al., 2019). Therefore, we focus on fast RL benchmarks with explicit constraints.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We introduce **CRAX**[1] (**C**onstrained **R**L **A**ccelerated with JA**X**), a novel hardware-accelerated SafeRL benchmark leveraging MuJoCo, a general-purpose 3D physics engine. The design principles of CRAX are inspired by BRAX (Freeman et al., 2021) and Safety Gymnasium (Ji et al., 2023). CRAX provides a set of simulated tasks, robots, and algorithm baselines for evaluating SafeRL leveraging parallel computing, resulting in higher simulation speeds than CPU-based setups, enabling more rigorous testing and faster algorithm development for the SafeRL community. Each task defines reward and cost signals, inducing a trade-off between performance and safety: achieving high reward typically incurs more cost, so satisfying safety constraints necessitates sacrificing some reward. Among numerous constraint formulations (Wachi et al., 2024), we focus on bounding the expected cumulative discounted cost, the most widely adopted approach in the SafeRL literature. Nonetheless, CRAX can easily be extended to handle other formulations.

CRAX offers a diverse set of environments and algorithms. It includes eight tasks, eight agent morphologies, three difficulty levels per task, and three observation types (state vector, egocentric view, and allocentric view). Furthermore, it reimplements seven SafeRL algorithms in JAX, covering primal-dual, state-augmented, and trust-region approaches, with support for off-policy methods. These include baseline algorithms for constrained RL, such as PPO Lagrangian (PPO-Lag; Ray et al., 2019), as well as unconstrained PPO (Schulman et al., 2017) as a reference. These implementations facilitate comparisons between new algorithms and relevant prior work.

The speed-ups of the hardware-accelerated implementation of CRAX enable:

1. **Reliable assessment.** CRAX makes extensive evaluations affordable, such as wide hyperparameter sweeps and more repetitions per experiment, which increases confidence in the results.
2. **Fast prototyping.** CRAX trains quickly, enabling rapid experimentation and speeding up the research process.

CRAX reaches two orders of magnitude higher throughput than Safety-Gymnasium (Section 5.3). This throughput enables the evaluation of seven SafeRL baselines across all tasks and difficulty levels, which shows that no method strictly dominates the others (Section 5). We complement final performance with sample efficiency and constraint violations during training (Appendix C), quantify how many seeds reliable comparisons require (Appendix B), and sweep the hyperparameters of each method (Section H.1). Beyond the baselines, we study learning from pixel observations (Section 5.2), the performance-safety trade-off under varying cost thresholds (Appendix E), safe curriculum learning and safety transfer (Appendix D), and off-policy methods (Appendix F).

2 Related Work

Safe Reinforcement Learning. In SafeRL, besides achieving high-performance, agents also need to adhere to established safety requirements during learning and deployment (García & Fernández, 2015). While safety can be encouraged indirectly through reward shaping, this approach places the burden of balancing performance and safety on the system designer. To reduce this burden, we can model safety requirements explicitly as constraints (Roy et al., 2022; Kamran et al., 2022), allowing the agent itself to autonomously find a trade-off between reward maximization and constraint satisfaction. A wide range of safety formulations have been studied, including chance constraints, almost-sure constraints, and per-step constraints (Wachi et al., 2024). In practical work, the most common experimental settings bound the expected sum of discounted costs over time (Yuan et al., 2022; Ji et al., 2023; Ramanujam et al., 2025). While CRAX is agnostic to the specific safety formulation, our empirical evaluation adopts this setting due to its popularity.

SafeRL Benchmarks. Safe RL was predominantly studied in low-dimensional 2D settings, such as gridworlds in AI Safety Gridworlds (Leike et al., 2017) and MiniGrid (Chevalier-Boisvert et al., 2023) tasks adapted for safety. More recent benchmarks have shifted toward environments for embodied, pixel-based learning (Dosovitskiy et al., 2017; Li et al., 2022; Tomilin et al., 2025) and physics-based continuous control (Yuan et al., 2022; Ji et al., 2023). These benchmarks are typically built on existing simulators rather than developing physics engines from scratch. For example, HASARD (Tomilin et al., 2025) is built on ViZDoom, while Safety-Gymnasium (Ji et al., 2023) extends MuJoCo tasks, adding safety constraints. CRAX follows this design principle by building on MJX (MuJoCo XLA Authors, 2023), the JAX-based accelerated backend of MuJoCo (Todorov et al., 2012).

[1]The code and environments are accessible on GitHub.

Table 1: Key characteristics of popular Reinforcement Learning benchmarks. CRAX uniquely combines a focus on safety with hardware acceleration and 3D physics-based tasks.

Benchmark	Safety	GPU	3D Phys.	Type	Reference
OpenAI Gym	×	×	×	Classic control	(Brockman et al., 2016)
Procgen Benchmark	×	×	×	Procedural generation	(Mohanty et al., 2020)
Atari (ALE)	×	×	×	Arcade games	(Bellemare et al., 2015)
Gymnax	×	√	×	Classic control / MinAtar	(Lange, 2022)
JaxMARL	×	√	×	Multi-agent	(Rutherford et al., 2024)
Craftax	×	√	×	Open-ended / gridworld	(Matthews et al., 2024)
Jumanji	×	√	×	Combinatorial optimization	(Bonnet et al., 2024)
XLand-MiniGrid	×	√	×	Meta-RL	(Nikulin et al., 2024)
VMAS	×	√	×	Multi Agent 2D Physics	(Bettini et al., 2022)
DeepMind Control Suite	×	×	√	3D physics	(Tassa et al., 2018)
Meta-World	×	×	√	Robotic manipulation	(Yu et al., 2019)
Brax	×	√	√	3D physics	(Freeman et al., 2021)
Isaac Lab	×	√	√	3D physics	(Mittal et al., 2025)
AI Safety Gridworlds	√	×	×	2D gridworlds	(Leike et al., 2017)
HASARD	√	×	×	FPS game	(Tomilin et al., 2025)
SafeOR-Gym	√	√	×	Operations research	(Ramanujam et al., 2025)
Bullet-Safety-Gym	√	×	√	Safe navigation	(Gronauer, 2022)
Safe-Control-Gym	√	×	√	Safe control	(Yuan et al., 2022)
Safety Gym (OpenAI)	√	×	√	Safe navigation	(Ray et al., 2019)
Safety-Gymnasium	√	▲	√	Safe navigation / locomotion	(Ji et al., 2023)
CRAX (Ours)	√	√	√	Safe navigation / locomotion	

▲ A subset of two robotic manipulation suites are GPU-accelerated through "Safe Isaac Gym", which is part of Safety Gymnasium.

Accelerated Benchmarking. RL experiments are typically data-intensive, as meaningful evaluation requires repeated environment interactions for hyperparameter tuning, statistical significance analysis, and testing across different tasks. While GPUs are routinely used to accelerate neural network training, online RL remains constrained when environment rollouts are executed on the CPU, making simulation throughput the main bottleneck. This has motivated moving both simulation and learning onto parallel hardware to accelerate the full training loop. Brax (Freeman et al., 2021) provides hardware-accelerated continuous-control environments in JAX. VMAS (Bettini et al., 2022) and JaxMARL (Rutherford et al., 2024) focus on scalable multi-agent RL. The former creates 2D physics environments implemented in PyTorch, and the latter creates JAX-native variants of many popular multi-agent environments. Craftax (Matthews et al., 2024) explores procedurally generated grid-based worlds optimized for large-scale parallel training. Table 1 summarizes a number of such widely used RL benchmarks. CRAX expands the available selection of GPU-accelerated, safety-focused, and 3D physics-based environments.

3 Constrained Reinforcement Learning

A constrained Markov decision process (CMDP; Altman, 1999) is an MDP (Puterman, 1994) with constraints, characterized by a tuple $\mathcal{M} = \langle S, A, P, r, c, d, \gamma \rangle$, where S is a continuous state space, A a continuous action space, P a transition function $P\colon S \times A \to \text{Distr}(S)$, r a reward function $r\colon S \times A \to \mathbb{R}^+$, c a cost function $c\colon S \times A \to \mathbb{R}^+$, $d \in \mathbb{R}^+$ a cost threshold, and $\gamma \in [0, 1)$ a discount factor.

An RL agent interacting with a CMDP follows a stochastic policy $\pi\colon S \to \text{Distr}(A)$. The value function $V^\pi(s)$ represents the expected cumulative discounted reward when following policy π starting from state s over a (potentially infinite) horizon T: $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \mid s_0 = s \right]$, where the expectation $\mathbb{E}_\pi$ is taken over the trajectory distribution induced by policy π, with actions $a_t \sim \pi(\cdot|s_t)$ and successor states $s_{t+1} \sim P(\cdot|s_t, a_t)$. Similarly, the cost function $C^\pi(s)$ captures the expected cumulative discounted cost under policy π starting from state s: $C^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t c(s_t, a_t) \mid s_0 = s \right]$.



Safe Goal
Reach the goal without colliding with the hazards

Safe Push
Push the block to the goal while avoiding obstacles

Safe Circle
Move along a circular path while avoiding hazards

Safe Reacher
Reach the target without touching the hazards

Safe Lift
Move forward while lifting designated legs

Safe Pathway
Traverse the path without stepping on the hazards

Safe Velocity
Move fast while adhering to a velocity limit

Safe Height
Move forward under the height bound

Figure 1: Overview of the CRAX benchmark environments.

Table 2: Overview of the available suites in the benchmark (rows), and which agents are compatible with them (columns). Starred entries ($\star$) have been selected for evaluation in Section 5.

	3D				2D			Fixed base
Task	**Point**	**Ant**	**Humanoid**	**Spider**	**HalfCheetah**	**Walker2d**	**Hopper**	**Reacher**
Goal	$\surd^\star$	$\surd$	$\surd$	$\surd$	–	–	–	–
Circle	$\surd^\star$	$\surd$	$\surd$	$\surd$	–	–	–	–
Push	$\surd^\star$	$\surd$	$\surd$	$\surd$	–	–	–	–
Velocity	$\surd$	$\surd$	$\surd^\star$	$\surd$	$\surd$	$\surd$	$\surd$	–
Height	–	–	$\surd^\star$	–	–	–	–	–
Lift	–	–	–	$\surd^\star$	–	–	–	–
Pathway	–	–	–	–	$\surd$	$\surd^\star$	$\surd$	–
Reach	–	–	–	–	–	–	–	$\surd^\star$

The objective in the CMDP framework is to find an optimal policy $\pi^* \in \Pi$ that maximizes the expected cumulative reward while ensuring the expected cumulative cost remains below the threshold d for all states $s \in S$. This constrained optimization problem is formulated as: $\max_{\pi \in \Pi} V^\pi(s) \quad \text{subject to} \quad C^\pi(s) \leq d, \quad \forall s \in S$. The constraint $C^\pi(s) \leq d$ enforces safety by requiring that the policy maintains cost levels below the specified threshold regardless of the initial state. This formulation, known as the expected cumulative cost constraint (Wachi et al., 2024), distinguishes CMDPs from standard MDPs, where the agent would simply maximize the reward without regard for cost constraints.

4 CRAX

This section covers the design choices behind CRAX. The benchmark has been inspired by the GPU-accelerated RL environments in BRAX (Freeman et al., 2021) and the safety environments of Safety Gymnasium (Ji et al., 2023). It is intended as a research and benchmarking platform for SafeRL, providing ready-to-use environment suites, agent morphologies, and utility tooling to design SafeRL experiments, alongside a set of pre-configured tasks of increasing difficulty. Every task exposes a cost signal in addition to a reward, and the entire training loop runs on the GPU through JAX. Two further principles shaped the design. First, the benchmark should be neither immediately solvable by current methods nor so hard that no progress is possible, which is why each suite comes in three difficulty levels. Second, it should include the tooling needed to assess the safety properties of the algorithms under evaluation, not only their returns.

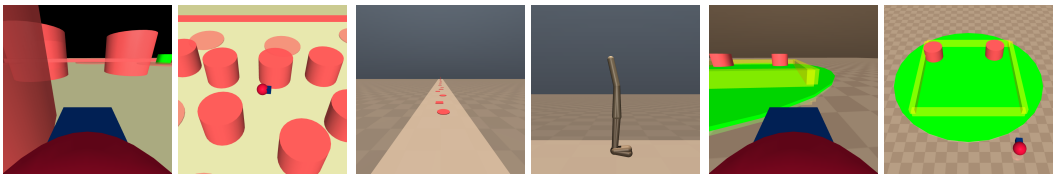


Figure 2: **Egocentric** (left of each pair) and **allocentric** (right) views. Policies receive them at a 64×64 resolution. Here we show them at a higher resolution for legibility. The camera tracks the agent in `Point Goal` and `Walker Pathway`, but stays fixed in `Point Circle`.

4.1 Environment Suites and Tasks

Environment *suites* define families of configurable tasks in simulated 3D environments. In each suite, an agent seeks to maximize reward while adhering to a predefined cost threshold. Figure 1 summarizes the objectives associated with each suite. Beyond suite-specific parameters, such as the number of obstacles in *Point Goal*, every suite specifies its own reward and cost signals. An instantiation of a suite's parameters constitutes a *task*. Appendix A provides a detailed description of all suites. The Navigation and Velocity suites descend from Safety-Gymnasium (Ji et al., 2023), while Pathway, Height, Lift and Reach are new. Appendix G brings an example of how closely an inherited suite tracks its original and compares training curves on the closest matching task.

4.2 Agents

Agents are the acting bodies in the environment. Each agent has a unique morphology and lidar sensors to detect its surroundings. The movement of some agents is constricted in one or more dimensions. Some suites are compatible with multiple agent types, while some of the suites allow for only a single agent type. Table 2 provides an overview of the agent types and their compatibility with the suites. Section A.3 provides further details about the agents.

4.3 Rewards, Costs and Constraint Types

Each environment comes with a distinct default reward signal. Cost signals are constructed using a variety of constraints. Appendix A provides an overview of the reward and cost signals of each environment. The benchmark employs five key constraint formulations. (1) **Hazard constraints**: binary cost on collision with solid obstacles or a cost growing with penetration into hazard regions. (2) **Velocity constraints**: binary cost whenever the agent exceeds a speed limit. (3) **Height constraints**: hinge cost that grows as the agent rises above a threshold. (4) **Contact constraints**: binary cost whenever a restricted limb touches the ground. (5) **Boundary constraints**: binary cost whenever the agent leaves the permitted region. As established in Section 3, staying within the safety budget does not preclude further gains: higher returns can be achieved with more refined strategies while remaining safe. Moreover, the safety bound is adjustable, allowing one to impose stricter or more permissive requirements, and thereby modulate the task difficulty (Appendix E).

4.4 Observation Modalities

By default, agents observe a vector of proprioceptive state together with lidar and compass readings (Section A.3), and all results use this modality unless stated otherwise. CRAX also facilitates pixel-based observations. The images are rendered on the GPU within the same compiled loop as the simulation, so pixel observations never leave the device, and any camera defined in a task can serve as an observation. Every agent and task provides two views (Figure 2). An egocentric camera is mounted on the agent, which is closer to how real robots perceive their surroundings. An allocentric camera observes the agent from outside, and its placement depends on the task. It follows the agent from above at an angle in `Point Goal`, stays fixed over the smaller arena of `Point Circle`, and follows the agent from the side in `Walker Pathway`. Each actor and critic head declares which observations it consumes, so the modalities can be routed to different heads independently.

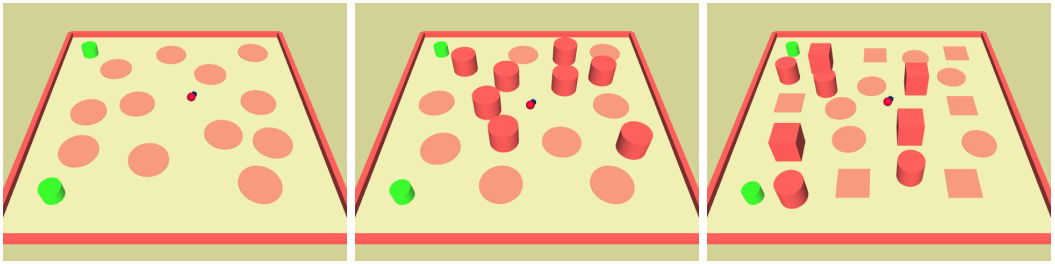


Figure 3: Higher **difficulty levels** of `Point Goal` (1 to 3, from left to right) decrease the size of the goals and increase the number and variety of hazards.

4.5 Difficulty Levels

A useful benchmark ought to serve two complementary purposes. First, it should have a lenient evaluation setting that allows for comparing and analyzing existing methods. Second, it should pose a significant challenge to remain relevant for more advanced future methods. To this end, we create each CRAX suite in three difficulty levels. The lowest level tasks are designed such that most existing methods are capable of learning a reasonable policy and achieving meaningful performance, while the highest levels leave substantial room for improvement. The difficulty increase between levels depends on the nature of the environment. For example, in `Point Goal`, higher levels introduce a greater number and variety of hazards that the agent must avoid (Figure 3). In `Spider Lift`, each successive level requires the *Spider* agent to keep one additional leg off the ground to avoid costs. Appendix A provides the exact parameter settings defining each difficulty level, and Figure 10 visually depicts the difficulty levels of the tasks.

5 Empirical Evaluation

To assess CRAX, we evaluate several popular SafeRL baselines and one unconstrained RL baseline. (1) We include **PPO** (Schulman et al., 2017) to serve as an unconstrained reference point, ignoring costs entirely. (2) **PPOCost** (Tomilin et al., 2025) extends PPO by treating costs as negative rewards. (3) **PPOLag** (Ray et al., 2019), a primal-dual approach that updates both the policy and a learned Lagrange multiplier to balance return and safety. (4) **PPOPID** (Stooke et al., 2020) refines **PPOLag**'s strategy by updating the Lagrange multiplier with a proportional-integral-derivative controller, allowing the reward-safety trade-off to adjust more responsively during training. (5) **PPOSauté** (Sootla et al., 2022) augments the observed state with a safety budget, treating the constraint as part of the dynamics. (6) **P3O** (Zhang et al., 2022) progressively increases a cost penalty coefficient when constraints are violated, encouraging the policy to adapt toward feasibility without explicit dual updates. (7) **FOCOPS** (Zhang et al., 2020) enforces safety by constraining policy updates through a trust-region formulation, optimizing reward while explicitly bounding expected cost. (8) Finally, **CRPO** (Xu et al., 2021) alternates between objectives without a multiplier, taking a reward-improving step when the estimated cost is within the threshold and a cost-reducing step otherwise.

Experimental Setup. We run all experiments for 500 million environment steps. To choose the number of seeds, we measured how the confidence intervals narrow as seeds are added (Appendix B). They narrow quickly up to five seeds and only slowly beyond, so we use ten seeds unless stated otherwise. We report the undiscounted episodic reward and cost of the final policy, averaged over 128 evaluation episodes, as empirical estimates of V^π and C^π from Section 3, and call a policy safe when its episodic cost stays below the threshold $d = 25$. All experiments are conducted on a dedicated compute node with a 72-core 3.2 GHz AMD EPYC 7F72 CPU and a single NVIDIA H100 GPU. Appendix H provides the exact hyperparameters with the respective sweeps.

5.1 Baseline Algorithm Analysis

Figure 4 shows the performance of baseline algorithms. PPO focuses entirely on maximizing rewards, providing a rough sense of the return achievable when safety is ignored. PPOCost simply subtracts

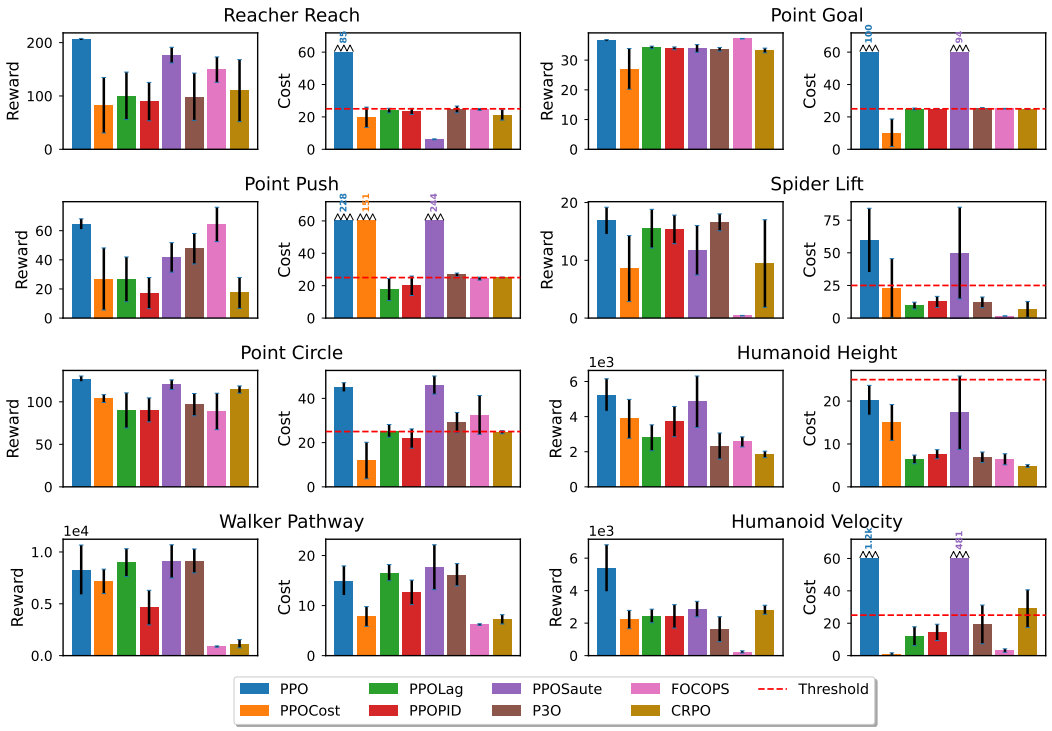


Figure 4: Rewards and costs of baseline methods on Level 1 tasks after 500M environment steps used for training. Error bars denote 95% confidence intervals across 10 seeds.

Table 3: Algorithm summary results across 8 CRAX environments. **Wins**: number of environments where the algorithm achieved the highest reward while being safe (cost < 25). **Safe%**: percentage of environments where the algorithm was safe. **Total**: sum of wins and average safety percentage across levels. Green indicates 100% safe.

	Level 1		Level 2		Level 3		Total	
Algorithm	Wins ↑	Safe% ↑	Wins ↑	Safe% ↑	Wins ↑	Safe% ↑	Wins ↑	Safe% ↑
PPO	1	25	1	12	0	0	2	12
PPOCost	0	88	0	38	1	25	1	50
PPOLag	**2**	88	0	100	0	100	2	96
PPOPID	0	100	0	50	1	88	1	79
PPOSaute	1	38	0	25	0	12	1	25
P3O	**2**	62	**3**	75	**3**	75	**8**	71
FOCOPS	1	75	2	88	**3**	88	6	83
CRPO	1	75	2	62	0	38	3	58

costs from the reward, settling into a compromise, but offers no guarantee of adhering to the constraint. Some methods show distinct affinities for constraint types. PPOSauté satisfies the cost bound in `Reacher` and `Pathway` but otherwise behaves close to unconstrained PPO. FOCOPS performs best on navigation tasks (`Goal`, `Push`) and `Reacher`, but worst on forward-locomotion tasks (`Spider`, `Height`, `Pathway`). All baselines evaluated thus far are on-policy and most build on PPO, which keeps the comparison controlled across constraint-handling mechanisms and suits a fully compiled training loop. CRAX nonetheless supports off-policy learning as well, and Appendix F compares SAC and SACLag with their on-policy counterparts, which they match in performance at about a third of the throughput.

Table 3 provides a summary of the evaluations. P3O and FOCOPS are the strongest baselines on CRAX. PPOLag achieves the highest safety percentage, being the only baseline to satisfy all cost bounds on Levels 2 and 3. However, it fails to reach high rewards. PPOPID is less conservative,

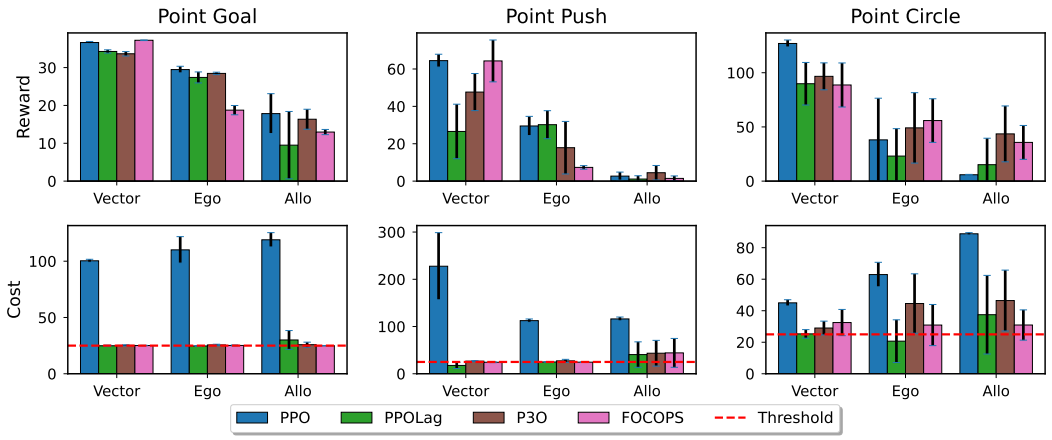


Figure 5: Final reward and cost on Level 1 when learning from **vector**, **egocentric (Ego)**, and **allocentric (Allo)** observations (See Figure 2). The dashed line marks the cost threshold. Error bars denote 95% confidence intervals across 10 seeds.

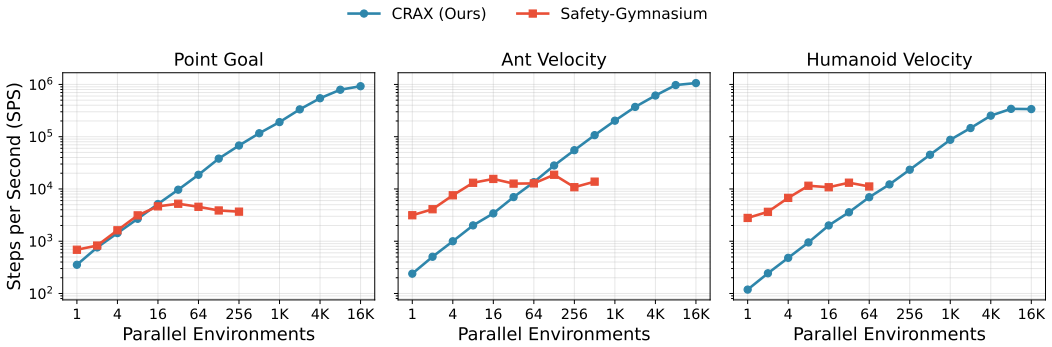


Figure 6: **Simulation throughput** of CRAX and Safety-Gymnasium against the number of parallel environments on identical hardware. Safety-Gymnasium curves end where further scaling failed.

trading stricter safety adherence for slightly higher performance. Appendix I provides more detailed baseline results and training curves, and Appendix C reports how quickly each method reaches its performance and how much it violates safety along the way.

5.2 Learning Safety from Pixels

We train PPO, PPOLag, P3O, and FOCOPS from vector, egocentric, and allocentric observations. Pixel-based agents receive 64×64 RGB frames, encode them with a convolutional network, and train with random-shift augmentation (Kostrikov et al., 2021). Figure 5 shows the results on Level 1 of `Point Goal`, `Point Push`, and `Point Circle`, as on the remaining five environments no method (including unconstrained PPO) learned a meaningful policy from pixels. Every method earns less reward from egocentric frames than from vectors, and less still from allocentric frames, from which no method learns to push the block in `Point Push`. Pixels also widen the confidence intervals and make the cost bound harder to satisfy, particularly from allocentric frames. Learning from pixels thus worsens both sides of the trade-off, as the constrained methods give up reward without reliably staying within the bound. Safe learning from pixels remains an open challenge in CRAX, particularly on the remaining environments.

5.3 Computational Efficiency: A Case Study

Simulation throughput is a core bottleneck in SafeRL (Section 1). We compare CRAX with Safety-Gymnasium (SG), the most widely used benchmark for continuous-control SafeRL, on three tasks that

exist in both, `Point Goal`, `Ant Velocity`, and `Humanoid Velocity` at Level 1, measuring steps per second (SPS) on identical hardware as the number of parallel environments grows. Figure 6 shows that SG is faster with few environments but saturates at 3K-20K SPS. Scaling beyond 256 environments exhausted the CPU memory on our hardware. CRAX keeps scaling to thousands of environments and peaks at roughly 340K SPS on `Humanoid Velocity` and around **1M SPS** on the other two tasks, which is 25 to 180 times the peak of SG. To put this in concrete terms, our full evaluation suite (algorithms $\times$ environments $\times$ difficulty levels $\times$ seeds) amounts to hundreds of training runs and trillions of environment steps. On a single H100 GPU, CRAX completes this in 2 weeks. The equivalent run on Safety-Gymnasium would take close to a year. This case study concerns throughput alone, and Appendix G compares the two benchmarks on learning behavior.

5.4 Reliable Assessment

Fast simulation matters most when it changes how methods are evaluated. We trained every baseline with 20 seeds on all eight Level 1 environments and measured how the 95% confidence interval of the final score narrows as seeds are added (Figure 7). Relative to the mean, the interval shrinks from about 90% at two seeds to 40% at three and 22% at five, after which gains are small. Few seeds can also mislead, as on `Reacher Reach` and `Point Push` the interval at three seeds is narrower than at 20, because the first runs happened to agree. Even at 20 seeds, the hardest environments retain intervals of about 30%, since the agent occasionally fails to learn a useful policy (Appendix B).

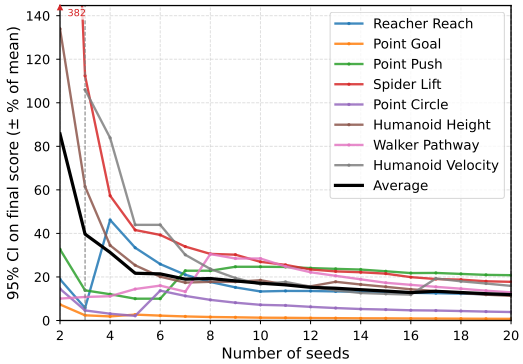


Figure 7: Half-width of the 95% CI on the final score, as a percentage of the across-seed mean, against the number of seeds on Level 1. Spider Lift reaches 382% at two seeds and is clipped.

The same throughput makes hyperparameter sweeps affordable (Section H.1). Cost stays close to the bound across most configurations of PPOLag, PPOPID, and FOCOPS, so their safety is robust to these choices, whereas reward depends strongly on the shared RL hyperparameters and much less on the safety-specific ones. Lowering the PPO clip ϵ from 0.3 to 0.1 alone raises the reward of PPOLag, PPOPID, and P3O on `Point Push` from 25–34 to about 81, which overtakes FOCOPS.

6 Conclusions

We introduced CRAX, a hardware-accelerated SafeRL benchmark built on MJX that reaches two orders of magnitude higher throughput than Safety-Gymnasium, making evaluations practical that are unaffordable on CPU-bound benchmarks. Its eight tasks span eight agent morphologies and three difficulty levels, and agents can learn from state or from egocentric and allocentric camera images, with both on- and off-policy methods. Using this throughput, we benchmarked seven SafeRL algorithms, quantified how many seeds reliable comparisons require, ran broad hyperparameter sweeps, and measured how tightening the cost bound trades off against reward. We found that curriculum learning and safety transfer help on some of the hardest tasks, whereas learning from pixels lowers both reward and constraint satisfaction, with most tasks unsolved from pixels. We hope CRAX makes such large-scale studies routine for the SafeRL community.

7 Limitations and Future Work

As of the time of writing, MJX does not yet support all features available in the CPU-based MuJoCo, such as certain rigid-body collision types. This restricts the range of scenarios that can currently be expressed within CRAX. In addition, we evaluate safety exclusively through expected cumulative cost constraints, leaving other safety formulations unexplored. Finally, our evaluation does not include model-based and multi-agent safe RL approaches, which remain as future work directions.

AI USE STATEMENT

LLM-based coding assistants were used during development to help implement the CRAX environments and the JAX reimplementations of the SafeRL baselines. All generated code was reviewed, tested, and validated by the authors against reference implementations and the reported empirical results. LLMs were also used to polish the writing of this paper.

REPRODUCIBILITY STATEMENT

The code for all environments, baselines, and experiments is available at the anonymous repository linked in Section 1, including the configurations and scripts used to produce every table and figure. Appendix A specifies the reward and cost functions, difficulty levels, and agents of every task, and Appendix H lists all hyperparameters, together with the sweeps that assess their sensitivity. Section 5 describes the evaluation protocol, the number of seeds, and the hardware, and Appendix B analyzes how the number of seeds affects the reliability of the results. Because CRAX runs on a single GPU, reproducing the full evaluation takes about two weeks on one NVIDIA H100, and a single run on one environment takes minutes to hours.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Eitan Altman. *Constrained Markov Decision Processes*. Routledge, 1st edition, 1999. doi: 10.1201/9781315140223.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *IJCAI*, pp. 4148–4152. AAAI Press, 2015.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pp. 41–48, 2009.

Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. VMAS: A vectorized multi-agent simulator for collective robot learning. In *DARS*, pp. 42–56, 2022.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphaël Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In *ICLR*, 2024.

Michal Bortkiewicz, Wladyslaw Palucki, Vivek Myers, Tadeusz Dziarmaga, Tomasz Arczewski, Lukasz Kucinski, and Benjamin Eysenbach. Accelerating goal-conditioned reinforcement learning algorithms and research. In *ICLR*, 2025.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv preprint arXiv:1606.01540, 2016.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan K. Terry. Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *NeurIPS*, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL*, pp. 1–16, 2017.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - A differentiable physics engine for large scale rigid body simulation. In *NeurIPS Datasets and Benchmarks*, 2021.

Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480, 2015.

Sven Gronauer. Bullet-Safety-Gym: A framework for constrained reinforcement learning. Technical report, mediaTUM, 2022.

Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Matthew Thomas Jackson, Uljad Berdica, Jarek Luca Liesen, Shimon Whiteson, and Jakob Nicolaus Foerster. A clean slate for offline reinforcement learning. In *NeurIPS*, 2025.

Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. In *NeurIPS*, 2023.

Danial Kamran, Thiago D Simão, Qisong Yang, Canmanie T Ponnambalam, Johannes Fischer, Matthijs TJ Spaan, and Martin Lauer. A modern perspective on safe automated driving for different traffic dynamics using constrained reinforcement learning. In *ITSC*, pp. 4017–4023, 2022.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2021. URL https://arxiv.org/abs/2004.13649.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pp. 1106–1114, 2012.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

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.

Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou. MetaDrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(3):3461–3475, 2022.

Michael T. Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *ICML*, 2024.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen,

Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac Lab: A GPU-accelerated simulation framework for multi-modal robot learning. arXiv preprint arXiv:2511.04831, 2025.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

Sharada P. Mohanty, Jyotish Poonganam, Adrien Gaidon, Andrey Kolobov, Blake Wulfe, Dipam Chakraborty, Grazvydas Semetulskis, João Schapke, Jonas Kubilius, Jurgis Pasukonis, Linas Klimas, Matthew J. Hausknecht, Patrick MacAlpine, Quang Nhat Tran, Thomas Tumiel, Xiaocheng Tang, Xinwei Chen, Christopher Hesse, Jacob Hilton, William Hebgen Guss, Sahika Genc, John Schulman, and Karl Cobbe. Measuring sample efficiency and generalization in reinforcement learning benchmarks: NeurIPS 2020 Procgen benchmark. In *NeurIPS (Competition and Demos)*, pp. 361–395, 2020.

MuJoCo XLA Authors. MuJoCo XLA (MJX) - MuJoCo documentation. https://mujoco.readthedocs.io/en/stable/mjx.html, 2023. [Accessed 28-01-2026].

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, 21(181):1–50, 2020.

Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Artem Agarkov, Viacheslav Sinii, and Sergey Kolesnikov. XLand-MiniGrid: Scalable meta-reinforcement learning environments in JAX. In *NeurIPS*, 2024.

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1 edition, 1994.

Asha Ramanujam, Adam Elyoumi, Hao Chen, Sai Madhukiran Kompalli, Akshdeep Singh Ahluwalia, Shraman Pal, Dimitri J. Papageorgiou, and Can Li. SafeOR-Gym: A benchmark suite for safe reinforcement learning algorithms on practical operations research problems. arXiv preprint arXiv:2506.02255, 2025.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. arXiv preprint arXiv:1910.01708, 2019.

Julien Roy, Roger Girgis, Joshua Romoff, Pierre-Luc Bacon, and Christopher J. Pal. Direct behavior specification via constrained reinforcement learning. In *ICML*, pp. 18828–18843, 2022.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schröder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert T. Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob N. Foerster. JaxMARL: Multi-agent RL environments and algorithms in JAX. In *NeurIPS*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Aivar Sootla, Alexander I. Cowen-Rivers, Taher Jafferjee, Ziyan Wang, David Henry Mguni, Jun Wang, and Haitham Ammar. Saute RL: almost surely safe reinforcement learning using state augmentation. In *ICML*, pp. 20423–20443, 2022.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by PID Lagrangian methods. In *ICML*, pp. 9133–9143, 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. DeepMind control suite. arXiv preprint arXiv:1801.00690, 2018.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, pp. 5026–5033, 2012.

Tristan Tomilin, Meng Fang, and Mykola Pechenizkiy. HASARD: A benchmark for vision-based safe reinforcement learning in embodied agents. In *ICLR*, 2025.

Akifumi Wachi, Xun Shen, and Yanan Sui. A survey of constraint formulations in safe reinforcement learning. In *IJCAI*, pp. 8262–8271, 2024.

Tengyu Xu, Yingbin Liang, and Guanghui Lan. Crpo: A new approach for safe reinforcement learning with convergence guarantee. In *International Conference on Machine Learning*, pp. 11480–11491. PMLR, 2021.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2019.

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 in robotics. *IEEE Robotics Autom. Lett.*, 7(4):11142–11149, 2022.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. MuJoCo playground. arXiv preprint arXiv:2502.08844, 2025.

Linrui Zhang, Li Shen, Long Yang, Shixiang Chen, Bo Yuan, Xueqian Wang, and Dacheng Tao. Penalized proximal policy optimization for safe reinforcement learning. arXiv preprint arXiv:2205.11814, 2022.

Yiming Zhang, Quan Vuong, and Keith Ross. First order constrained optimization in policy space. In *NeurIPS*, pp. 15338–15349, 2020.

Markel Zubia, Thiago D Simão, and Nils Jansen. Robust transfer of safety-constrained reinforcement learning agents. In *ICLR*, 2025.

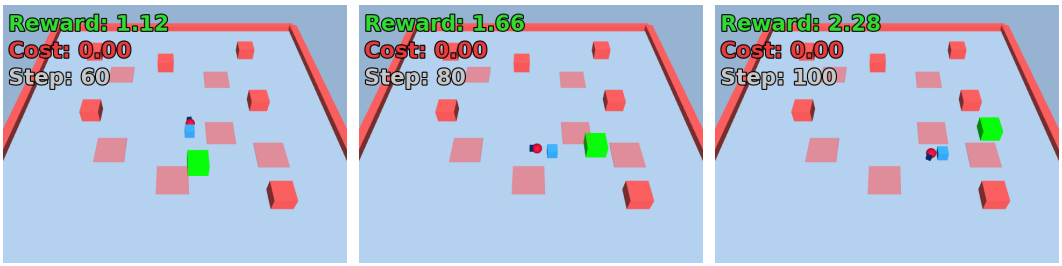


Figure 8: In levels 2 and 3 of `Push`, the goal that the agent must push the block into moves at a fixed velocity. To succeed, the agent ought to anticipate the goal's trajectory while avoiding hazard zones.

A Environment Descriptions

A.1 Task Descriptions

Table 2 lists which agents each suite accepts, and Table 6 gives the parameter varied across its difficulty levels.

A.1.1 Navigation Tasks

In the first three tasks, a mobile agent moves through an arena containing hazards. They are compatible with Point, Ant, Humanoid, and Spider.

Goal. Navigate to goal regions while avoiding hazards scattered throughout an arena. When the agent reaches a goal, it respawns at a new random location. Hazards may be collidable (blocking, with contact-based cost) or non-collidable (pass-through, with proximity-based cost). This is a continuous task with no terminal success state.

Circle. Maintain a circular orbit at a target radius around a fixed center point. The agent is rewarded for tangential velocity and penalized for deviating from the target radius. Higher levels shrink the arena boundary and add cylindrical hazards inside it.

Push. Push a movable block into goal regions while avoiding hazards. Unlike Goal, the *block* rather than the agent, must reach the goal, so the agent has to coordinate approaching the block with pushing it in the right direction. Only agent-hazard interactions incur cost, and the block passes through hazards freely. At Levels 2 and 3, the goal moves at a fixed velocity, so the agent has to anticipate its trajectory rather than steer toward its current position (Figure 8).

A.1.2 Locomotion Tasks

In these tasks, an agent must move forward while respecting a constraint on its speed, posture, gait, or foot placement.

Velocity. Move forward while keeping the speed below a threshold, which is set relative to the speed an unconstrained agent of the same morphology reaches (Table 7).

Height. Move forward while staying below a maximum height threshold, simulating a low ceiling. Cost increases smoothly as height exceeds the threshold, encouraging the agent to crouch. The Humanoid implementation is based on the `HumanoidStandup` environment with an added forward locomotion reward.

Pathway. Traverse a path with non-collidable hazard zones placed along its route. Cost is incurred when feet step inside hazard regions and scales with penetration depth, so grazing the edge of a zone is cheaper than fully landing in it (Figure 9). Hazards are randomly placed with varying gaps and lateral offsets, so the agent has to time its steps carefully. Falling incurs a terminal cost.



Figure 9: The `Pathway` agent incurs a per-step cost when its foot contacts a hazard, scaled by the penetration depth into the hazard region.

Lift. Walk forward while keeping specified legs off the ground, forcing unusual gaits. Cost is incurred each timestep a restricted foot contacts the floor.

A.1.3 Reaching Tasks

Reach. Reach a randomly placed target while avoiding flat hazards scattered on the workspace. Currently, the only compatible agent is Reacher: a planar 2-link robotic arm. Cost is incurred when any part of the arm intersects a hazard, detected by sampling discrete points along both links. Reward emphasizes proximity to the target with a bonus for reaching it.

A.2 Reward and Cost Functions

Throughout, r_t and c_t denote the reward and cost at timestep t, $d(\cdot,\cdot)$ Euclidean distance, and $\mathbf{1}[\cdot]$ the indicator function. Rewards are assembled from four recurring terms: forward progress $v_{\text{forward}} = (x_t - x_{t-1})/\Delta t$ along the agent's forward axis, a constant survival bonus r_{healthy} paid while the agent remains in a healthy state, a sparse goal reward with an optional dense shaping term $d_{t-1} - d_t$, and a quadratic control penalty on the action vector $\mathbf{a}$. Each suite instantiates a subset of these with its own weights (Table 4).

Table 4: Per-timestep reward. Superscripts bg and ab in `Push` denote the block-goal and agent-block distances.

Task	Reward
Goal	$r_t = \alpha_{\text{dist}}(d_{t-1} - d_t) + \alpha_{\text{goal}} n_{\text{reached}}$
Circle	$r_t = \alpha\, v_{\tan}\,(1 + \lvert r_{\text{actual}} - r_{\text{target}}\rvert)^{-1}$
Push	$r_t = \alpha_{\text{bg}}(d^{\text{bg}}_{t-1} - d^{\text{bg}}_t) + \alpha_{\text{ab}}(d^{\text{ab}}_{t-1} - d^{\text{ab}}_t) + \alpha_{\text{goal}}\mathbf{1}[\text{reached}]$
Velocity	$r_t = \alpha\, r_{\text{base}}$
Height	$r_t = v_{\text{forward}} + 1.0 - 0.01\lVert\mathbf{a}\rVert^2$
Pathway	$r_t = \alpha(v_{\text{forward}} + r_{\text{healthy}})$
Reach	$r_t = \alpha(1 - d_t/d_{\max})^{\gamma} + r_b\mathbf{1}[d_t < \epsilon]$
Lift	$r_t = v_{\text{forward}} + r_{\text{healthy}} - 0.5\lVert\mathbf{a}\rVert^2$

The cost formulations follow the five constraint types introduced in Section 4.3. Table 5 gives their exact form, where h_i indexes hazards, d_i and r_i are the distance to hazard i and its radius, τ a velocity threshold, and $\mathcal{F}_{\text{restr}}$ the set of restricted feet.

A.3 Agents

CRAX provides eight agent morphologies (Table 8), split into three groups. The 3D agents (Point, Ant, Humanoid, Spider) move freely in the plane and are the ones used in the navigation suite (`Goal`, `Push`, `Circle`), where they perceive goals and hazards through lidar and compass sensors rather than through privileged positions. The 2D agents (HalfCheetah, Walker2d, Hopper) are the standard planar locomotion models, restricted to the sagittal plane and used in the `Velocity` and `Pathway` tasks. Reacher is a fixed-base 2-link arm, used only in the `Reach` task. Table 2 lists which tasks are compatible with which agent.

Agents that can lose balance (Ant, Humanoid, Spider, Walker2d, Hopper) terminate the episode when the torso leaves a healthy height or angle range, following the standard MuJoCo convention.

Table 5: Per-timestep cost. `Goal` and `Push` sum contact and proximity terms over all hazards, using the cylinder or cube form depending on hazard geometry. `Velocity` uses the binary form by default.

Task	Type	Cost
Goal, Push	Contact	$c_t = \sum_i c_{\text{col}} \mathbf{1}[\text{contact}(a, h_i)]$
	Proximity (cyl.)	$c_t = \sum_i c_{\text{prox}} \max(0, 1 - d_i/r_i)$
	Proximity (cube)	$c_t = \sum_i c_{\text{prox}} \mathbf{1}[\lvert dx_i \rvert \le s \wedge \lvert dy_i \rvert \le s]$
Circle	Proximity + boundary	$c_t = c_{\text{hazard}} + c_b \mathbf{1}[\text{outside boundary}]$
Velocity	Binary	$c_t = w\mathbf{1}[v_t > \tau]$
	Hinge	$c_t = w \max(0, v_t - \tau)$
Height	Soft hinge	$c_t = w \max(0, h_t - h_{\max})/\delta$
Pathway	Quadratic	$c_t = \beta \sum_i \max_f[\max(0, 1 - d_f^{(i)}/r_i)]^2 + c_{\text{term}}\mathbf{1}[\text{fell}]$
Reach	Binary	$c_t = \beta \sum_i \mathbf{1}[\text{arm} \cap h_i]$
Lift	Binary	$c_t = \beta \sum_{f \in \mathcal{F}_{\text{restr}}} \mathbf{1}[\text{contact}(f, \text{floor})]$

Table 6: Parameter varied across difficulty levels in each suite. Level 3 of `Goal` mixes cylindrical and cubic hazards, while Levels 1 and 2 use cylinders only. `Circle` Level 1 bounds the x axis only; Levels 2 and 3 bound both axes.

Suite	Parameter	Level 1	Level 2	Level 3
Goal	Hazards (proximity + collidable)	12 + 0	8 + 8	12 + 8
Circle	Boundary extent / hazards	± 1.125 / 0	± 1.05 / 1	± 0.975 / 2
Push	Goal velocity	0.0	0.3	0.6
Reach	Hazards in workspace	4	7	10
Pathway	Max gap between hazards (m)	6.0	4.0	2.0
Height	Max CoM height (m)	1.30	1.15	1.00
Lift	Restricted feet	2 (diagonal)	3 (tripod)	4 (center only)
Velocity	Speed limit ($\times$ baseline)	1.0	0.75	0.5

Point terminates on the same mechanism, though for it the range acts as a sanity bound against the sphere being launched off the ground rather than as a fall condition. HalfCheetah and Reacher never terminate early. Observation vectors consist of joint positions and velocities with the global root coordinates removed, extended per agent with contact forces (Ant), body inertias and actuator forces (Humanoid), or lidar and compass readings (Point). The exact composition is defined in the released environment code.

A.4 Difficulty Levels

Each task varies a single dominant parameter across its three levels (Table 6). Figure 10 shows the visual effect. The `Velocity` limit is set relative to the speed an unconstrained agent reaches on the same morphology, so the absolute threshold differs per agent (Table 7).

B Seed Variance

Reinforcement learning results are notoriously sensitive to the number of seeds used to produce them (Agarwal et al., 2021). This matters especially for reliably benchmarking results. In this section, we therefore measure how sensitive our baseline results are on CRAX tasks, extending the summary in Section 5.4. We run 20 seeds on all 8 environments for 500M steps each. As described in Section 5.3, the JAX-enabled throughput makes this affordable.

Figure 11 shows what visual effect this has on the training curves. At 3 seeds, the confidence bands overlap across most of training, and the ordering between methods is not resolved. To quantify this, we take the half-width of the 95% confidence interval around the across-seed mean of the final score and express it as a percentage of that mean, so tasks with different reward scales can be compared.

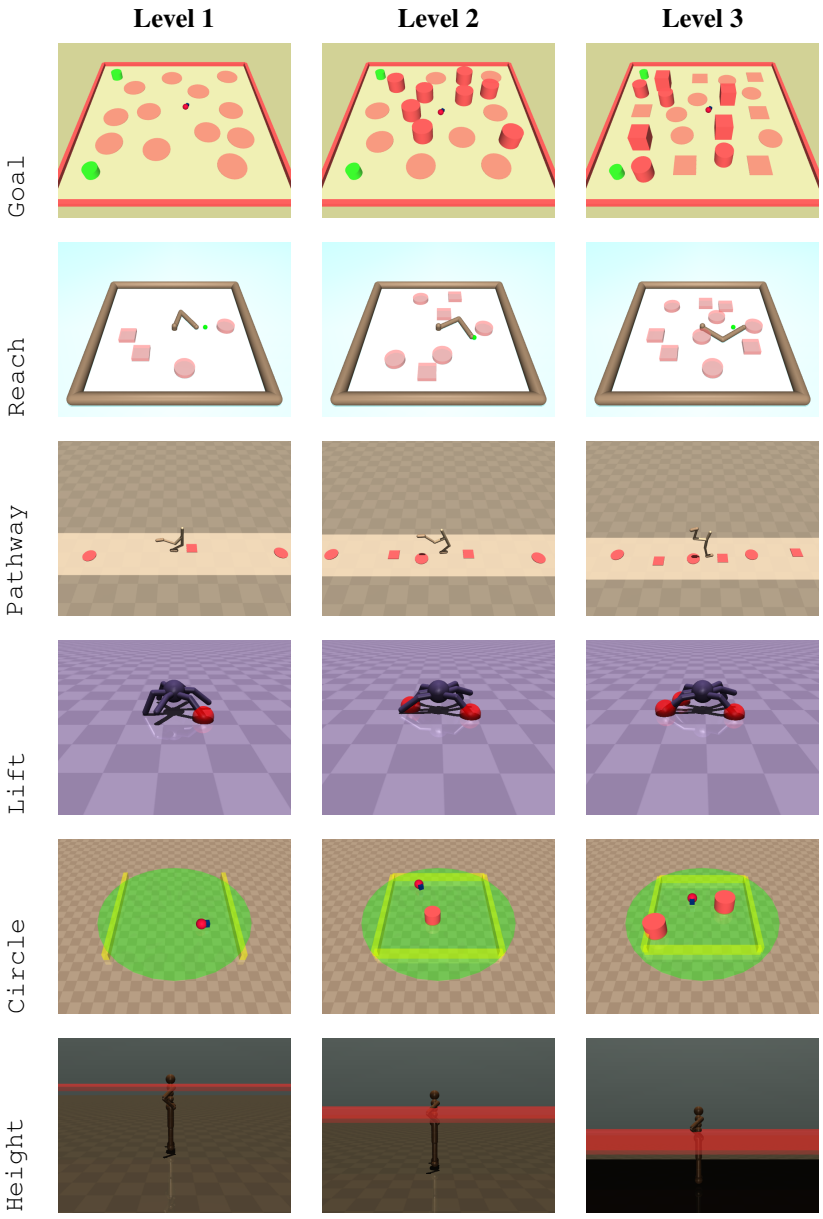


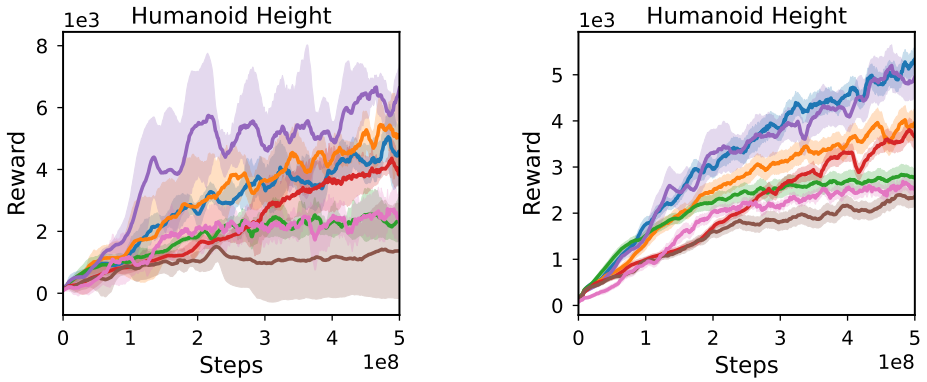
Figure 10: Visual comparison of difficulty levels across environments. Increasing difficulty generally adds more hazards, tightens constraints, or reduces margins for error.

Table 7: Speed limits in the `Velocity` task, in m/s.

	Ant	HalfCheetah	Hopper	Humanoid	Walker2d
Level 1 (1.0×)	2.62	3.21	0.74	1.41	2.34
Level 2 (0.75×)	1.97	2.41	0.56	1.06	1.76
Level 3 (0.5×)	1.31	1.60	0.37	0.71	1.17

Table 8: Agent morphologies. **Act** and **Obs** give the action and observation dimensionality, **DoF** the degrees of freedom including the root, and Δt the control timestep. **Healthy range** gives the torso height (and where applicable the torso angle) outside which the episode terminates.

Agent	Type	Act	Obs	DoF	Bodies	Δt (s)	Healthy range
Point	Holonomic sphere	2	62	3	1	0.008	0.05–0.3
Ant	3D quadruped	8	27	15	13	0.05	0.2–1.0
Humanoid	3D bipedal	17	376	23	13	0.015	1.0–2.0
Spider	3D hexapod	12	35	19	13	0.05	0.2–1.0
HalfCheetah	2D planar runner	6	18	9	8	0.05	no termination
Walker2d	2D bipedal	6	17	9	7	0.008	0.8–2.0, $\lvert\theta\rvert < 1.0$
Hopper	2D one-legged	3	11	6	4	0.008	> 0.7, $\lvert\theta\rvert < 0.2$
Reacher	2-link arm	2	11	4	3	0.02	no termination



(a) Training curves averaged over 3 seeds.

(b) Training curves averaged over 20 seeds.

Figure 11: Training curves on Level 1 with 95% confidence intervals over 3 seeds (**left**) and 20 seeds (**right**). The legend is omitted, since the point is the width of the bands rather than which method is which.

Each value averages over the eight baselines and over both reward and cost. Figure 7 shows how this metric changes as results with additional seeds are added.

Most of the gain comes early. Averaged over tasks, the interval narrows from about 90% at 2 seeds to 40% at 3 and then only to 22% at 5. Since every additional seed costs the same as the first, the return on seeds past five is small on most tasks.

Low seed counts can also understate variance. On `Reacher Reach` and `Point Push` the interval is wider at 20 seeds than at 3, because three runs happened to agree and produced a tight estimate that the next 17 contradicted. A task can therefore look settled at small n with nothing in the confidence interval to indicate otherwise.

How many seeds an environment needs, of course, depends on the learning algorithm. For instance, in our case, `Point Push`, `Spider Lift`, and `Walker Pathway` stay at around 30% at 10 seeds, while `Point Goal` and `Point Circle` are only at around 5%. We report ten seeds in the main paper, which narrows the interval to about 5% on the easiest tasks and 30% on the hardest.

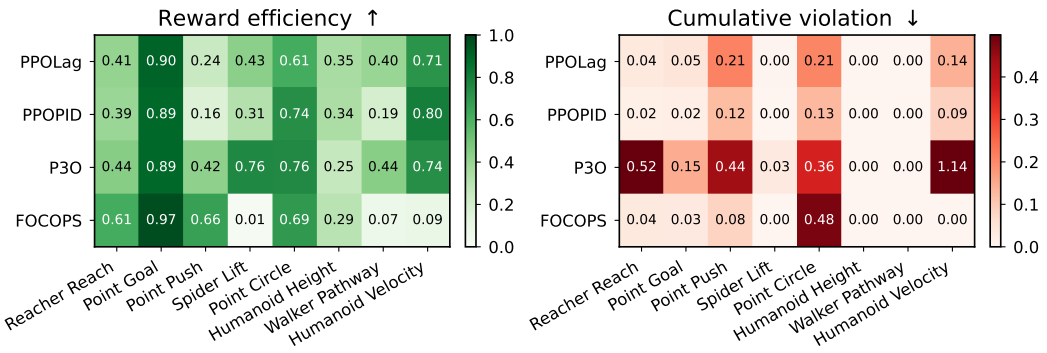


Figure 12: Efficiency and safety during training of the baselines on Level 1 tasks. **Left**: reward efficiency, the normalized area under the reward curve, where higher values indicate that a method reached strong performance earlier in training. **Right**: cumulative constraint violation during training, where zero means the method never exceeded the cost budget. Averages over 10 seeds.

C Efficiency and Safety During Training

The results in Section 5 report performance at a fixed budget of 500M environment steps, which says nothing about how quickly a method gets there or how much constraint violation it accumulates on the way. Two algorithms that end at the same reward and cost can differ substantially in how they spend the budget. We therefore report two additional metrics computed from the training curves of the same runs.

Reward efficiency (RewEff) is the area under the reward curve, normalized by the span of training and by the best final reward achieved on that task. For a policy π trained over the interval $[T_0, T_1]$, let $\hat{R}^\pi(t)$ denote its evaluated episodic return after t environment steps. Then

$$\mathrm{RewEff}(\pi) = \frac{1}{\hat{R}^\star (T_1 - T_0)} \int_{T_0}^{T_1} \hat{R}^\pi(t)\, dt, \tag{1}$$

where $\hat{R}^\star$ is the largest final return observed on that task across the compared methods, with final return taken as the mean of $\hat{R}^\pi$ over the last 10% of the curve. The unconstrained baseline is excluded from $\hat{R}^\star$, since normalizing by a method that ignores the cost budget would compress all constrained methods toward zero on the tasks where safety is expensive. The best method on each task is therefore pinned near 1 by construction, and RewEff should be read as a relative score within the compared set. Values fall in $[0, 1]$ in practice and are comparable across tasks. A method that reaches a good policy early scores close to the best final performer, while one that only catches up at the end of the budget scores lower.

Cumulative violation (CumViol) measures how much a method exceeds the safety threshold d over the course of training, rather than only at convergence. Writing $\hat{C}^\pi(t)$ for the evaluated episodic cost of π after t environment steps, we integrate the excess cost above the bound, normalized by the span of training and by the budget itself,

$$\mathrm{CumViol}(\pi) = \frac{1}{d\,(T_1 - T_0)} \int_{T_0}^{T_1} \max\left(\hat{C}^\pi(t) - d,\, 0\right) dt. \tag{2}$$

CumViol is thus the average overshoot during training, expressed as a fraction of the cost budget: a value of zero means the method stayed within the budget throughout training, and a value of 0.5 means that, averaged over training, its cost exceeded the bound by half the budget. Because the excess is measured against the same budget on every task, values are directly comparable across environments.

Figure 12 reports both metrics for PPOLag, PPOPID, P3O, and FOCOPS on Level 1. The ranking by reward efficiency differs from the ranking by final reward. P3O learns faster than both Lagrangian methods on five of the eight tasks and has the highest mean efficiency (0.59), but it also violates

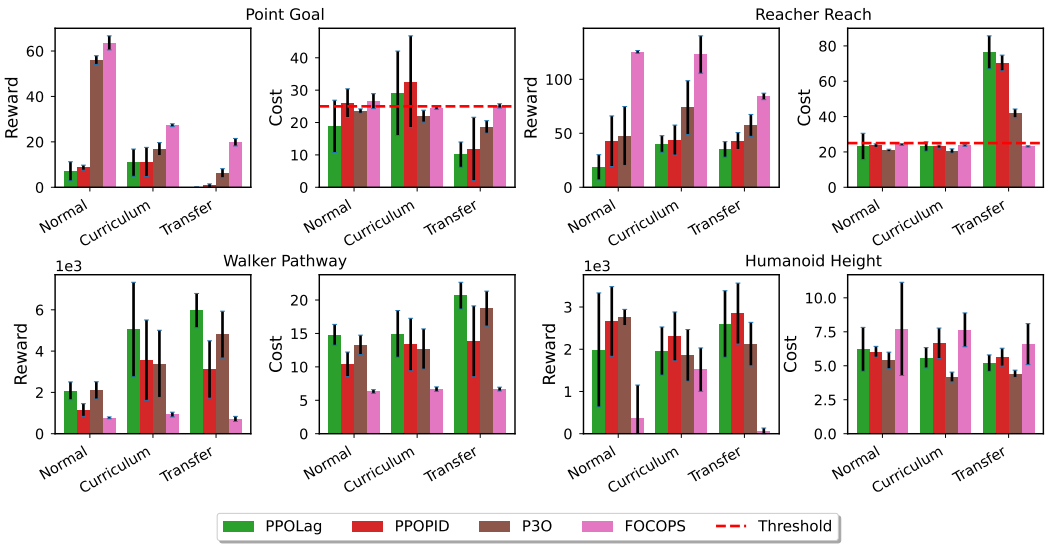


Figure 13: **Curriculum learning and safety transfer in CRAX environments.** We compare direct training (**Normal**), curriculum learning across difficulty levels (**Curriculum**), and transfer from an unconstrained PPO policy (**Transfer**) on Level 3 tasks.

the constraint most during training, with the highest cumulative violation on every task except `Point Circle` and an average overshoot of more than the full budget on `Humanoid Velocity`. PPOLag and PPOPID reach similar mean efficiency (0.51 and 0.48) but differ per task, with PPOLag ahead on `Point Push`, `Spider Lift`, and `Walker Pathway` and PPOPID ahead on `Point Circle` and `Humanoid Velocity`. PPOPID also violates less than PPOLag on every task where either violates, consistent with its design goal of damping the overshoot of the Lagrange multiplier (Stooke et al., 2020). The small gap between the two at convergence (Section 5) therefore hides different training trajectories. FOCOPS shows its affinity for navigation from the start of training, as it is the most efficient method on `Reacher Reach`, `Point Goal`, and `Point Push`, but barely learns on `Spider Lift`, `Walker Pathway`, and `Humanoid Velocity`.

Where violations occur also depends on the task. No method ever exceeds the bound on `Humanoid Height` and `Walker Pathway`, where even unconstrained PPO ends within it (Table 16), so the Level 1 threshold is not binding there, and differences between methods lie on the reward side alone. On `Humanoid Velocity`, in contrast, all methods except FOCOPS exceed the bound during training even though all of them satisfy it at the end, which final-performance tables cannot show. The navigation tasks `Point Push` and `Point Circle` are the only ones where every method violates the constraint noticeably, since the agent must first discover which behaviors are unsafe before it can avoid them.

D Curriculum Learning and Safety Transfer

As seen in Figure 21, when trained directly on the hardest difficulty level, agents often struggle to discover a good policy, as exploration becomes dominated by constraint violations and sparse progress. Training RL agents on progressively more complex settings has been shown to substantially improve learning efficiency, a paradigm commonly referred to as curriculum learning (Bengio et al., 2009; Narvekar et al., 2020). In parallel, transfer learning aims to reuse knowledge acquired in one environment to accelerate learning in a related setting (Taylor & Stone, 2009; Zubia et al., 2025).

We investigate whether curriculum and transfer learning can improve performance on the most difficult tasks in CRAX. In the curriculum setting, agents are trained sequentially on increasing difficulty levels, carrying over parameters between stages, with the data budget split equally across levels. This exposes the agent to simpler dynamics before confronting denser hazards. In the transfer setting, we first train an unconstrained PPO policy directly on Level 3 and subsequently use its

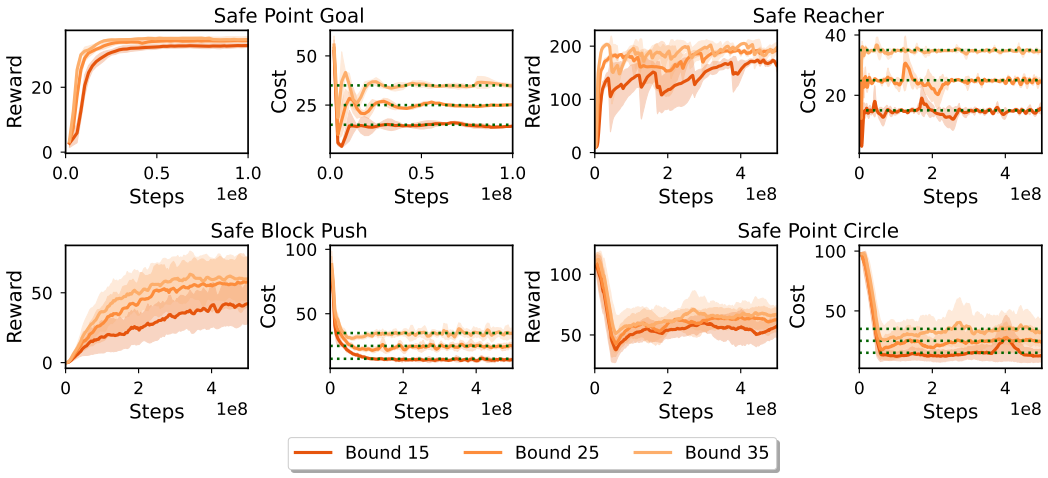


Figure 14: PPOLag increasingly sacrifices rewards to adhere to tighter safety bounds in all tasks.

parameters to initialize a safe RL algorithm, which is then trained with the remaining half of the allowed timesteps.

As shown in Figure 13, the impact of curriculum learning and transfer is strongly environment- and algorithm-dependent. In `Point Goal`, neither curriculum nor transfer improves over direct training. In `Reacher Reach`, curriculum learning boosts performance for all methods except PPOPID. Transfer proves ineffective, as all methods except FOCOPS violate the threshold after transfer, suggesting that unconstrained policies struggle to adapt for safety in this environment. Curriculum learning improves performance in `Walker Pathway` and `Humanoid Height`, whereas safe transfer yields benefits only in `Walker Pathway`. Here, direct training yields overly conservative policies that underutilize the available safety budget, while safety-transferred agents make better use of it. Overall, these results indicate that curriculum learning and transfer can be beneficial in some circumstances for learning challenging safety-constrained tasks.

E Varying Safety Bounds

Safety requirements in real-world applications can vary substantially across use cases and are often accompanied by trade-offs with performance. For instance, autonomous vehicle navigation must contend with unpredictable pedestrian behavior and sensor limitations, making perfect safety unattainable in certain circumstances. The objective instead becomes to minimize unsafe behavior. In contrast, domains such as nuclear reactor control require absolute precision and tolerate no margin for error. To support such requirements, CRAX environments expose an adjustable safety bound. We examine how varying this constraint induces safety-performance trade-offs by evaluating PPOLag under stricter and more lenient cost budgets $d \in \{15, 25 \text{ (default)}, 35\}$. As shown in Figure 14, PPOLag reliably adapts to the varied bounds across all environments, with a non-linear trade-off in performance. Tightening the bound leads to a substantially larger drop in score than the gains obtained by increasing it by the same amount. Moreover, some safe RL methods (Wachi et al., 2024) are explicitly designed to enforce strict safety guarantees rather than negotiate trade-offs, and can therefore also be evaluated under the most stringent safety bounds.

F Off-Policy Baselines

The baselines in Section 5 are all on-policy and most build on PPO. On-policy methods suit a fully compiled JAX training loop particularly well, since environment stepping, rollout collection, and the policy update compile into a single on-device program, and the regular alternation of large synchronous rollouts and updates keeps the accelerator busy. Off-policy methods fit this design less cleanly. A replay buffer requires dynamically indexed reads and writes, and the ratio of environment steps to gradient updates is less regular. They still depend on environment interaction, however, and

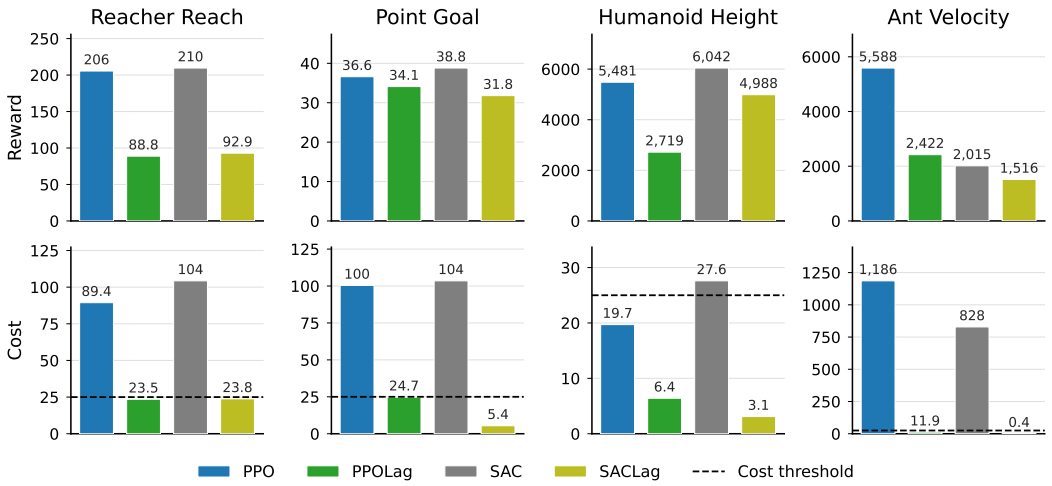


Figure 15: Final reward (top) and cost (bottom) of on-policy (PPO, PPOLag) and off-policy (SAC, SACLag) methods on four Level 1 environments. The dashed line marks the cost threshold of 25.

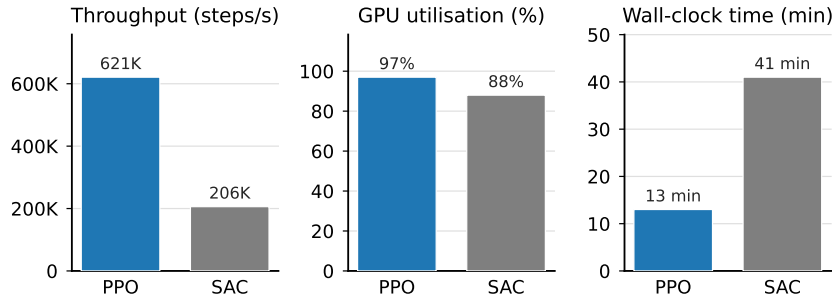


Figure 16: Training efficiency of PPO and SAC on `Reacher Reach` Level 1 for a 500M-step run on a single NVIDIA H100 GPU.

a replay buffer does not substitute for simulation throughput, so they also benefit from accelerated simulation.

To investigate how well CRAX supports off-policy learning, we implement SAC and SACLag and compare them with PPO and PPOLag on four Level 1 environments. SAC (Haarnoja et al., 2018) is an off-policy actor-critic method that maximizes an entropy-regularized return, training twin Q-critics on transitions sampled from a replay buffer. SACLag (Ha et al., 2020) adds a cost critic and a Lagrange multiplier λ, so that the actor maximizes $Q_r - \lambda Q_c$ while λ grows as long as the episodic cost exceeds the bound, mirroring the dual update of PPOLag.

Two choices keep the comparison fair. First, all methods receive the same budget of 500M environment steps. Because off-policy methods reuse each transition, matching the budget does not by itself fix the amount of learning, so we also control how often each transition is replayed. Each actor step of CRAX's SAC collects N transitions across N parallel environments and is followed by `grad_updates_per_step` gradient steps on minibatches of `batch_size`, so each transition is replayed $(\texttt{grad_updates_per_step} \times \texttt{batch_size})/N$ times on average. We set $N = 2048$, $\texttt{batch_size} = 1024$, and $\texttt{grad_updates_per_step} = 32$, which gives a replay ratio of 16 while still saturating the GPU with large synchronous rollouts. Second, the replay buffer holds the most recent 10^6 transitions rather than the full budget, since storing all 500M would be infeasible and most of them would be far off-policy by the end of training. For SACLag, the multiplier learning rate is expressed on a per-episode scale and normalized internally by the episode length, which makes it directly comparable to PPOLag's `lagrangian_coef_rate`.

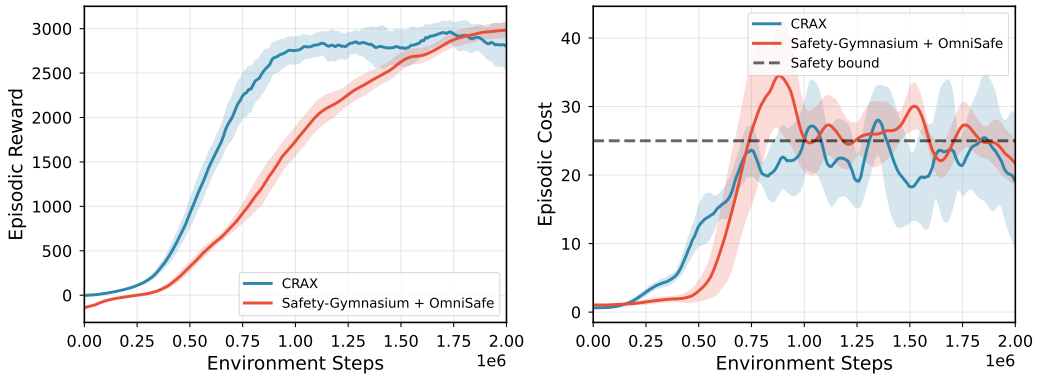


Figure 17: Safe Ant Velocity in CRAX and in Safety-Gymnasium with OmniSafe, both trained with PPO-Lagrangian for 2M environment steps over 10 seeds under a cost threshold of 25. Shaded regions denote 95% confidence intervals.

Figure 15 shows that the off-policy methods perform comparably to their on-policy counterparts, with neither family dominating. SACLag satisfies the cost bound on all four environments and exceeds PPOLag in reward on `Humanoid Height` (4,988 against 2,719) and marginally on `Reacher Reach`, while PPOLag leads on `Point Goal` and `Ant Velocity`. On these two, SACLag settles far below the bound (costs of 5.4 and 0.4), whereas PPOLag uses most of the available budget. Unconstrained SAC exceeds the bound on all four environments, including `Humanoid Height`, where PPO stays within it.

Figure 16 compares training efficiency on `Reacher Reach`. SAC reaches 206K steps per second against 621K for PPO, so a full run takes 41 minutes rather than 13, and GPU utilization drops from 97% to 88%. Off-policy training is therefore about three times slower in CRAX, but its throughput remains far above the 5K to 20K steps per second that Safety-Gymnasium reached in Section 5.3, although that measurement used a different environment.

G Fidelity to Safety-Gymnasium

The Navigation and Velocity task in CRAX descend from Safety-Gymnasium (SG). We did not aim for exact replicas of the SG versions. Instead, we made the tasks harder, since CRAX's throughput lets us train for far more environment steps, and we wish the tasks to remain a meaningful challenge for future methods. Point Goal and Point Push, for instance, place more goals and denser hazards of mixed geometry and size within a walled arena, and Point Push additionally makes the goal move at higher difficulty levels. Absolute scores on these tasks are therefore not comparable to published Safety-Gymnasium numbers.

The `Velocity` task is an exception, however. The task is fully specified by a standard MuJoCo locomotion model together with a scalar speed threshold, so it can, in principle, be aligned closely across the two backends. We use this alignment to check how faithfully CRAX reproduces SG's learning behavior. We pick the Ant agent, whose morphology both benchmarks share and whose Level 1 cost threshold of 2.62 matches the one used in `SafetyAntVelocity-v1`.

Setup. On Safety-Gymnasium we train `SafetyAntVelocity-v1` with OmniSafe's PPO-Lagrangian under its default configuration. On CRAX we train `safe_velocity_ant` with our own PPO-Lagrangian, disabling the reward scaling (Table 9) and fixing the episode length to 1000 steps so that returns are on the same scale. Both use a cost threshold of 25, a budget of 2M environment steps, and 10 seeds. Figure 17 depicts the training curves with 95% confidence intervals.

Several differences remain that we did not attempt to remove. MJX and CPU MuJoCo resolve contacts differently. The two PPO implementations do not share network sizes or optimizer defaults, and CRAX squashes actions through a `tanh` while OmniSafe samples raw Gaussian actions that

MuJoCo clips at the actuator. The comparison is between two implementations of the same task rather than two runs of the same program, so small numerical gaps are expected.

Results. Both setups converge to an episodic reward close to 3000 with episodic cost oscillating around the bound of 25, and both reproduce the same Lagrangian behavior. A few differences stand out. (1) CRAX reaches its reward plateau after $\sim 1.0M$ steps against $\sim 1.9M$ for SG. We do not claim per-step equivalence, as the contact dynamics and PPO implementations both differ. (2) CRAX overshoots the safety bound less. (3) At initialization, SG starts near -175 reward and CRAX near -10. Squashing bounds the control penalty from the first step, whereas unclipped Gaussian samples incur a large `ctrl_cost` before MuJoCo clips them. The gap closes once both policies move away from their initial action scale. This comparison is meant to serve as a sanity check on the inherited part of the benchmark.

H HYPERPARAMETERS

The network architecture, optimizer, and base PPO hyperparameters follow the Brax PPO implementation (Freeman et al., 2021). Table 9 lists the shared configuration, Table 10 the algorithm-specific settings, and Table 11 the settings of the off-policy baselines and of pixel-based training. All experiments use these values unless stated otherwise. Reported reward and cost are undiscounted episode sums from the training rollouts, logged every 10^6 environment steps.

Table 9: Shared configuration of the PPO-based methods.

Parameter	**Value**
Optimization	
Optimizer	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
Learning rate	$5 \cdot 10^{-4}$, constant
Gradient clipping	None
Entropy coefficient	$5 \cdot 10^{-3}$
Discount γ	0.99
GAE λ	0.95
Clip ϵ	0.3 (not used by FOCOPS)
Reward scaling	0.1
Advantage normalization	Yes
Value loss coefficient	0.25
Networks	
Policy	MLP, 4×32, Swish
Action distribution	Tanh-squashed Gaussian, state-dependent std (softplus $+ 10^{-3}$)
Reward critic	MLP, 5×256, Swish
Cost critic	Separate MLP, 5×256, Swish (methods with a cost critic)
Observation normalization	Running mean and standard deviation
Training	
Environment steps	$5 \cdot 10^8$
Parallel environments	2048
Unroll length	8
Batch size	1024
Minibatches per update	32
SGD epochs per batch	6
Episode length	200 (Reach), 500 (Lift), 1000 (Goal, Button, Circle, Height), 2000 (Push, Pathway, Velocity)
Cost bound d	25 per episode

H.1 HYPERPARAMETER SWEEPS

The Brax defaults were tuned for unconstrained locomotion, so we ran two sweeps on `Point Goal`, `Point Push`, and `Reacher Reach` at Level 1, varying one hyperparameter at a time over three seeds. The first varies the shared RL hyperparameters of PPOLag, PPOPID, P3O, and FOCOPS around the configuration in Table 9 for 300M steps (Tables 12 to 14). The second varies

Table 10: Algorithm-specific hyperparameters. Multiplier and penalty updates use the per-step bound d/T, where T is the episode length.

Method	Parameter	Value
PPOCost	Cost weight	1.0
PPOLag	Multiplier learning rate	3
	Initial λ	0
PPOPID	Gains (K_p, K_i, K_d)	$(10,\ 0.01,\ 0.01)$
	Integral clip / λ clip	1.0 / 10^6
	Derivative EMA β	0.95
PPOSauté	Budget discount	0.99 (equal to γ)
	Violation penalty	-1
	Normalized budget observation	Yes
P3O	Initial κ	0.01
	κ increase / decrease factor	1.1 / 0.9
	Maximum κ	50
FOCOPS	Initial ν / ν learning rate / maximum ν	0.1 / 1.0 / 100
	KL coefficient λ	1.5
	Temperature η	0.02
CRPO	Switching tolerance (per step)	0

Table 11: Configuration of the off-policy baselines (Appendix F) and of pixel-based training (Section 5.2).

Parameter	Value
SAC and SACLag	
Learning rate (policy and critics)	$5 \cdot 10^{-4}$
Entropy temperature	Tuned automatically (learning rate $3 \cdot 10^{-4}$, initial $\alpha = 1$)
Target entropy	$-\lvert\mathcal{A}\rvert/2$
Policy	MLP, 2×256, ReLU
Critics	Twin Q-networks, 2×256, ReLU (separate cost pair for SACLag)
Discount γ / target smoothing τ	0.99 / 0.005
Parallel environments	2048
Batch size	1024
Gradient updates per environment step	32
Replay buffer	10^6 transitions, FIFO, uniform sampling
Reward scaling	0.1
SACLag multiplier learning rate	0.01 per episode (divided by T)
SACLag initial / maximum λ	0 / 2
Pixel-based training	
Observation	One 64×64 RGB frame
Encoder	Nature-DQN CNN (32@8×8/4, 64@4×4/2, 64@3×3/1), ReLU, 1024 features per camera, shared by policy and critics
Heads	Policy 2×256, reward critic 2×256, cost critic 5×256, Swish
Augmentation	Random shift, 4-pixel edge padding

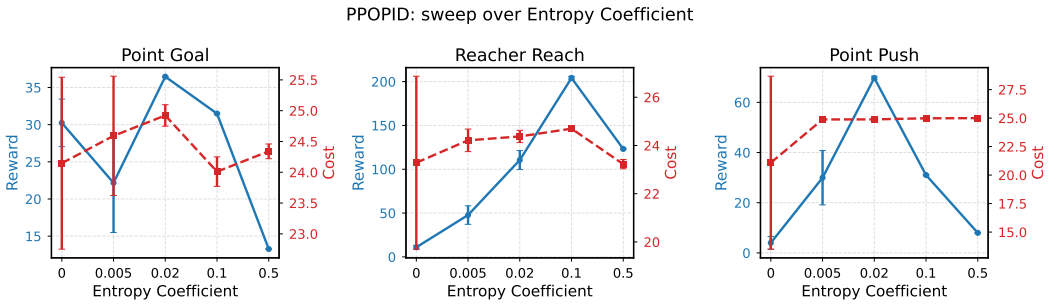


Figure 18: **Example sweep** of the entropy coefficient for PPOPID. Reward (blue, left axis) peaks at intermediate values, while cost (red, right axis) stays within about one unit of the bound of 25. Note the narrow range of the cost axis. Error bars denote 95% confidence intervals across three seeds.

the safety-specific hyperparameters of each method for 500M steps (Table 15). We aim to determine the best hyperparameters and measure their sensitivity. Figure 18 shows one sweep in detail.

Cost is largely insensitive to these choices. PPOLag, PPOPID, and FOCOPS stay within about 1.5 of the bound on `Point Goal` and `Point Push` in all but one configuration, and exceed it more on `Reacher Reach` only in a few cases, such as PPOLag without entropy regularization (32.0) and PPOPID with a GAE λ of 0.9 (31.4). P3O is less robust and exceeds the bound on `Reacher Reach` under its base configuration (32.2) and several others, up to 49.8. PPOSauté is the most sensitive method, since its budget discount decides whether it ignores the constraint, with costs of 84 to 102 at 0.95, or satisfies it at near-zero or negative reward at 0.999.

Reward is far more sensitive, mostly to the shared hyperparameters. For the three PPO-based methods, lowering the clip ϵ from 0.3 to 0.1 raises the reward on `Point Push` from between 25 and 34 to between 81 and 82, and raising the entropy coefficient from 0.005 to 0.02 raises it to between 70 and 75. A larger learning rate also helps, whereas the discount and GAE λ matter less. The safety-specific hyperparameters of these methods change reward by at most a factor of about 2.5, as for P3O's initial κ on `Reacher Reach` (31.4 to 76.5). FOCOPS is the least sensitive method, with similar reward across all settings except a zero entropy coefficient on `Reacher Reach` (22.8 against 112.9). `Point Goal` is the least sensitive task, with rewards between 26 and 38 for all methods other than PPOSauté, apart from an entropy coefficient of 0.5 for PPOPID.

Table 12: Shared RL hyperparameter sweep on `Point Goal` Level 1 (300M steps, three seeds). R: reward, C: cost. The base value of each hyperparameter is marked with *.

		PPOLag		PPOPID		P3O		FOCOPS	
Hyperparameter	Value	R	C	R	C	R	C	R	C
Learning rate	10^{-4}	31.6	24.8	32.2	24.8	26.2	24.9	36.7	24.9
	$5 \cdot 10^{-4*}$	33.9	23.8	34.0	23.9	34.3	24.7	36.8	24.8
	10^{-3}	34.9	24.8	35.4	25.0	34.8	25.5	35.6	25.1
Entropy coef.	0	31.8	23.3	30.2	24.1	31.3	24.0	37.4	24.9
	0.005*	33.9	23.8	34.0	23.9	34.3	24.7	36.8	24.8
	0.02	36.4	24.9	36.5	24.9	36.4	25.0	32.7	25.2
Discount γ	0.95	34.7	25.0	34.9	25.0	34.2	25.2	36.7	24.8
	0.99*	33.9	23.8	34.0	23.9	34.3	24.7	36.8	24.8
	0.995	34.2	24.9	33.9	25.0	31.5	24.4	36.6	25.2
GAE λ	0.9	34.7	24.7	34.1	24.3	34.1	25.1	36.9	25.0
	0.95*	33.9	23.8	34.0	23.9	34.3	24.7	36.8	24.8
	0.99	34.2	25.0	33.9	25.1	33.6	24.8	36.7	24.8
Clip ϵ	0.1	37.2	24.9	37.3	25.0	37.4	25.3	–	–
	0.2	36.2	24.9	36.0	24.9	36.1	25.2	–	–
	0.3*	33.9	23.8	34.0	23.9	34.3	24.7	–	–

Table 13: Shared RL hyperparameter sweep on `Point Push` Level 1 (300M steps, three seeds). R: reward, C: cost. The base value of each hyperparameter is marked with *.

		PPOLag		PPOPID		P3O		FOCOPS	
Hyperparameter	Value	R	C	R	C	R	C	R	C
Learning rate	10^{-4}	10.1	24.7	8.5	25.0	20.9	25.8	49.8	24.6
	$5 \cdot 10^{-4*}$	25.3	24.6	31.4	25.2	34.0	25.3	50.3	24.2
	10^{-3}	51.4	25.1	58.4	24.7	60.2	25.3	47.7	24.9
Entropy coef.	0	4.7	24.6	4.0	21.1	9.8	24.6	53.5	26.4
	0.005*	25.3	24.6	31.4	25.2	34.0	25.3	50.3	24.2
	0.02	70.1	25.0	69.7	24.9	74.5	25.7	29.3	24.8
Discount γ	0.95	33.8	25.1	38.3	25.0	39.7	25.8	43.2	24.3
	0.99*	25.3	24.6	31.4	25.2	34.0	25.3	50.3	24.2
	0.995	14.7	25.1	23.0	24.9	23.4	25.5	48.0	25.3
GAE λ	0.9	19.9	24.8	27.6	25.0	38.0	25.0	47.3	24.6
	0.95*	25.3	24.6	31.4	25.2	34.0	25.3	50.3	24.2
	0.99	21.3	25.0	30.0	25.0	31.6	25.2	48.4	25.0
Clip ϵ	0.1	81.2	24.9	81.4	25.1	82.1	25.9	–	–
	0.2	74.7	25.1	75.3	24.9	66.7	25.4	–	–
	0.3*	25.3	24.6	31.4	25.2	34.0	25.3	–	–

Table 14: Shared RL hyperparameter sweep on `Reacher Reach` Level 1 (300M steps, three seeds). R: reward, C: cost. The base value of each hyperparameter is marked with *.

		PPOLag		PPOPID		P3O		FOCOPS	
Hyperparameter	Value	R	C	R	C	R	C	R	C
Learning rate	10^{-4}	20.7	23.8	26.0	24.4	61.0	49.8	94.8	24.1
	$5 \cdot 10^{-4*}$	38.0	25.6	48.8	25.4	28.9	32.2	112.9	24.3
	10^{-3}	82.3	24.4	59.2	24.1	83.9	22.3	109.0	24.2
Entropy coef.	0	11.8	32.0	11.0	23.3	29.4	23.5	22.8	22.1
	0.005*	38.0	25.6	48.8	25.4	28.9	32.2	112.9	24.3
	0.02	117.3	24.3	110.5	24.4	65.8	22.3	111.5	24.3
Discount γ	0.95	49.8	23.5	34.7	21.8	43.0	30.4	116.8	24.4
	0.99*	38.0	25.6	48.8	25.4	28.9	32.2	112.9	24.3
	0.995	28.5	25.4	31.2	26.3	21.4	29.6	106.2	24.2
GAE λ	0.9	42.0	24.8	39.4	31.4	25.3	27.2	116.3	24.3
	0.95*	38.0	25.6	48.8	25.4	28.9	32.2	112.9	24.3
	0.99	46.9	22.8	50.6	24.3	49.0	26.9	112.1	24.2
Clip ϵ	0.1	108.7	24.3	94.6	24.1	116.4	22.1	–	–
	0.2	70.4	23.3	83.8	24.1	90.4	22.2	–	–
	0.3*	38.0	25.6	48.8	25.4	28.9	32.2	–	–

Table 15: Safety-specific hyperparameter sweep on Level 1 (500M steps, three seeds). R: reward, C: cost. Base values are marked with *.

			Point Goal		Point Push		Reacher Reach	
Method	Hyperparameter	Value	R	C	R	C	R	C
PPOLag	Multiplier LR	3*	34.5	24.5	32.1	25.2	56.8	24.0
		10	34.4	24.9	39.4	25.1	50.7	22.3
		30	34.1	25.0	23.7	24.9	40.6	28.3
PPOPID	K_p	1	34.8	28.3	37.4	23.5	81.2	25.4
		10*	34.6	25.1	20.8	25.2	67.9	24.2
		30	34.2	25.1	30.1	25.9	40.5	23.2
	K_i	0.001	34.3	25.0	38.2	24.6	62.4	24.2
		0.01*	34.6	25.1	20.8	25.2	67.9	24.2
		0.1	34.2	24.4	36.3	25.0	62.5	24.4
PPOSauté	Budget discount	0.95	36.9	99.8	54.7	84.5	194.4	102.2
		0.99*	34.6	93.6	16.6	48.4	24.8	12.1
		0.999	-0.6	0.2	-5.1	4.0	34.4	11.4
	Violation penalty	-20	-11.2	12.2	-26.4	4.5	-135.3	7.9
		-5	20.1	92.8	-12.4	7.6	-27.7	8.8
		-1*	34.6	93.6	16.6	48.4	24.8	12.1
P3O	Initial κ	0.001	33.8	24.4	57.1	25.6	66.5	23.4
		0.01*	34.5	25.2	47.9	25.3	31.4	25.4
		0.1	34.6	25.0	34.0	24.6	76.5	22.6
	κ increase factor	1.05	33.2	25.2	63.3	26.2	39.9	30.4
		1.1*	34.5	25.2	47.9	25.3	31.4	25.4
		1.3	33.8	24.5	28.8	22.5	67.5	21.1
FOCOPS	λ (KL coef.)	0.5	37.9	25.0	71.2	24.0	123.7	24.3
		1.5*	37.2	25.0	52.8	25.4	120.7	24.1
		5	34.0	24.9	36.0	24.8	120.8	24.2
	ν learning rate	0.3	37.3	24.9	55.5	24.2	123.2	24.1
		1*	37.2	25.0	52.8	25.4	120.7	24.1
		3	37.3	24.9	53.2	24.8	131.0	24.1

I Detailed Baseline Results

In this section, we provide detailed experimental results that complement the main findings.

I.1 Final Performance

Table 16 provides a comprehensive overview of the baseline results across all difficulty levels and tasks.

I.2 Training curves

Figures 19 and 21 show training curves for all baseline methods across difficulty levels, illustrating differences in learning dynamics and convergence behavior.

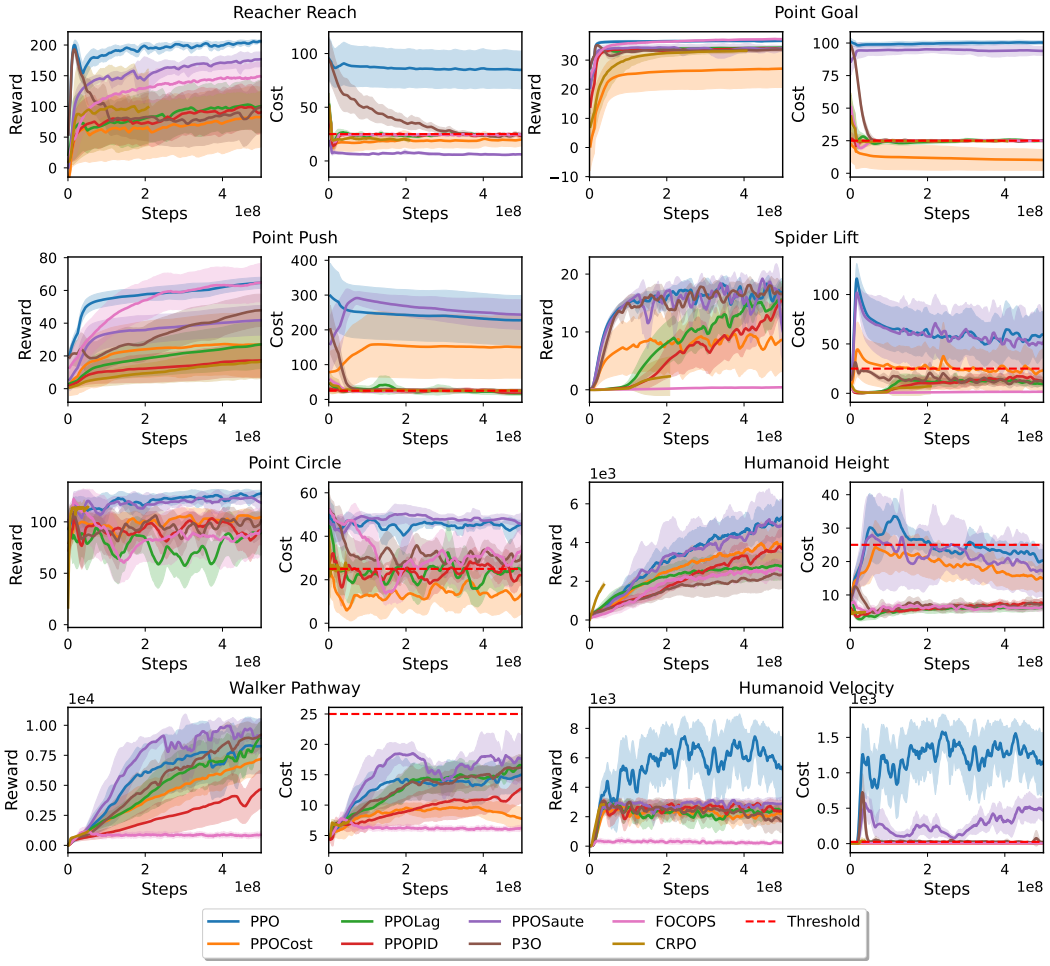


Figure 19: Level 1 training curves.

Table 16: Detailed algorithm comparison across environments and difficulty levels. R: Reward (↑ higher is better), C: Cost (↓ lower is better). Green indicates safe (cost < 25). **Bold** indicates best safe result.

	Point Goal						Reacher Reach					
	Level 1		Level 2		Level 3		Level 1		Level 2		Level 3	
Algorithm	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓
PPO	36.6	100.4	30.0	97.3	25.7	192.7	205.9	84.8	203.6	74.1	206.2	105.6
PPOCost	27.1	10.2	23.5	50.2	14.8	87.5	82.6	19.8	144.9	51.4	78.6	36.4
PPOLag	**34.3**	**24.9**	16.7	7.3	7.0	18.8	100.2	23.9	66.6	24.9	18.5	23.1
PPOPID	34.0	24.9	25.6	25.3	8.6	26.0	89.4	23.4	129.9	26.1	42.7	23.6
PPOSaute	33.9	94.0	61.3	126.9	52.7	191.0	**176.6**	**6.2**	104.2	13.1	21.8	11.6
P3O	33.6	25.2	64.1	23.4	**56.0**	**23.5**	98.6	24.6	139.3	24.7	47.5	20.8
FOCOPS	37.2	25.1	**73.3**	**24.6**	63.6	26.6	149.4	24.6	177.5	24.6	**125.4**	**24.2**
CRPO	33.2	24.8	18.9	26.1	12.1	25.9	109.9	21.3	**186.9**	**23.0**	140.0	25.0

	Walker Pathway						Humanoid Velocity					
	Level 1		Level 2		Level 3		Level 1		Level 2		Level 3	
Algorithm	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓
PPO	8262.4	15.0	**10433.8**	**21.8**	11024.5	36.7	5376.4	1153.0	5966.2	1244.7	2583.0	606.5
PPOCost	7163.5	7.8	5121.1	7.8	**4302.8**	**8.9**	2220.5	1.0	1542.4	0.4	887.3	1.5
PPOLag	8974.3	16.6	6823.0	20.0	2076.2	14.8	**2442.8**	**12.0**	1594.2	13.9	844.1	14.5
PPOPID	4633.2	12.6	4186.2	13.3	1147.7	10.4	2416.0	14.3	1258.6	9.0	**902.5**	**4.2**
PPOSaute	9095.2	17.7	6978.6	19.5	9304.0	35.1	2866.5	480.5	1372.0	267.3	972.3	220.5
P3O	**9137.1**	**16.2**	7460.7	19.2	2098.4	13.3	1625.6	19.3	**1785.6**	**18.1**	856.4	19.7
FOCOPS	876.5	6.2	856.3	6.8	769.1	6.3	228.3	3.1	119.8	3.3	79.1	4.5
CRPO	1147.4	7.3	785.3	7.0	628.0	7.3	2824.4	29.0	2953.1	30.0	2845.7	36.3

	Spider Lift						Point Push					
	Level 1		Level 2		Level 3		Level 1		Level 2		Level 3	
Algorithm	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓
PPO	16.8	59.5	17.0	59.1	15.6	135.9	64.5	227.7	54.6	368.1	49.5	461.3
PPOCost	8.6	23.0	16.3	56.7	17.5	115.8	26.9	150.8	40.1	333.0	39.5	418.0
PPOLag	15.5	9.6	4.9	8.8	0.0	1.4	26.6	17.6	10.9	24.4	5.8	24.9
PPOPID	15.4	12.4	10.2	14.7	0.0	1.6	17.2	19.9	7.4	25.2	5.9	24.5
PPOSaute	11.7	49.8	13.0	76.1	17.0	107.2	41.7	244.3	34.1	279.7	25.0	293.6
P3O	**16.6**	**12.1**	**13.6**	**14.5**	**0.1**	**1.4**	47.7	26.7	14.1	26.3	13.9	25.6
FOCOPS	0.4	1.6	0.1	1.0	0.0	1.3	**64.3**	**24.4**	**40.5**	**24.6**	**40.8**	**24.3**
CRPO	9.4	6.7	0.2	0.9	0.0	1.1	17.3	25.0	7.7	26.3	10.1	25.2

	Point Circle						Humanoid Height					
	Level 1		Level 2		Level 3		Level 1		Level 2		Level 3	
Algorithm	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓	R ↑	C ↓
PPO	127.0	45.0	127.2	98.8	127.5	99.4	**5230.3**	**20.2**	3532.3	52.1	3868.4	53.8
PPOCost	103.8	11.9	82.8	9.3	80.8	38.1	3873.8	15.0	3657.3	33.8	3753.8	57.5
PPOLag	89.9	25.3	64.5	24.2	53.5	18.7	2792.9	6.4	1703.3	5.5	1983.7	6.2
PPOPID	90.5	21.8	68.9	26.5	39.9	21.6	3716.6	7.6	1087.3	5.4	2657.9	6.0
PPOSaute	120.2	46.0	128.6	97.3	123.1	96.8	4847.5	17.3	5652.4	60.9	5910.1	35.7
P3O	96.7	29.0	60.1	28.3	61.3	27.5	2328.8	6.9	**2310.9**	**4.5**	**2753.3**	**5.4**
FOCOPS	88.7	32.5	62.2	25.1	**57.7**	**24.0**	2563.5	6.4	1244.1	6.9	376.4	7.7
CRPO	**114.5**	**24.7**	**88.4**	**24.4**	67.4	25.3	1847.4	4.9	2013.3	5.5	1419.9	5.3

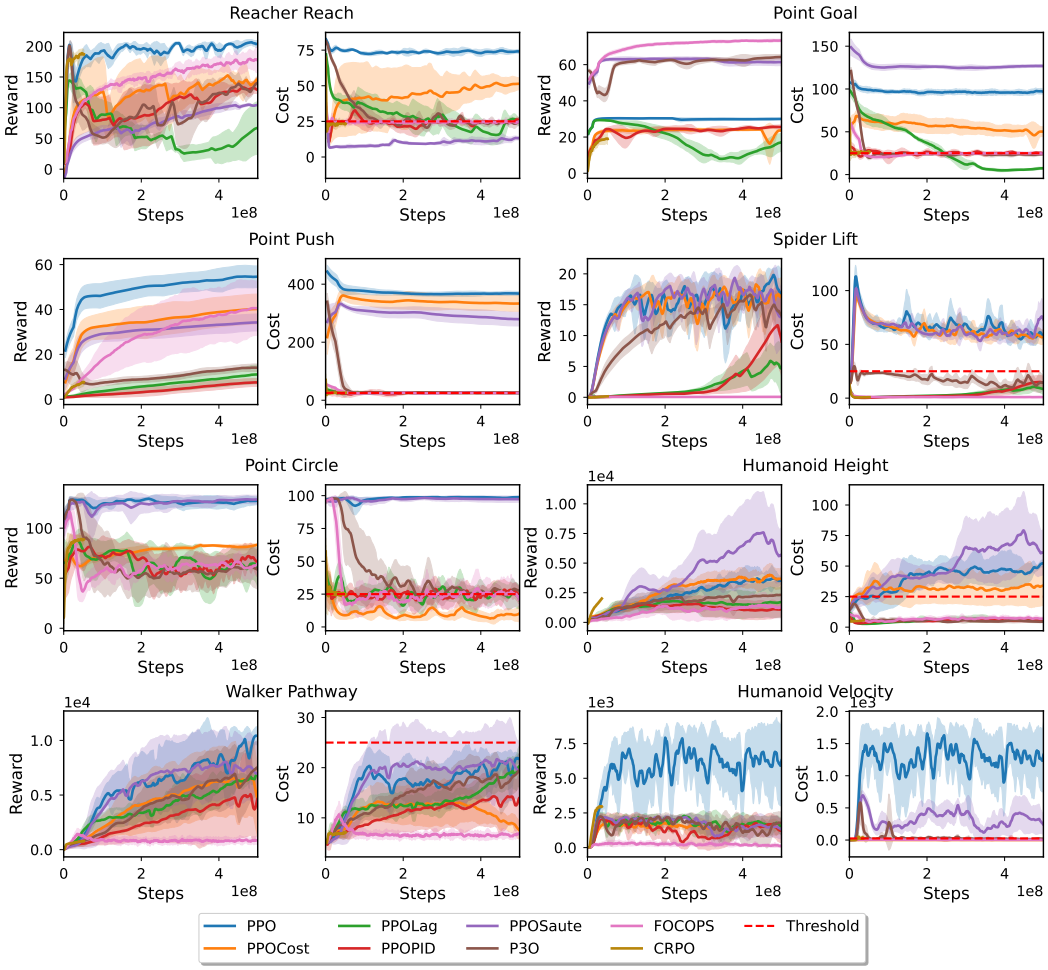


Figure 20: Level 2 training curves.

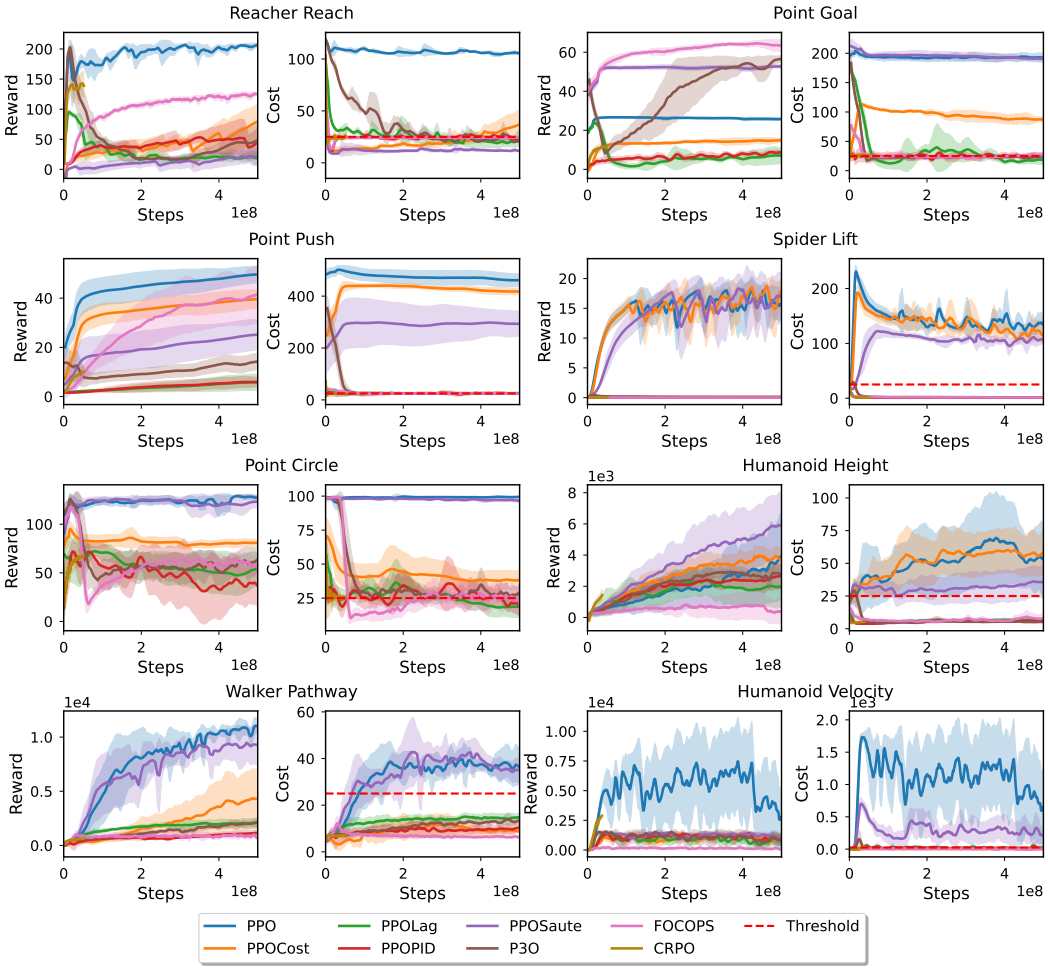


Figure 21: Level 3 training curves.