RL Applications

Algorithms for Envrironments where the state and action space are so large that it is computationally impossible to calculate alle Q-values, one will always a neural network to approximate the $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ by $NN : \mathbb{R}^{d \times l} \to \mathbb{R}$ where $d < |\mathcal{A}|$ and $l < |\mathcal{S}|$.

1 Implementation

To conduct a comprehensive and reproducible evaluation of various reinforcement learning algorithms, we structured our project as a modular Python-based framework. The entire evaluation pipeline is available at https://github.com/eelisee/reinforcementlearning/tree/main/evaluation_study. The framework is organized into configuration files, training scripts, evaluation utilities, and algorithm wrappers.

1.1 Classic Control Environments

We evaluated all algorithms on a selection of classic control environments provided by OpenAI Gym, specifically `CartPole-v1`, `Acrobot-v1`, `Taxi-v3`, `Pendulum-v1` and `MountainCarContinuous-v0`. Environments were categorized as either discrete-action or continuous-action, which determined the subset of algorithms evaluated per environment. Below we provide a concise description of each environment, highlighting its objectives, action and observation space, and reward dynamics.

CartPole-v1 In this environment, the agent must balance a pole attached to a moving cart by applying forces to the left or right. The observation space is a 4-dimensional continuous vector that includes the cart position and velocity, and the pole angle and angular velocity. The action space is discrete with two actions: push cart left or push cart right. A reward of $+1$ is given for every timestep the pole remains upright (i.e., the angle is within a threshold), up to a maximum of 500. The episode terminates early if the pole falls beyond a certain angle or the cart moves out of bounds. The task is considered solved when the agent consistently achieves near-maximum episode duration. CartPole is often considered a benchmark for testing stability in early-stage learning. The CartPole environment consists of a pole attached to a cart moving along a 1D track. The agent must apply discrete forces (left or right) to balance the pole upright. The 4D observation space comprises:

- Cart position $\in [-4.8, 4.8]$
- Cart velocity (unbounded but typically $\in [-inf, inf]$)
- Pole angle $\theta \in [-0.418, 0.418]$ radians ($\approx 24°$)
- Pole angular velocity (unbounded)

The action space is discrete: {0: push left, 1: push right}. Episodes start near the upright position with small random perturbations.

Acrobot-v1 Acrobot is a two-link, underactuated robot resembling a gymnast swinging on a bar. The agent must apply torque to the second joint (elbow) to swing the tip of the second link above a target height. The observation space is a 6-dimensional continuous vector representing the cosines and sines of the two joint angles and their angular velocities. The action space is discrete with three values: apply -1, 0, or $+1$ torque. The reward is always -1 per timestep, incentivizing the agent to reach the goal as quickly as possible. The

environment ends when the tip reaches the target height. The objective is to swing the lower link such that the tip of the upper link reaches a vertical height (position $y \geq 1.0$). The 6D observation includes:

- $\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2)$
- Angular velocities $\dot{\theta}_1, \dot{\theta}_2 \in [-4\pi, 4\pi]$

The agent always starts from the hanging-down position with zero velocity. A constant reward of -1 per step penalizes time; episodes terminate when the goal height is achieved or after 500 steps. Thus, higher performance corresponds to fewer steps to reach the target. This task is notably more challenging due to the delayed rewards and dynamics involving momentum buildup.

Taxi-v3 Taxi is a discrete environment where an agent controls a taxi in a 5x5 grid world, picking up and dropping off passengers at designated locations. The observation space is a single integer encoding the taxi's location, passenger position, and destination (500 discrete states). The action space is also discrete with six actions: move in four directions, pickup, and drop-off. The reward function penalizes each timestep with -1, applies -10 for illegal pick-up/drop-off actions, and gives +20 for successful drop-off. Although conceptually simpler than others, Taxi is useful for testing algorithms in a fully discrete MDP setting with sparse rewards and delayed success signals. The state space has 500 discrete states, encoding:

- Taxi position (25 combinations)
- Passenger location (5 possibilities)
- Destination location (4 possibilities)

Optimal episodes are short and efficient, while poor policies may accrue high penalties. Maximum episode length is 200.

Pendulum-v1 In this continuous control environment, the agent must swing a pendulum upright and keep it balanced. The observation space is a 3D vector consisting of cosine and sine of the pendulum angle and its angular velocity. The action space is a 1D continuous torque value in the range $[-2, 2]$. The reward at each step is calculated as $-(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$, where θ is the angle, $\dot{\theta}$ is the angular velocity, and a is the applied torque. Thus, the optimal policy minimizes deviation from the upright position while applying minimal force. The 3D observation vector is:

- $\cos(\theta), \sin(\theta)$ (angle)
- $\dot{\theta} \in [-8, 8]$ (angular velocity)

The action space is continuous: torque $a \in [-2, 2]$. The pendulum starts at a random angle $\theta \in [-\pi, \pi]$ with initial angular velocity near 0. This yields a minimum reward near -16 and maximum near 0 per step. Episodes last for 200 steps. Better policies maintain the pendulum close to the upright position with minimal torque usage. This task is used to test the performance of algorithms in low-dimensional continuous action spaces.

MountainCarContinuous-v0 This continuous version of MountainCar requires the agent to drive a car up a steep hill. The car lacks sufficient power to climb the hill in one go and must build momentum by moving back and forth. The observation space consists of two values: position and velocity of the car. The action space is one-dimensional and continuous, representing the force applied ($[-1, 1]$). The reward is computed based on velocity and reaching the goal state, with a bonus of $+100$ for reaching the flag at the top of the hill. The episode ends upon reaching the goal or after 999 steps. The observation space is 2D:

- Position $x \in [-1.2, 0.6]$
- Velocity $\dot{x} \in [-0.07, 0.07]$

The car starts near position -0.6 with zero velocity. Rewards are shaped:

$$r = 100 \cdot \mathbb{1}_{x>0.45} - 0.1a^2$$

Thus, agents receive a large bonus upon reaching the goal at $x > 0.45$, and are penalized for energy usage. Episodes run for up to 999 steps, though successful policies solve it in under 200. Early plateauing indicates suboptimal exploration strategies. The environment tests the agent's ability to explore and exploit continuous control policies under sparse rewards.

1.2 Configuration and Experiment Execution

Each environment has an associated YAML configuration file (e.g., `Acrobot-v1.yaml`) specifying:

- **Timesteps:** 1,500 for discrete environments; 50,000 for continuous control
- **Evaluation frequency:** every 100 timesteps
- **Evaluation episodes:** 10 episodes per evaluation phase
- **Learning rate:** 0.001 for REINFORCE variants
- **Discount factor:** $\gamma = 0.99$
- **Batch size** of 64 for minibatch methods

The main experiment loop is initiated via `main.py`, which internally invokes `run_all.py` to parse configurations and execute training routines for each specified algorithm.

1.3 Algorithm Implementations

We include implementations of two baseline algorithms: **REINFORCE** and **Mini-batch REINFORCE**. These are implemented from scratch under `models/reinforce/`. The minibatch variant uses a batch size of 64, allowing for more stable gradient updates.

For state-of-the-art algorithms, we use implementations from `Stable-Baselines3` and `SB3-Contrib`, wrapped in a consistent training interface in `models/sota/`. The algorithms were evaluated: ARS, A2C, PPO, TRPO, DDPG, SAC, TD3 and TQC.

Each wrapper follows a standard interface: instantiating the model with a policy network, training it over the configured timesteps using a specified seed, and periodically evaluating it via an `EvalCallback`. Evaluation returns and timesteps are saved to CSV for downstream aggregation and plotting.

Each training run is repeated for multiple random seeds (typically two per configuration, e.g., seed 0 and 1), enabling statistically meaningful comparisons. All environments are seeded consistently using utilities defined in `utils/seed_utils.py`, ensuring reproducibility across evaluations.

These parameters were kept consistent across algorithms to ensure fair comparison. In the case of continuous action environments (e.g., Pendulum, MountainCarContinuous), only continuous-compatible algorithms were evaluated.

During training, performance is periodically evaluated and logged using the `EvalCallback` from Stable-Baselines3. The rewards collected during these evaluations are saved and later plotted to visualize the learning progression of each algorithm on each environment. These plots are stored in `plots/` as PNG files named according to their environment.

Results from these experiments were stored uniformly and form the basis for our subsequent analysis. We tried to use the rliable package, but found an unsolvable import-error. For that reason, we implemented the following functions by hand:

- Interquartile Mean (IQM) across all runs: Performance on middle 50% of combined runs
- Interval estimates using stratified bootstrap confidence intervals (CIs)
- Score distributions (performance profiles)

2 Results

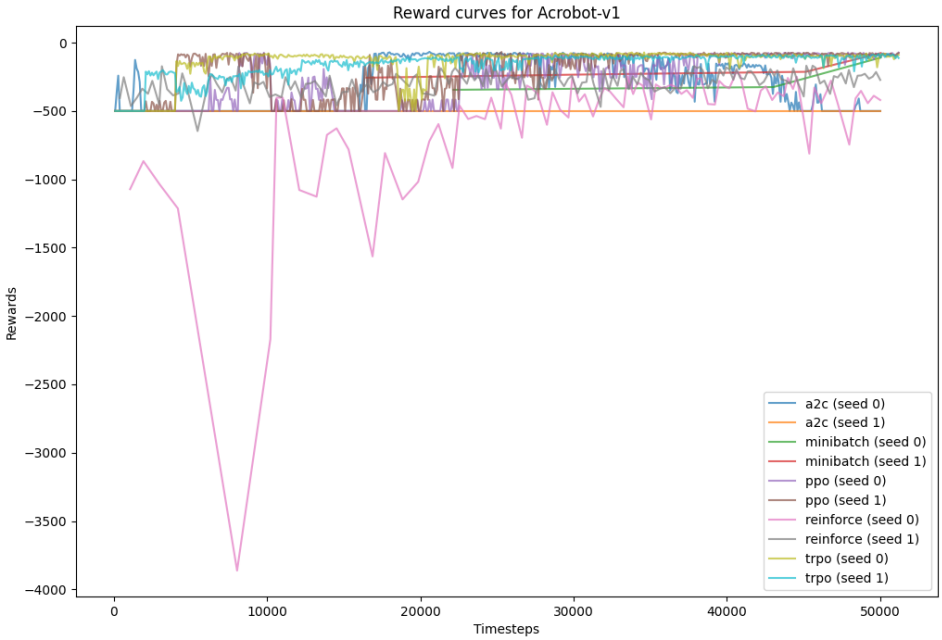

Figure 0.1: Average episodic return across training timesteps for different algorithms on `Acrobot-v1`.

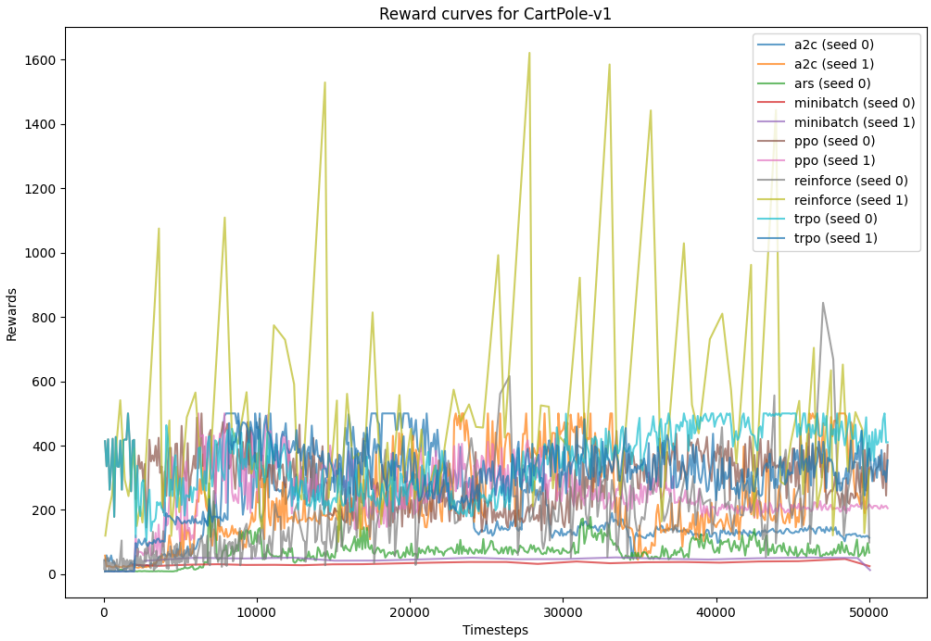

Figure 0.2: Reward progression over time for evaluated algorithms on `CartPole-v1`. A plateau near 500 indicates optimal performance.

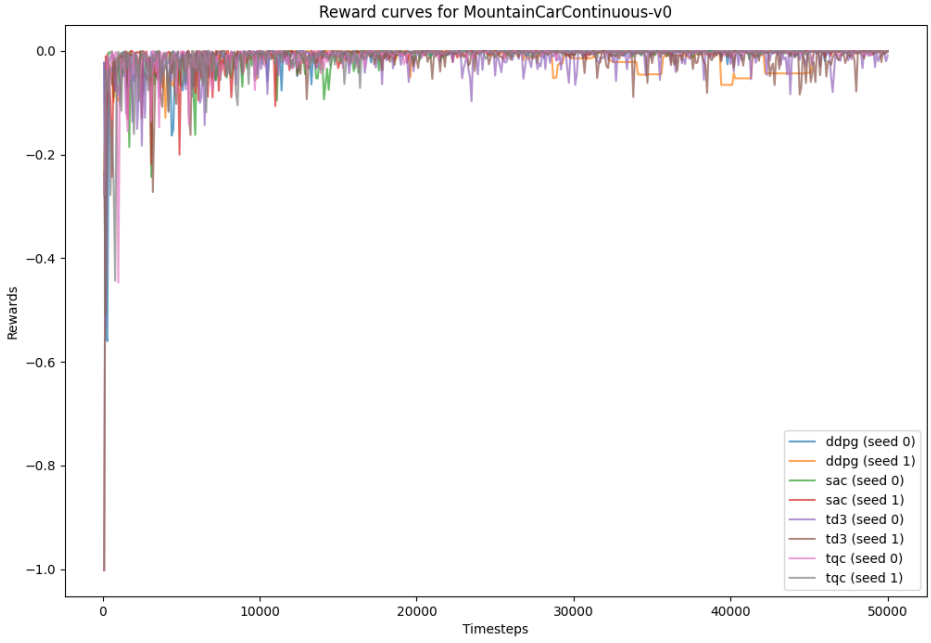

Figure 0.3: Reward evolution for continuous control agents on `MountainCarContinuous-v0`. Sharp jumps indicate successful goal-reaching.

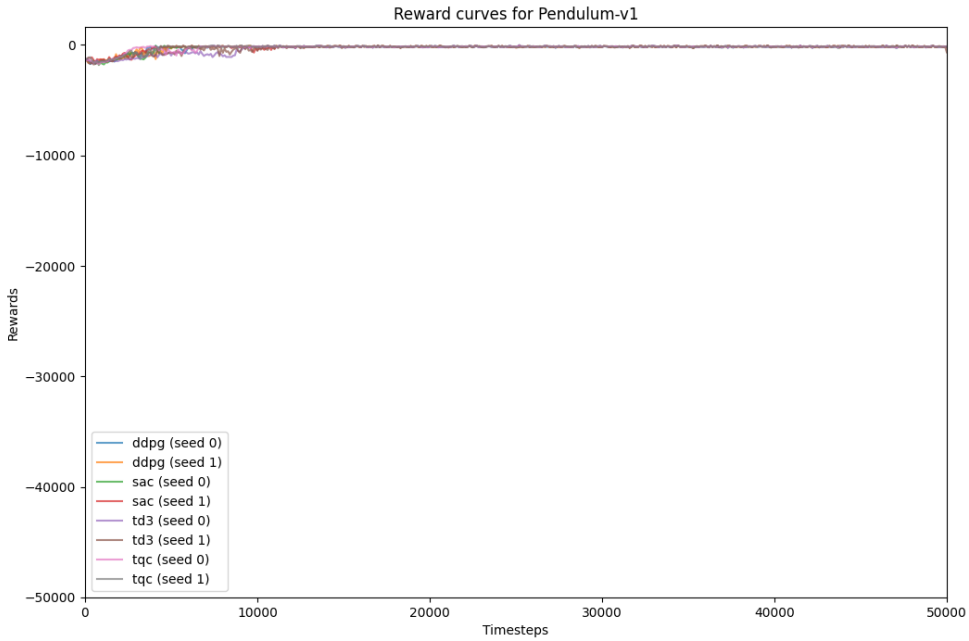

Figure 0.4: Performance trends on `Pendulum-v1`. Higher (less negative) values indicate better stabilization.

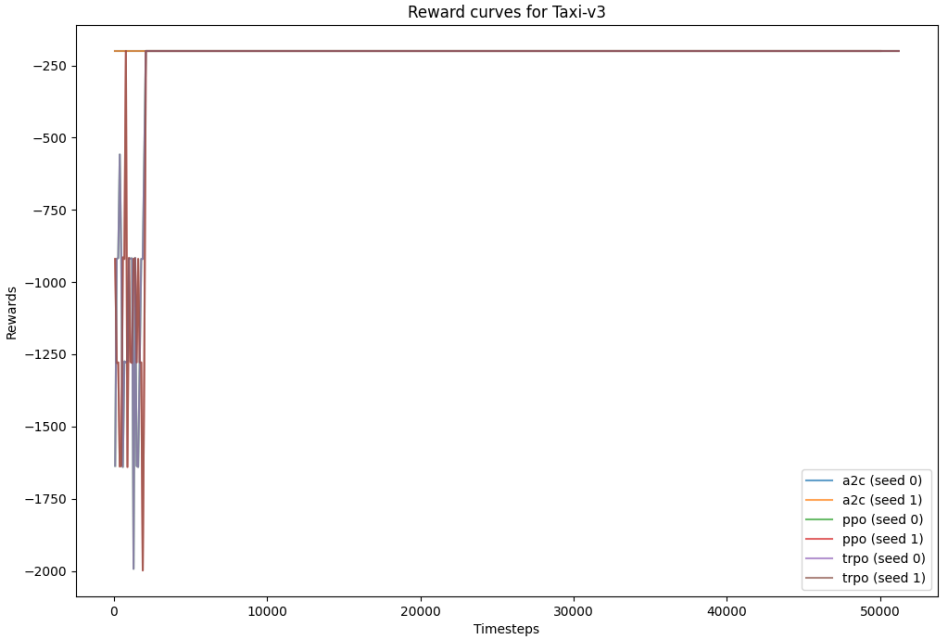

Figure 0.5: Reward development on the discrete `Taxi-v3` environment. Spikes reflect successful pickups and drop-offs.

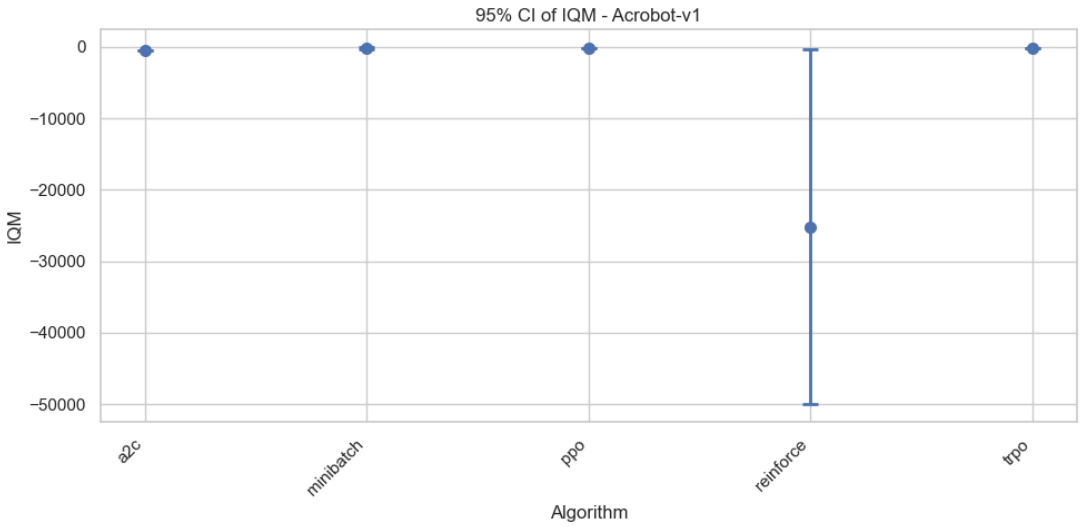

Figure 0.6: IQM development on the `Acrobot-v1` environment. The vertical lines show the confidence interval.

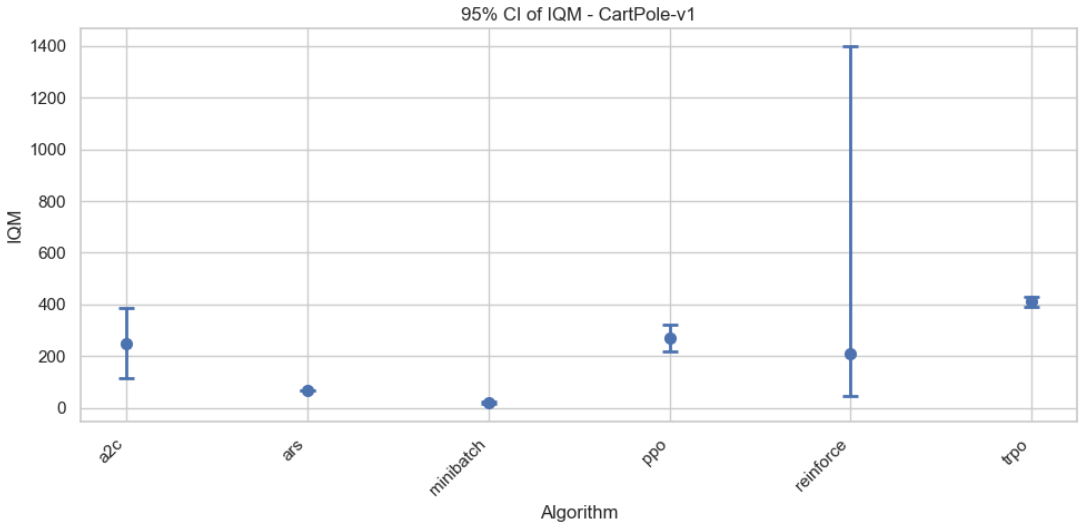

Figure 0.7: IQM development on the `CartPole-v1` environment. The vertical lines show the confidence interval.

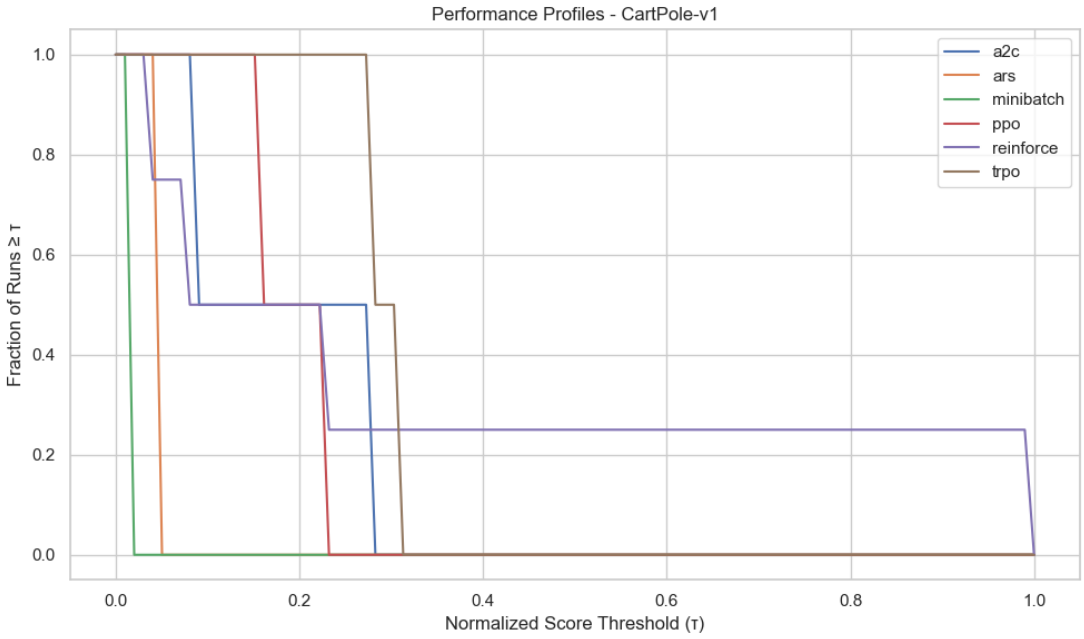

Figure 0.8: Development of the `CartPole-v1` environment. Fraction of runs (for each algorithm) that achieve at least a normalized reward threshold τ.

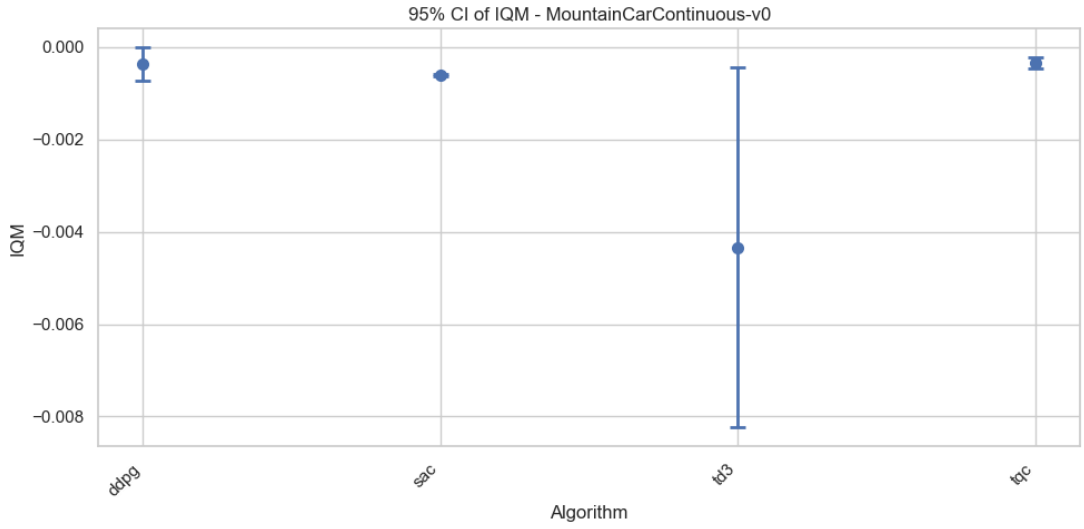

Figure 0.9: IQM development on the `MountainCarContinuous-v1` environment. The vertical lines show the confidence interval.

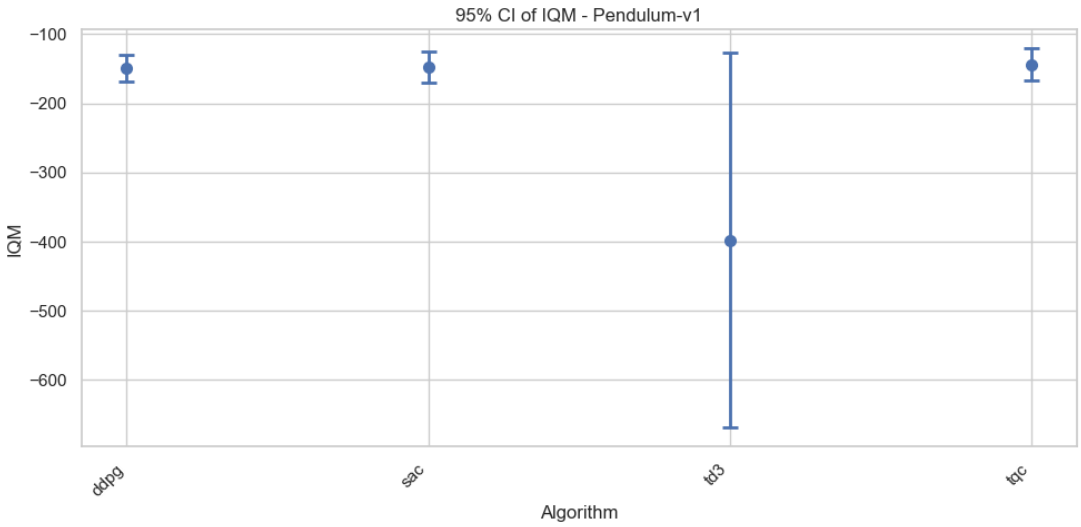

Figure 0.10: IQM development on the `Pendulum-v1` environment. The vertical lines show the confidence interval.

3 Analysis

3.1 Evaluation of the Algorithms

Reward Progression Over Time

Before analyzing statistical summaries such as IQM and performance profiles, we first turn to the raw learning curves to understand how different algorithms behave during training across environments (Figures 0.1–0.5). These plots reveal rich dynamics in reward progression that offer insights into each algorithm's stability, sample efficiency, and sensitivity to initialization.

A recurring and striking pattern across multiple environments is the erratic behavior of REINFORCE. Its learning curves frequently exhibit dramatic spikes and drops, particularly early in training. For instance, on `Acrobot-v1` (Figure 0.1), REINFORCE suffers an early collapse to a reward as low as -3800 before partially recovering. While some degree of convergence is eventually achieved in certain environments, others such as `CartPole-v1` (Figure 0.2) show continued volatility with no clear trend towards stability. This instability is expected, as REINFORCE lacks variance reduction mechanisms and is highly susceptible to high-variance gradient estimates, especially in environments with sparse or delayed rewards.

Other policy-gradient methods such as A2C and PPO display more structured learning trajectories, though not without irregularities. In `CartPole-v1`, A2C demonstrates alternating phases of high and low performance, often hovering near optimal values before abruptly regressing, a pattern mirrored by PPO to some extent. Such oscillations may stem from high sensitivity to hyperparameters or insufficient entropy regularization, which can cause premature policy convergence or erratic updates.

A noteworthy phenomenon is the presence of consistently underperforming seeds for certain algorithms. For example, in `Acrobot-v1`, A2C has a seed that remains stuck at -500 throughout training, while in `Taxi-v3`, another A2C run remains flatlined at -200 (Figure 0.5). These cases suggest that specific initializations or local minima may prevent learning altogether, revealing limitations in exploration or adaptability.

Some algorithms stand out for their stability and rapid convergence. TRPO in `Acrobot-v1` exhibits a steep improvement early on and maintains high performance thereafter, demonstrating the effectiveness of trust-region constraints in avoiding catastrophic updates. Similarly, SAC in `MountainCarContinuous-v0` shows a

consistent ascent towards optimal performance with minimal fluctuation, likely due to its entropy-maximizing objective, which promotes robust exploration and smoother updates.

Perhaps the most dramatic learning dynamics are observed in `Pendulum-v1` (Figure 0.4). Across all algorithms, rewards begin at extremely low levels (as low as $-50,000$) but experience a steep increase within the first few thousand steps, reaching around $-2,000$. This rapid initial improvement is followed by a more gradual convergence over the next 12,000 steps. The sharp early rise suggests that even minimal policy structure allows agents to learn coarse motor control, while fine-tuning to stabilize the pendulum takes significantly more iterations.

Taken together, these learning curves illustrate that while some algorithms achieve robust performance, their success can vary dramatically across environments and even across seeds. Instability, slow convergence, or early plateaus are common challenges, particularly for algorithms lacking strong regularization or exploration strategies. These observations highlight the value of robust evaluation metrics, such as IQM and performance profiles, to complement raw reward curves and better assess overall algorithmic reliability.

Interpreting the CI-IQM Plots

The CI-IQM plots visualize the interquartile mean (IQM) of the final performance (i.e., average return) achieved by each algorithm across different seeds, evaluated at a target timestep (typically 50,000). For each environment, these plots offer a statistically grounded summary of performance variability and typical behavior.

- **X-axis:** Lists all algorithms evaluated for a given environment.
- **Y-axis:** Shows the IQM of final rewards. IQM computes the mean over the middle 50% of scores, thus ignoring both low and high outliers.
- **Error bars:** Represent 95% confidence intervals (CIs) computed via bootstrapping over the seeds.
- **Point estimate:** The central dot denotes the median IQM estimate from the bootstrap distribution.

Interpretation: Higher IQM values imply better and more reliable typical performance. Narrower confidence intervals suggest consistent performance across runs, whereas wide CIs indicate significant variance. If the CIs of two algorithms overlap, the difference in their typical performance may not be statistically significant.

These plots help filter out noise from outlier-heavy or heavy-tailed reward distributions, focusing the analysis on reproducible performance.

Understanding Performance Profiles

The performance profiles show how reliably different algorithms achieve varying degrees of normalized performance. Each curve represents the cumulative distribution function (CDF) of normalized final rewards per algorithm.

- **X-axis (τ):** Normalized reward thresholds between 0 and 1, where 1 denotes the best observed score across all runs and algorithms in that environment.
- **Y-axis:** Fraction of runs (across all seeds) for a given algorithm that achieved a normalized reward of at least τ.
- **Each curve:** Represents one algorithm, summarizing how often it reaches various performance levels.

Interpretation: Algorithms whose curves remain closer to the top-right corner are more robust—they consistently achieve high performance across seeds. A sharp drop-off indicates that an algorithm only rarely reaches high scores, which can be a sign of instability or sensitivity to initialization. Comparing curves allows us to see which algorithms are not only strong in the best case, but also consistently good.

Empirical Analysis and Interpretation

IQM Performance on `Acrobot-v1` Figure 0.6 shows the Interquartile Mean (IQM) of several policy-gradient algorithms on the `Acrobot-v1` environment. Notably, all algorithms except REINFORCE converge to nearly identical IQM values close to zero, with narrow confidence intervals. This suggests stable but limited learning progress, likely due to the sparse reward structure of the environment. In contrast, REINFORCE exhibits a drastically lower mean around -25,000, with a confidence interval ranging from 0 to -50,000. This extreme variance and underperformance can be attributed to REINFORCE's lack of variance reduction techniques (e.g., baseline subtraction or entropy regularization), leading to unstable updates in environments where delayed rewards make credit assignment difficult.

IQM Performance on `CartPole-v1` Figure 0.7 presents a more differentiated outcome. Minibatch REINFORCE and ARS exhibit poor performance with mean returns around 25 and 75, respectively, reflecting their limited ability to stabilize pole balancing. TRPO significantly outperforms all others with an IQM close to 400 and a very tight confidence interval, underscoring the advantage of trust region optimization in stabilizing updates. PPO and A2C follow with moderate IQMs (225 and 250), but A2C shows higher variance, possibly due to higher sensitivity to hyperparameters or poorer exploration. REINFORCE again displays the widest confidence interval, indicating inconsistent performance and high sensitivity to initialization or noise.

Performance Profile on `CartPole-v1` The performance profile in Figure 0.8 reveals complementary insights. Minibatch REINFORCE and ARS both drop to zero at very low thresholds ($\tau < 0.05$), showing that they almost never achieve meaningful performance. REINFORCE has a multi-stepped decline, reflecting high variability across seeds, with some runs succeeding completely ($\tau = 1.0$) and others performing poorly. A2C and PPO show better consistency, achieving moderate thresholds ($\tau \approx 0.25$) across all seeds. TRPO stands out by maintaining high scores across all runs, dropping only slightly at higher thresholds. This indicates not only superior average performance but also robustness across initializations, one of the key strengths of TRPO's constrained optimization design.

IQM Performance on `MountainCarContinuous-v0` In Figure 0.9, all algorithms except TD3 reach similar performance levels, with IQMs around -0.0005 and extremely narrow confidence intervals for SAC and TQC. This suggests convergence to a stable policy that reliably solves the task. DDGP shows slightly more variance, likely due to its reliance on a deterministic policy and lack of entropy regularization. TD3, while designed to mitigate overestimation bias, underperforms here (IQM around -0.0041), possibly because this environment lacks the high variance in Q-values that TD3 is designed to correct, making its architectural complexity unnecessary.

IQM Performance on `Pendulum-v1` Figure 0.10 compares the same set of continuous control algorithms. SAC, TQC, and DDPG all cluster around an IQM of -150, with tight confidence intervals between -125 and -175, indicating consistent performance in this torque-control task. This environment is less stochastic and more amenable to stable gradient-based learning, favoring SAC's entropy-augmented updates. TD3, however, shows a stark deviation with an IQM of -400 and a wide CI extending to -675. This might

indicate either insufficient exploration or that the clipped double-Q trick, while robust in high-noise domains, may overly restrict value updates in lower-noise settings like `Pendulum-v1`, leading to conservative learning and suboptimal final policies.

Across environments, algorithms with stronger regularization or adaptive step sizes (e.g., TRPO, PPO, SAC) consistently outperform simpler or more naïve approaches like REINFORCE and ARS.

Benefits of Reliable Evaluation Metrics

Traditional metrics such as the mean or max reward per environment often obscure performance variance or are overly sensitive to outliers. In contrast, the IQM and performance profile-based evaluation allows for a more nuanced view, balancing robustness, statistical reliability, and interpretability. These plots, coupled with confidence intervals and normalized comparisons, offer a comprehensive and reproducible assessment of algorithmic performance across environments and seeds.

3.2 Evaluation Metrics for Tabular and non Tabular

Tabular algorithms permit in theory evaluation of exact norms: $\|V^\pi - V^\star\|_\infty, \|Q^\pi - Q^\star\|_\infty$ and can sample complexity until ϵ-optimality. Non-tabular methods operate in lower dimensional spaces in comparison to $\mathcal{S} \times \mathcal{A}$ and thus exact comparisons are not possible. Only tabular permits provable finite-time bounds and in non-tabular algorithms the regret curve can only be considered in its speed and stability. (At least for all metrics we have learned in the lectures.) The rliable package now gives us some methods to evaluate policies in a more complex setting:

3.3 rliable Package usefullnes for complex envirnoments evaluation

Let $S \in \mathbb{R}^{T \times K}$ store the evaluation scores of a policy, with task indexed by $i = 1, \ldots, T$ and the random-seed indexed by j= $1, \ldots, K$. rliable supplies three statistics with estimates.

Inter-quartile mean :

Flatten S into a vector of length $N := TK$ and sort it ascending, $(S_{(1)}, \ldots, S_{(N)})$ then $S_{(r)}$ denotes the r^{th} order statistic. Discard the lowest and highest quartile and then the average the middle half is given by

$$\text{IQM}(S) = \frac{2}{N} \sum_{r=\lfloor 0.25N \rfloor + 1}^{\lfloor 0.75N \rfloor} S_{(r)}$$

This 25% trimmed mean" equals the usual mean when there are no extreme values and equals the median when 50% of the data are trimmed.

Its idea is: Suppose $X_{(1)} \leq \cdots \leq X_{(n)}$ ordered statistics then for fixed $0 < p < \frac{1}{2}$ the term

$$T_{n,p} := \frac{1}{(1-2p)n} \sum_{i=\lceil pn \rceil + 1}^{\lfloor (1-p)n \rfloor} X_{(i)}$$

converges a.s. to

$$\mu_p := \frac{1}{1-2p} \int_{F^{-1}(p)}^{F^{-1}(1-p)} x \, dF(x),$$

and under certain further conditions

$$\sqrt{n}\big(T_{n,p} - \mu_p\big) \xrightarrow{d} \mathcal{N}\big(0, \sigma_p^2\big), \quad \sigma_p^2 = \frac{1}{(1-2p)^2}\big(\text{Var}[X \mathbf{1}_{[F^{-1}(p), F^{-1}(1-p)]}] + p\,(q_p)^2 + (1-p)\,(q_{1-p})^2\big),$$

where q_p and q_{1-p} are the Quantile densities. Now statistical tests are possible. Further we need fewer runs for the trimmed mean. This is due to the fact that the half width of the confidence interval $w := q_{1-\alpha/2}\,\frac{\sigma}{\sqrt{N}}$, solved for N is given by $N = \frac{q_{1-\alpha/2}^2\,\sigma^2}{w^2}$. Because the trimmed mean has a lower variance, the number of runs is smaller in order to obtain the same confidence interval.

Optimality gap:

For each task, let $S_i^\star := \max_j S_{ij}$ be the best observed score. The normalised shortfall of each run is $\delta_{ij} := (S_i^\star - S_{ij})/S_i^\star \in [0,1]$. Averaging gives

$$\mathrm{OG}(S) = \frac{1}{TK}\sum_{i,j}\delta_{ij}.$$

OG =0 is equivalent to every run matches the task best. OG =1 is equivalent to the event that every run scores zero relative to the best.

Probability of improvement :

Given two polcies with score tables S^A, S^B, define

$$\mathrm{PI}(A,B) = \frac{1}{TK}\sum_{i,j}\mathbf{1}\{S^Aij > S^Bij\}.$$

PI is the empirical probability that A beats B on the same task and seed.

3.4 Implications for Evaluation of tabular algorithms

IQM: Because random seeds alter the exploration path, it is still a benifit to calculate the trimmed, as it has a lower variance.

Optimality gap: With tabular tasks you know Q^*, so the task-wise best score S_i^* equals the optimum rather than the best observed run!

Probability of Improvement: In tabular settings Probability of imrpovement the fact that two policies might achieve identical means, but PI = 0.5 indicates they succeed on complementary seed-task pairs.