Leopold-Franzens-
Universität
Innsbruck

Department of Computer Science,
Intelligent and Interactive Systems

Bachelor Thesis

Sources of Randomness in Deep Reinforcement Learning

Elizaveta Terente

elizaveta.terente@student.uibk.ac.at

Supervised by:
Jakob Hollenstein, Dipl.-Ing.
Samuele Tosatto, Asst. Prof.

Innsbruck, 25 September 2024

Contents

List of Figures

List of Tables

Declaration

By my own signature, I declare that I produced this work as the sole author, working independently, and that I did not use any sources and aids other than those referenced in the text. All passages borrowed from external sources, verbatim or by content, are explicitly identified as such.

Signature: _____ Date: _____

1 Abstract

Deep Reinforcement Learning (DRL) is a highly advanced area in artificial intelligence where agents learn optimal behaviors through trial and error, guided by complex neural network architectures. The effectiveness of the strategies learned by the agent can be quantified through its ability to learn and adapt to its environment, which is referred to as *performance*.

A fundamental aspect of DRL training involves the agent exploring its environment through strategies introducing randomness into the decision-making process. While beneficial for exploring a wide range of possible actions and states, this randomness also means that the learning process can be unpredictable, leading to very different results even if the training conditions are the same.

The randomness is controlled by a random number generator that uses a *seed*, which is a starting value that determines the sequence of random numbers generated and thus influences the random decision-making. Explicitly setting the same seed ensures that the learning process follows an identical path, resulting in the same outcomes. This approach allows us to achieve reproducibility. Conversely, using different seeds will result in varying outcomes.

The purpose of this bachelor's thesis is to explore and define the impact of different sources of randomness on performance.

This research involved designing an experimental setup, conducting experiments, and collecting a dataset for analysis. As a result, the gathered data enabled conclusions to be drawn about the impact on the performance of each source of randomness and quantitatively compare them.

The experimentation focused on identifying the best and worst performance scenarios for each source of randomness *independently*. To achieve this, the algorithms of use (specified in section 2.2) were modified to allow *separate* seeds for each source. By fixing all sources (their seeds) and varying only the seed of the one we want to investigate, it was possible to isolate and study how that specific source influenced performance.

After identifying the best and worst performance cases, the corresponding learning curves were compared to analyze their differences. To quantitatively assess the divergence between them, the Symmetric Mean Absolute Percentage Error (SMAPE) metric, explained in detail in section 4.1, was used. A high SMAPE value (in %) indicates a significant difference between the best and worst performance curves, suggesting that variations in the seed for a particular source of randomness can greatly influence the learning process and performance, either enhancing or degrading it. In contrast, a low SMAPE value signifies minimal divergence, meaning that different seeds for the source under investigation have little impact on the learning process.

The experiments were conducted across various algorithms and environments.

Based on the collected data, it was determined that 75% of the time, the impact (quantified using SMAPE) of any source of randomness will not exceed 45%. This implies that by running the same algorithm under identical training conditions, without explicitly setting the seed, there is a 75% probability that the difference in performance will remain below 45% (SMAPE).

It was also discovered that 65% of the time, the impact (quantified using SMAPE) of the sources in the current environment is nearly equal (within a 10% SMAPE range).

The highest recorded impact reached 92% (SMAPE), while the lowest impact accounted for only 0.6% (SMAPE).

Overall, the sources can be grouped by their SMAPE-based impact, with the first group showing a slightly stronger influence than the second. However, a comprehensive analysis of the collected data and its mean values indicate that the impact of each source is relatively similar, with only minor differences.

2 Introduction

In DRL, just like in real life, we aim to find the best possible behavioral strategy within a given environment. The main character in DRL, known as the agent, has the primary goal of maximizing rewards.

The agent performs actions in its environment, leading to changes in its *state*, for example navigating through a maze. With each new state (new position of the agent in the maze), it receives information on how beneficial it is to be in that state, quantified as a *reward*. Using the collected information, it continuously updates the *policy (behavioral strategy)*, based on which it makes further decisions. This process is illustrated in Figure 1

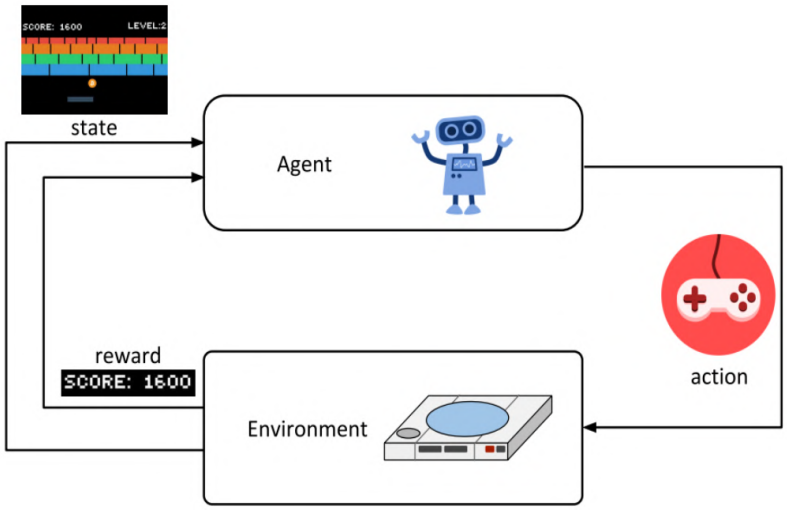


Figure 1: A simplified view of DRL [9]

The effectiveness of the agent's learned strategies and its overall success referred to as *performance*, can be quantified by the rewards it accumulates during training. This is often measured using metrics like *average cumulative rewards*, which reflect the total rewards gathered over time. Tracking these rewards provides insight into how well the agent is learning and adapting to its environment. Generally, a higher average cumulative reward indicates better performance, as the agent is making more effective decisions to reach its goals.

Ideally, to find the optimal strategy we would need to explore all possible combinations of actions and states within the environment. But realistically, that's just not feasible—it would take an enormous amount of time and computing power.

In practice, DRL works by exploring only a subset of all possibilities. This is implemented by introducing a degree of randomness, which also pushes the agent to try out different actions, even if they don't seem like the best choice at first. This approach balances exploitation (choosing the best-known action) and exploration (trying new actions to discover potentially better strategies).The randomness is typically controlled by a random number generator that initializes a **seed**, influencing the sequence of random decisions in each run. The **seed** is the starting value that determines the sequence of random numbers generated and thus influences the random decision-making. Explicitly setting the same seed ensures that the learning process follows an identical path, resulting in the same outcomes. This approach allows us to achieve reproducibility. Conversely, using different seeds will result in varying outcomes.

Randomness is essential because it allows the agent to explore a diverse range of strategies. Without it, the agent might quickly converge to a suboptimal strategy and miss out on better possibilities. However, since exploration is random, it does not guarantee consistent or stable results

across different runs. Sometimes, the exploration leads to better strategies, which can then be incorporated into the policy, while other times it may result in worse performance.

An example of this variability is illustrated in Figure 2, which shows the learning curves of two runs conducted with the *same* algorithm, the *same* input parameters, the *same* environment and the *same* number of steps, *without* explicitly setting the seed. While we might intuitively expect two runs under identical conditions to produce equal or similar results, this is not always the case. The reason lies in the underlying randomness of the process: if a seed is not explicitly set, the algorithm will randomly choose a different seed for each run, potentially leading to significantly different outcomes. As already mentioned, to achieve consistent, repeatable results, we must explicitly define the seed ensuring it does not change between executions.

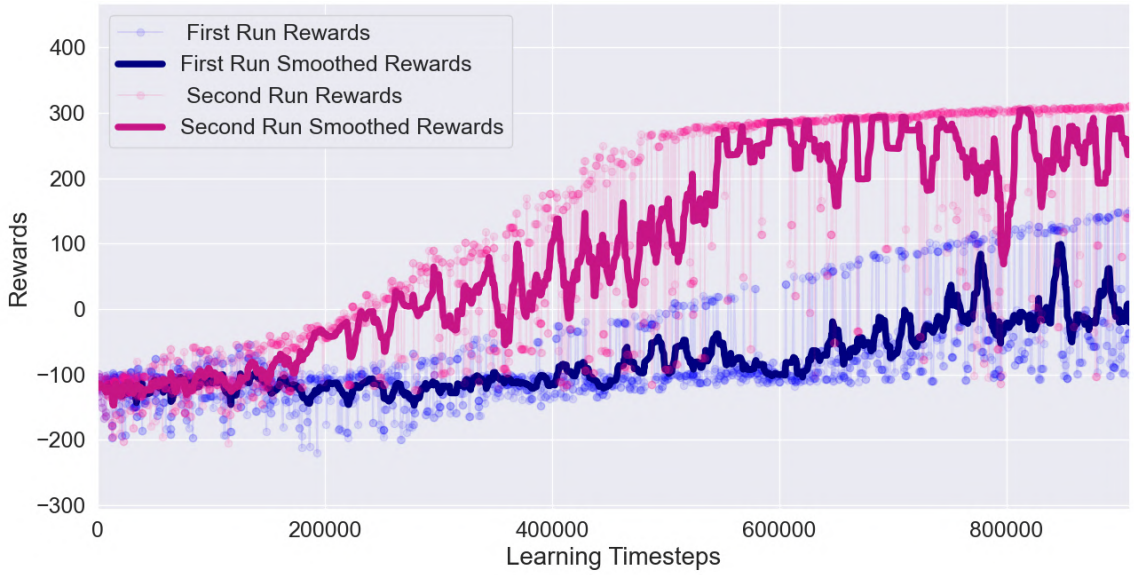


Figure 2: Learning curves from two runs conducted under identical conditions, **without** explicit seed, exhibit noticeably different performance outcomes

2.1 Actor-Critic algorithm

The actor-critic method is a popular approach used in DRL, where an agent learns to make better decisions over time by interacting with an environment. The actor-critic method gets its name from its two main components: the **actor (policy)** and the **critic (value function)**.

The **actor** is the component of the system responsible for selecting the appropriate *action* based on the current situation, or *state*, which represents all the relevant information about the environment at that specific moment. We can think of it like a person playing a game who needs to decide what move to make next. The actor uses a policy, which is a function that tells it how likely it is to choose certain actions depending on the current state. The actor's goal is to learn a strategy that will lead to the best possible results, like winning a game or getting the most points. Over time, it tries to get better at choosing actions that will maximize the reward it earns from the environment.

On the other hand, the **critic** helps the actor by evaluating how good or bad the action was. It estimates the value of the current state, which is a measure of how much future reward the agent can expect if it keeps acting in a certain way. The critic provides feedback by comparing what the agent expected to happen with what actually happened. This comparison is called the **Temporal Difference (TD) error**. The TD error tells the actor whether its action was better or worse than expected:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

In this formula, δ_t is the TD error, r_t is the reward the agent received, and $V(s)$ is the estimated value of the state. If the action led to a better result than the actor expected, the TD error will be positive, meaning the action was good. If the action led to a worse result, the TD error would be negative, meaning the action was not as good as expected.

Here is how the actor and critic work together step by step (illustrated in Figure 3):

1. The actor observes the current state (the situation it is in) and selects an action based on its policy.
2. The environment responds to the chosen action by providing a reward and transitioning to a new state.
3. The critic evaluates the outcome by comparing the reward with what was expected. It calculates the Temporal Difference (TD) error, which represents the difference between the actual reward and the critic's predicted value.
4. The critic uses this TD error to update its value estimates, improving its future predictions and feedback.
5. The actor also uses the TD error to adjust its policy, allowing it to make better decisions in future states. If the TD error is positive, the actor reinforces the action; if it is negative, the actor adjusts to avoid similar actions in the future.

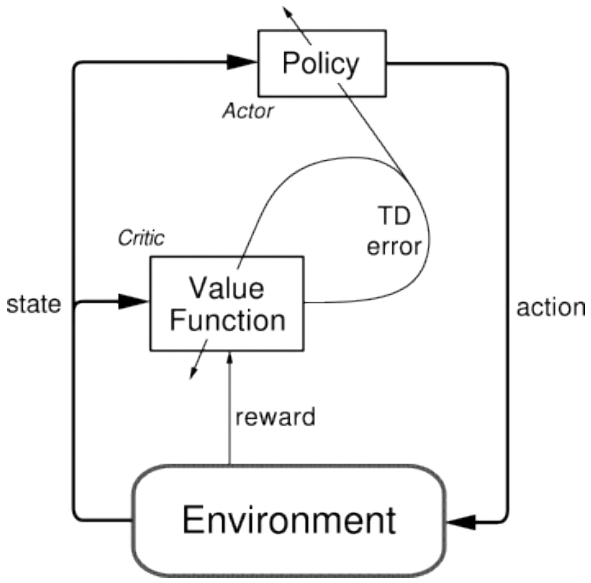


Figure 3: Actor-Critic method simplified [5]

This process continues through multiple interactions with the environment, with the actor getting better at choosing actions and the critic getting better at evaluating them. The collection of actions, state transitions, and rewards that occur from the start to the end of a task is known as an **episode**. In reinforcement learning, an episode typically begins when the agent starts in a state and ends when it reaches a terminal state or predefined stopping condition (e.g., time limit or goal achievement).

One of the main benefits of the actor-critic method is that it combines the strengths of two types of learning: **policy-based** learning (the actor's ability to choose actions directly) and **value-based** learning (the critic's evaluation of how good the current state is). The actor-critic method is especially useful in situations where the possible actions are continuous, meaning there are many possible actions to choose from, rather than just a few.

In practical implementations of the actor-critic method, neural networks are used to model both the actor and the critic. Neural networks are computational models that learn patterns from data, allowing the actor to choose actions and the critic to evaluate them. The actor's network outputs action probabilities, while the critic's network estimates how valuable the current state is for future rewards. It is common to initialize the neural networks used *randomly* for both the actor and critic components. This ensures that the agent starts with no prior knowledge, allowing it to learn from scratch through interactions with the environment.

Moreover, the use of **replay buffers** and **minibatch sampling** is also a widely adopted technique in actor-critic methods. The replay buffer stores past experiences (state, action, reward, next state tuples), allowing the agent to learn from them in a more sample-efficient manner by breaking the correlation between consecutive experiences. Instead of learning from a single interaction, the agent can sample a minibatch of experiences randomly from the buffer to estimate gradients for policy and value updates. This helps stabilize learning and improves the convergence of the actor-critic algorithm.

To sum it up, the actor-critic method allows an agent to learn by both acting (through the actor) and getting feedback on its actions (through the critic). Over time, the actor makes better choices, and the critic provides more accurate evaluations, leading to improved performance.

In this research, Actor-Critic methods will be utilized.

2.2 On-policy vs Off-policy methods

In DRL, algorithms can generally be divided into two main types: *on-policy* and *off-policy*. The key difference between these two approaches is how they handle and learn from the data generated during interactions with the environment.

On-policy algorithms learn from the data generated by the policy that the agent is currently using. This means that the agent gathers new experiences while following its **current** policy, and these experiences are then used to update the **same** policy. As a result, on-policy methods continuously adapt the policy based on the most recent interactions with the environment. However, they require new data for each update and do not utilize past experiences. This constant demand for fresh data defines on-policy algorithms, which iteratively update the policy based solely on the agent's most recent interactions with the environment. An example of an on-policy algorithm is **Proximal Policy Optimization (PPO)**, which updates the policy gradually to avoid large changes that could destabilize learning.

Off-policy algorithms, on the other hand, allow the agent to learn from data that may have been generated by a **different** policy. This means that off-policy methods can utilize experiences collected at any point in the past, regardless of the policy that was in place at the time. They store these experiences in a replay buffer, making it possible to reuse and learn from them later. Off-policy methods are more flexible in this way, as they can benefit from a diverse set of experiences, not just those from the current policy. However, the ability to learn from data generated by different policies introduces the challenge of ensuring that the learning process remains stable and accurate. **Soft Actor-Critic (SAC)** is an example of an off-policy algorithm that takes advantage of this flexibility, using an entropy term to promote wider exploration while also making good use of past experiences.

The key difference between *on-policy* and *off-policy* algorithms is that on-policy methods rely exclusively on the most recent experiences generated by the current policy, requiring continuous interaction with the environment for each policy update. In contrast, off-policy methods can learn

from a broader set of data, including past experiences collected under different policies, allowing them to store and reuse data over time.

In this research, SAC and PPO will be utilized.

2.3 Sources of randomness

The main seed serves as the starting point for generating random numbers, which are then used to introduce variability in different aspects of the DRL process, ultimately leading to different outcomes in terms of the learning process and performance. These areas where randomness typically comes into play from now on will be called **sources of randomness**.

Sources of randomness include:

- **Policy: actor weights and critic weights**

  The policy determines the agent's actions based on the current state and incorporates randomness as part of its decision-making process. In *stochastic policie*s, which are used in *SAC* and *PPO*, instead of selecting a single, fixed action, the policy outputs a probability distribution over possible actions. The agent then samples an action from this distribution, allowing it to explore a variety of actions over time. This randomness helps the agent discover and evaluate different strategies, preventing it from getting stuck in a single, potentially suboptimal, behavior pattern.

  In the actor-critic methods, both the actor and the critic have their own deep neural networks, each with its own set of weights. These networks are updated independently as the agent learns from its interactions with the environment. The actor's weights are responsible for improving the policy, while the critic's weights are responsible for refining the evaluation of actions. Together, they form an integrated system that ensures both better action selection (via the actor) and more accurate value estimation (via the critic).

  In this research, the critic and actor weights are considered together and collectively referred to as the `policy` *source of randomness.*

- **Noise in action selection**

  Noise is an additional layer of randomness that is applied after the policy has selected an action. While the policy's randomness helps in choosing diverse actions, noise further perturbs the chosen action to push the agent to explore even more. This is especially useful in continuous action spaces, where small variations can lead to different outcomes.

  From now on referred to as the `noise` *source of randomness.*

- **Environment Initialization**

  The initial state of the environment is randomized at the start of each episode. This prevents the agent from overfitting to specific scenarios.

  From now on referred to as the `environment` *or* `env` *source of randomness.*

- **Experience Replay Buffer**

  During training, experiences are stored in a replay buffer, and a random sampling of these experiences is used to update the policy and value networks. The randomness in sampling ensures that the agent learns from a diverse set of past experiences, rather than being biased towards the most recent events.

  From now on referred to as the `buffer` *source of randomness.*

The aforementioned sources of randomness will be further investigated in this research. Their individual impact will be analyzed, and a comparison will be made to determine which sources have a greater influence and which have a smaller impact on performance.

2.4 Problem Statement

Now, with all the given information, we can move on to the central point of this research: even under identical training conditions without explicit seed setting, running the same DRL algorithm multiple times will produce different outcomes due to randomness. The differences can vary, sometimes being significant and other times minor. The purpose of this research is to examine the impact of randomness on performance by investigating several sources of randomness - `environment`, `buffer`, `noise`, and `policy`. While it is clear that randomness affects the learning process we still do not know much about it.

The questions this research aims to answer are:

1. How much do sources of randomness affect agent performance?
2. Which sources of randomness exert a greater impact, and which are less influential?

3 Methodology

This section will provide a detailed overview of the entire experimental process. The code used for the experiments can be accessed at GitLab [11].

The algorithms of interest are *Actor-Critic* methods, specifically Soft *Actor-Critic (SAC)* and *Proximal Policy Optimization (PPO)*. This research will utilize Stable Baselines 3 [10] implementations of these algorithms, focusing on continuous action domain environments provided by Gymnasium [2] and MuJoCo [6].

Table 1 presents the specific environments used, along with their corresponding *state* and *action dimensions*.

State dimensions refer to the number of variables or features that describe the current condition of the environment, known as the state. These variables provide essential information that the agent needs to make decisions. For example, in the *Pendulum-v1* environment, the state consists of 3 dimensions: the cosine of the pendulum's angle, the sine of the pendulum's angle, and its angular velocity. These three variables together describe the current position and movement of the pendulum. In the *CarRacing-v2* environment, the state is much more complex, with 27648 dimensions representing the pixels of an image (96 x 96 with 3 color channels) that shows the track the car is driving on.

Action dimensions refer to the number of different actions the agent can perform at each time step. In environments with continuous action spaces, these actions are represented by variables the agent controls. For example, in the *Pendulum-v1* environment, the agent controls the torque applied to the pendulum (1 action dimension). In the *Ant-v4* environment, the agent controls the angles of 8 different joints, resulting in 8 action dimensions. Each action dimension gives the agent a degree of control over its movements or decisions in the environment.

Environment	**State Dimensions**	**Action Dimensions**
Pendulum-v1	3	1
MountainCarContinuous-v0	2	1
CarRacing-v2	27648	3
LunarLanderContinuous-v2	8	2
BipedalWalker-v3	24	4
BipedalWalkerHardcore-v3	24	4
Ant-v4	111	8
HalfCheetah-v4	17	6
Hopper-v4	11	3
Humanoid-v4	376	17
Reacher-v4	11	2
Walker2d-v4	17	6
HumanoidStandup-v4	376	17
InvertedPendulum-v4	4	1
InvertedDoublePendulum-v4	11	1
Pusher-v4	23	7

Table 1: State and action dimensions of environments used in the research

3.1 Abstract of the Central Concept

The key idea is to extend the algorithms of interest from Stable Baselines 3 [10] to enable the setting of *separate* seeds for each source of randomness. Since using the same seed consistently produces identical results—because the same seed leads to the same outcomes— by assigning distinct seeds to each source of randomness, it would be possible to isolate and assess the impact of each source independently.

To be able to do so, a series of experiments would be conducted where all sources of randomness (their seeds) are kept constant, except for the one under investigation.

For example, if we aim to investigate the impact range of the environment's randomness, we would train several models where all other seeds are fixed, varying only the environment seed. This approach would allow us to observe how changes in the environment's seed affect performance.

Experimental input:

- source of randomness = environment;
- buffer seed = 5;
- noise seed = 6;
- policy seed = 7;
- range of values to test for the environment seed = [1..4];

Experimental output:

name	buffer_seed	noise_seed	policy_seed	env_seed	performance
model1	5	6	7	**1**	***0.85***
model2	5	6	7	**2**	***0.83***
model3	5	6	7	**3**	***0.87***
model4	5	6	7	**4**	***0.82***

This way impact of the source (environment) can be quantified by analyzing the variations in performance.

To generalize from this specific example: numerous experiments will be conducted using *SAC* and *PPO* algorithms across all the mentioned environments for every source of randomness. This experimentation will result in an extensive dataset, allowing for broad conclusions to be drawn regarding each source of randomness.

3.2 Extension of SAC and PPO to be able to set separate seeds

The implementation of these algorithms is initially based on Stable Baselines 3 [10]. The task then was to develop functionality that would allow **separate** seeds for the four sources of randomness being investigated— `environment`, `buffer`, `noise` and `policy` —to be passed as input parameters. The goal was to create two scripts, one for *SAC* and one for *PPO*, that incorporated this functionality.

Gymnasium [2] already provided a function for setting the seed in the `environment`, so no further implementation was needed.

However, for the `buffer`, `noise`, and `policy`, it was necessary to create custom classes, inheriting from those used in Stable Baselines 3 [10] or other relevant implementations [7], [3].

For the replay buffer in SAC and rollout buffer in PPO, custom classes were created inheriting from those used in the original Stable Baselines 3 [10] implementation. These custom classes take seed as an input parameter and initialize the Random Number Generator with it, which is then used to sample batch indices from the replay buffer.

For both SAC and PPO, recent research has demonstrated the benefits of using colored noise over traditional uncorrelated white noise. In SAC, *pink noise* has shown superior performance [1], and an implementation [7] was utilized that allows for the input of an RNG seed.

Similarly, in PPO, a variant of colored noise between white and pink has proven to enhance exploration and outperform white noise [4]. Implementation [3] already provides functionality to seed noise generator.

For the policy in SAC, a custom class was created based on the one utilized in Stable Baselines 3 [10], adding functionality to initialize the distribution according to the seed provided as an input parameter. For the PPO policy, the implementation referenced earlier in the noise paragraph was used [3].

3.3 Oracle

With our algorithms now in script form, we can move on to training multiple models and comparing their performance—a task managed by the script called *Oracle*.

The *Oracle* receives input specifying the algorithm we want to use and the source of randomness we wish to investigate, including the range of seed values (size n) to try for the source of interest, along with fixed seeds for other sources. It then selects the appropriate script corresponding to the algorithm and trains n identical models in parallel, differing only in the seed value for the source of interest. In the end, we obtain the models and performance results, which allow us to determine the best-performing seed value under current parameters.

This process is described in Algorithm 1 and illustrated in Figure 4.

Algorithm 1 Oracle

Input:

- Algorithm $a \in \{\text{SAC}, \text{PPO}\}$
- Environment name $envName$
- Number of learning timesteps $learningTimesteps$
- Source of randomness $sourceToInvestigate \in \{\text{env}, \text{buffer}, \text{noise}, \text{policy}\}$
- Range of seed values $[n, m]$ to try for $sourceToInvestigate$
- Fixed seeds values S_{fixed} for other sources of randomness

Output: Trained models $\mathcal{M}_n ... \mathcal{M}_m$ (along with the corresponding buffer and noise distribution) and their performance results $p_n ... p_m$ saved in CSV files.

1: **for** each i in $[n, m]$ **do**
2: Train model $\mathcal{M}_i$ using a ,$envName$, $learningTimesteps$, $sourceToInvestigate$, S_{fixed}, i
3: Save the model $\mathcal{M}_i$, corresponding buffer and noise distribution
4: Evaluate the model $\mathcal{M}_i$
5: Write performance result p_i to **backlog_evaluation.csv**
6: **end for**
7: based on **backlog_evaluation.csv** choose best-performed model and write corresponding information to **result.csv**
8: clean up other models

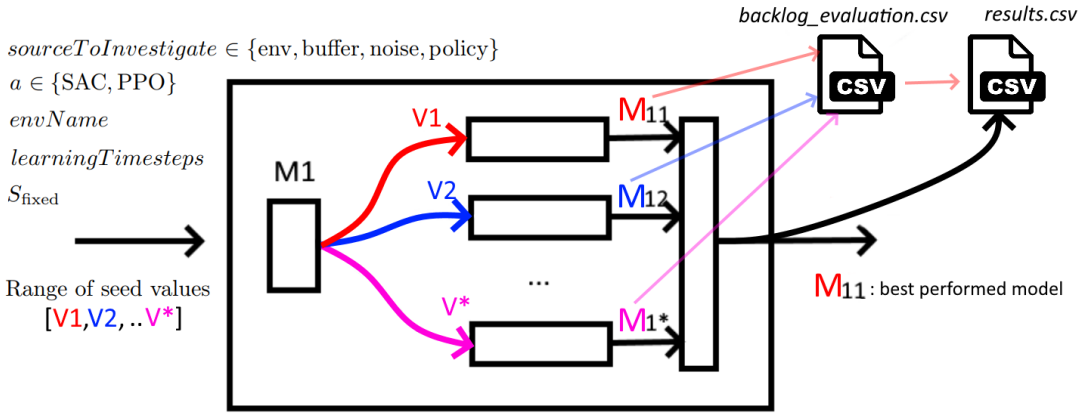


Figure 4: Oracle simplified

For example, resulting CVS files for :

- a=SAC
- $envName$=MountainCarContinuous-v0
- $learningTimesteps$ = 15000
- $sourceToInvestigate$=buffer
- $[n, m]$ = [1,6]
- S_{fixed} = 111(env), 222(noise), 333(policy)

would be:

1. **backlog_evlaustion.csv (Table 2) :**

model_name	average_reward
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=1	90.28210528571428
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=2	88.08339229411766
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=4	89.15047222950818
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=6	89.05499977777778
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=5	91.11692455
SAC_15000_MountainCarContinuous-v0_111_222_333_buffer=3	91.195188525

Table 2: backlog_evaluation.csv

2. **results.csv (Tables 3 , 4 , 5):**

anti*	algorithm	section*	learning_timesteps	environment
False	SAC	0	15000	MountainCarContinuous-v0

Table 3: Part 1 of results.csv

* *gray out areas do not make sense for now and will be explained later in Section 3.5.*

env_seed	policy_seed	noise_seed	buffer_seed	source	seed_value
111	222	333	None	buffer	3

Table 4: Part 2 of results.csv

model_name	avg_reward
SAC_0_15000_MountainCarContinuous-v0_111_222_333_None_buffer=30	91.195188525

Table 5: Part 3 of results.csv

3.4 Impact of randomness in different stages of learning

To accurately investigate the impact of sources not just once during the training process but throughout different phases, a more detailed analysis was undertaken. The following approach allows us to examine the effects at the beginning, middle, and end of the training process, as results may vary across these stages. To make this possible, a simple input parameter *sectionsNumber* was introduced, which divides the learning process into several sections. Although this parameter is configurable, in this thesis it will be used with its default value of **3**, corresponding to the beginning, middle, and end of the training.

A common example of the learning curve is shown in Figure 5.

In the first section (beginning of the training), we typically observe rapid growth. In the second section (middle of the training), the growth rate slows down, and by the third section (end of the training), we either see further deceleration or stabilization.

These are distinct phases of the learning process that were specifically studied.

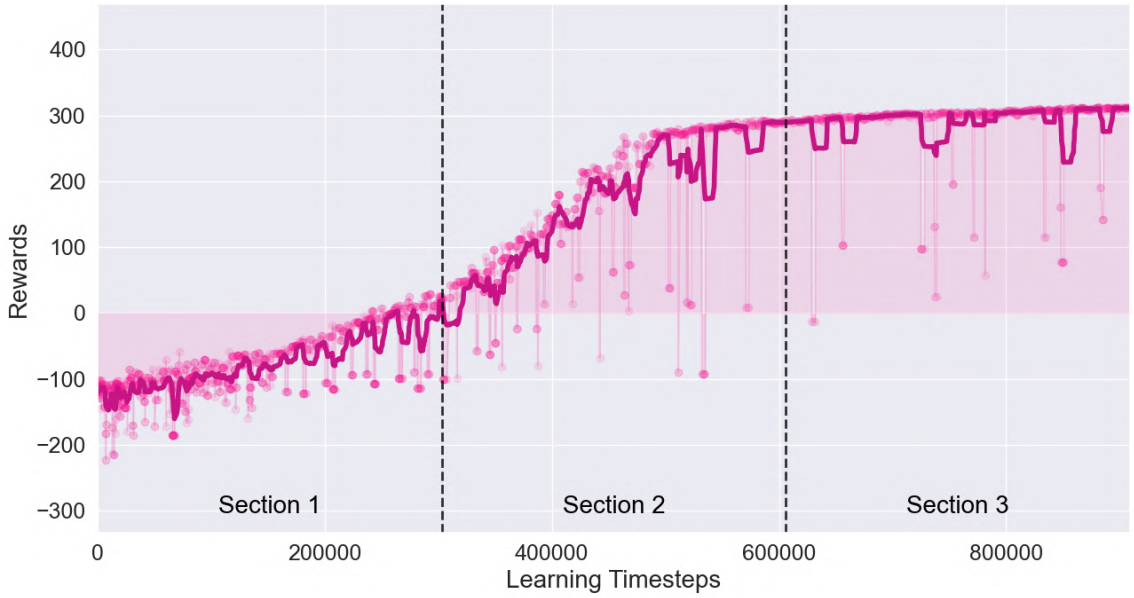


Figure 5: Learning curve divided into 3 sections

3.5 SuperOracle and AntiSuperOracle

To implement the handling of several sections, *SuperOracle* is introduced.

SuperOracle consists of sequential *Oracle* runs (one *Oracle* per section). It operates similarly to the previously described **Oracle**, where we receive input parameters such as algorithm a, $envName$, $learningTimesteps$, $sourceToInvestigate$, the seed values range $[n, m]$, and S_{fixed}. However, in addition, *SuperOracle* also takes the parameter $sectionsNumber$. The division into sections works by taking the total number of steps $learningTimesteps$, dividing it by the number of sections $sectionsNumber$, and then running subsequent *Oracle*s $sectionsNumber$ times, each with $\frac{totalTimesteps}{sectionsNumber}$ steps.

In essence, to estimate the impact of randomness, we only need the best-performing model and the worst-performing model, which would represent the upper and lower bounds of possible performance.

Therefore, **SuperOracle** has an additional parameter, *anti*. When this parameter is set to False, it selects the best model, and when set to True, it focuses on the worst model.

Therefore, **SuperOracle** has an additional parameter, *anti*. When this parameter is set to False, it tells **Oracle** to select the *best* model, and when set to True - the *worst* model. In the earlier **Oracle** section 3.3, we only considered the best-performing model for simplicity.

To obtain the lower and upper bounds on performance, we would need to run *SuperOracle* twice with the same set of parameters: once with *anti* set to True and once with *anti* set to False, as illustrated in Figure 6. From now on, for simplicity, *SuperOracle* with *anti* set to True will be referred to as *AntiSuperOracle*.

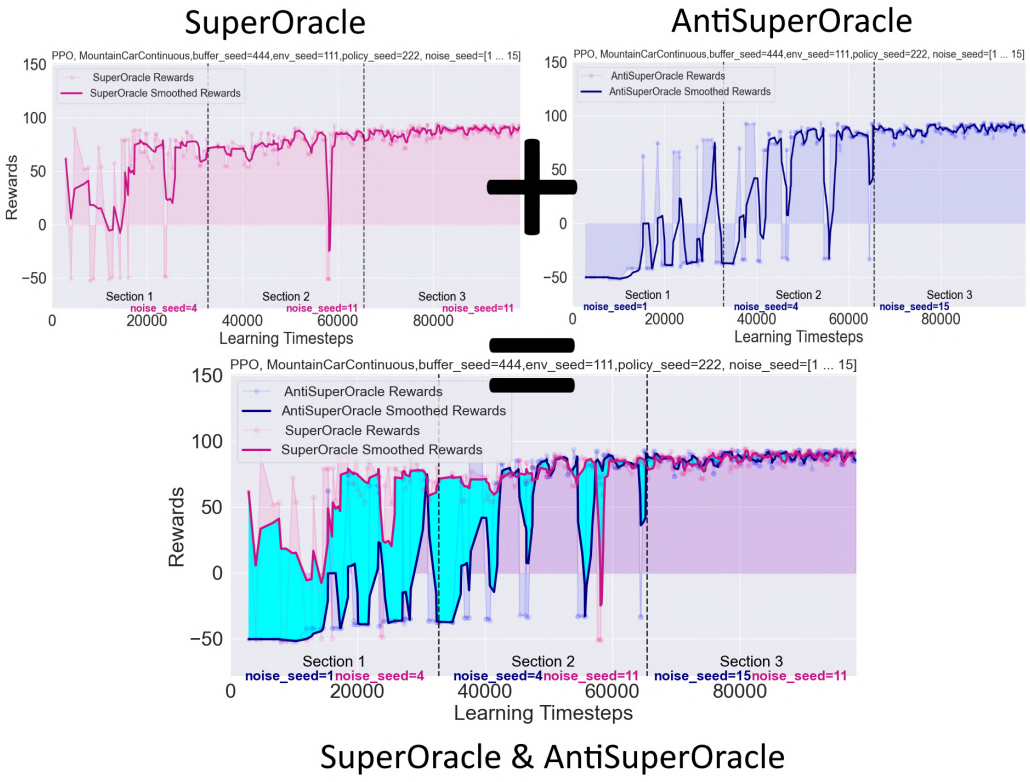


Figure 6: The resulting graph shows the upper bound (SuperOracle) and lower bound (AntiSuperOracle) of performance together to highlight the difference

The process of (*Anti*)*SuperOracle* works as follows: after each sequential *Oracle* run, it selects the best (or worst) model and passes it as the starting model for the next **Oracle**. This approach allows us to retain the learned experience and continue training from where the previous section ended.

This process is described in Algorithm 2 and illustrated in Figure 7.

Algorithm 2 (Anti)SuperOracle

Input:

- $anti \in \{\text{True}, \text{False}\}$
- number of sections $sectionsNumber$
- Algorithm $a \in \{\text{SAC}, \text{PPO}\}$
- number of learning timesteps $learningTimesteps$
- environment name $envName$
- Source of randomness $sourceToInvestigate \in \{\text{env}, \text{buffer}, \text{noise}, \text{policy}\}$
- Range of seed values $[n, m]$ for $sourceToInvestigate$ to try
- Fixed seeds values S_{fixed} for other sources of randomness

Output: The best (or worst) performing model (along with the corresponding buffer and noise distribution), a **backlog_evaluation.csv** file containing all intermediate performance results, and a **results.csv** file with the final results, including the best (or worst) models and corresponding information for each section.

$learningTimestepsPerSection \leftarrow \frac{learningTimesteps}{sectionsNumber}$
if *anti* is True **then**
  Choose the worst-performed model.
else
  Choose the best-performed model.
end if
for each s in $[1..sectionsNumebr]$ **do**
  if $s == 1$ **then**
    $model \leftarrow oracle(anti, learningTimestepsPerOracle, a, envName, sourceToInvestigate, [n, m], S_{\text{fixed}})$
  else
    (for the further sections respectively *Oracle* runs model received from the previous *Oracle* used)
    $model \leftarrow oracle(anti, learningTimestepsPerOracle, \textbf{model}, a, envName, sourceToInvestigate, [n, m], S_{\text{fixed}})$
  end if
  clean all the models except best/worst one.
end for

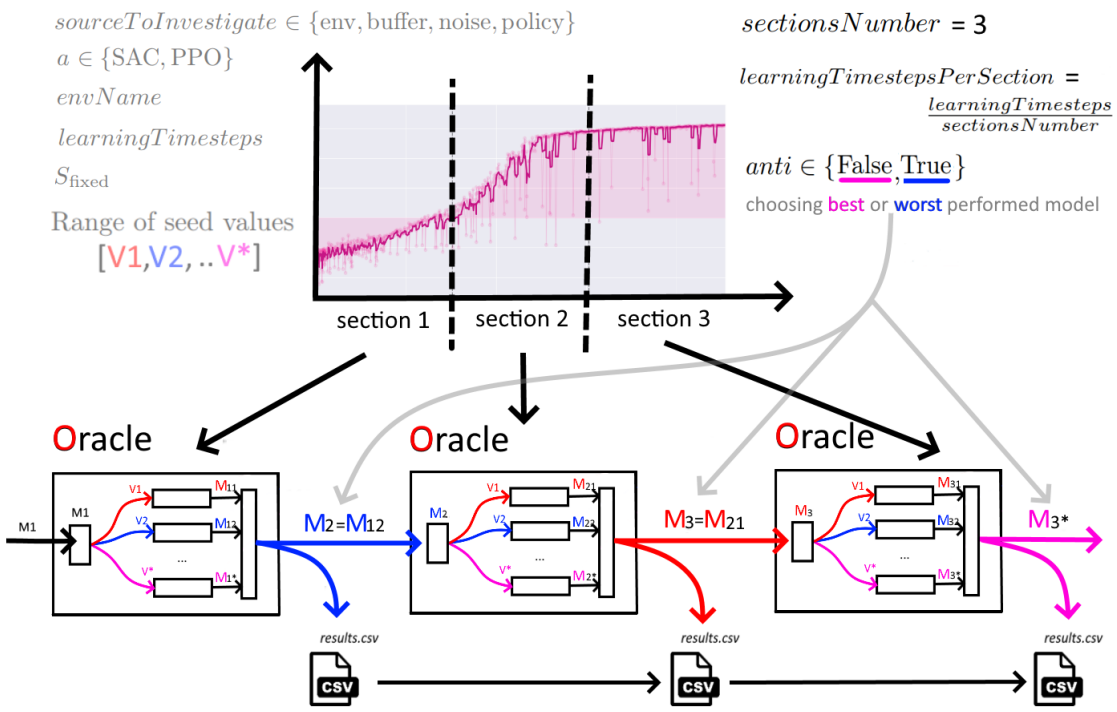


Figure 7: SuperOracle simplified

For example, **results.cvs** for :

- *anti*=False
- *a*=PPO
- *sectionsNumber*=3
- *envName*=MountainCarContinuous-v0
- *learningTimesteps* = 75000
- *sourceToInvestigate*=buffer
- $[n, m]$ = [1,30]
- S_{fixed} = 3(env), 2(noise), 1(policy)

would be:

anti	algorithm	section	learning_timesteps	environment
False	PPO	1	25000	MountainCarContinuous-v0
False	PPO	2	25000	MountainCarContinuous-v0
False	PPO	3	25000	MountainCarContinuous-v0

Table 6: Part 1 of results.csv

env_seed	policy_seed	noise_seed	buffer_seed	source	seed_value
3	1	2	None	buffer	30
3	1	2	None	buffer	14
3	1	2	None	buffer	13

Table 7: Part 2 of results.csv

model_name	avg_reward
PPO_1_25000_MountainCarContinuous-v0_3_1_2_None_buffer=30	42.4579
PPO_2_25000_MountainCarContinuous-v0_3_1_2_None_buffer=14	66.4768
PPO_3_25000_MountainCarContinuous-v0_3_1_2_None_buffer=13	77.6513

Table 8: Part 3 of results.csv

3.6 Resetting between sections

Starting with the second section (or the second **Oracle** run), we do not create a new model but instead reuse the one from the previous **Oracle** step, which we receive as an input parameter. Along with the model, we also load the corresponding buffer and noise distribution. Section 3.2 describes how we set seeds for a **new** model. Now, we need to **reset** the seeds for an **already existing** model, which is a slightly different process.

We need to reset all sources of randomness, setting three of them to the same value, and update the source of interest with a new value. To achieve this, additional reset functions were implemented for the `buffer`, `policy`, and `noise` distributions. Fortunately, Gym has a built-in feature for this, so no additional work on `environment` source was needed.

For the `buffers` and `noise` distributions in SAC and PPO, a function was created to reinitialize the Random Number Generator with the provided seed.

While resetting the `buffer`, `noise`, and `environment` is straightforward and does not significantly disrupt the learning process, resetting the `policy` - does, therefore requires more caution.

3.6.1 Policy resetting

Resetting the policy understandably causes a significant drop in performance because the policy serves as the central decision-making component of the model. The difference between SAC, an *off-policy* algorithm, and PPO, an *on-policy* algorithm, as mentioned in Section 2.2, also influences how these algorithms handle policy resets.

In SAC, which is an *off-policy* algorithm, periodically resetting the policy can improve performance. Research [8] has shown that deep DRL algorithms often suffer from "primacy bias", where the model overfits to early experiences. Experimental results, as illustrated in Figure 8, suggest that resets act as a form of regularization, preventing overfitting and improving overall performance and rewards across various environments and algorithms.

I attempted to replicate the results from the research [8], as shown in Figure 9. The pink curve represents the learning process **with** policy resetting, and the blue one - **without**. However, I was unable to achieve noticeable performance improvements. In my case, resetting the policy initially causes a drop, with rare cases of temporary performance rising afterward, but eventually, it returns to its original trajectory. As a result, both *with* and *without* the policy reset, the algorithm ultimately reached similar results. Although I did not observe performance improvements from resetting the policy, I did notice that it does not disrupt the learning process. Therefore, it was decided to implement policy resets in SAC.

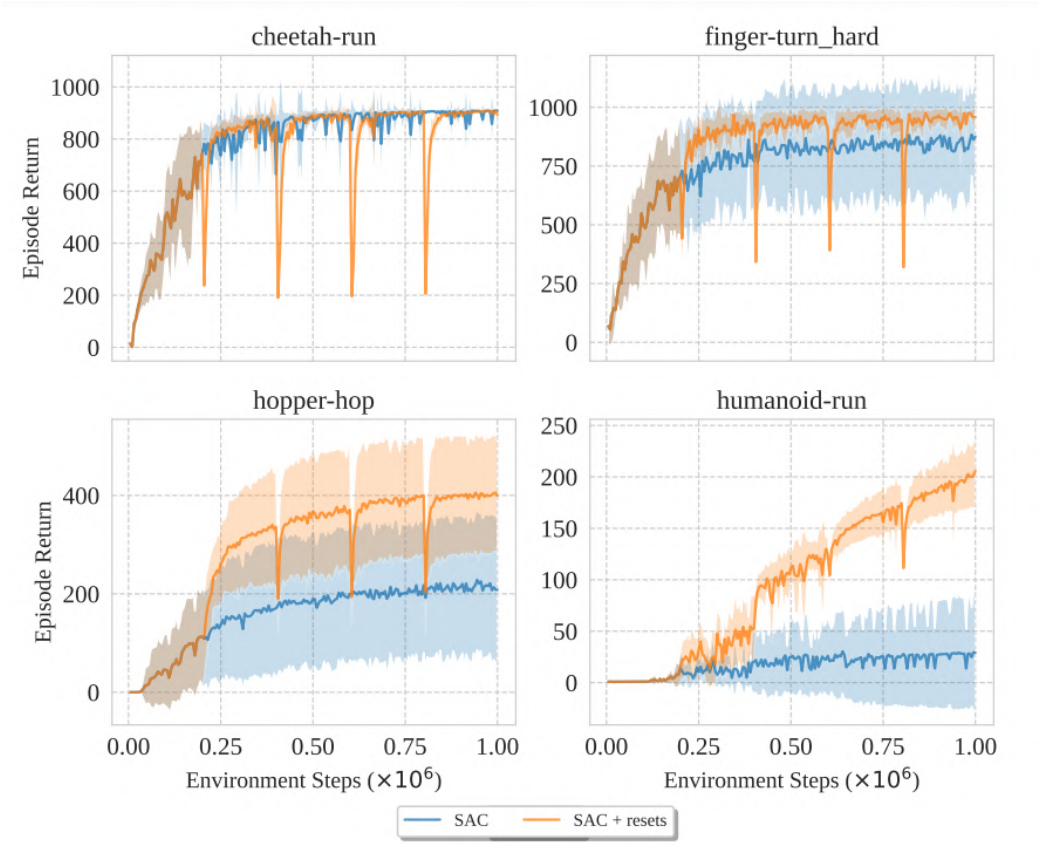
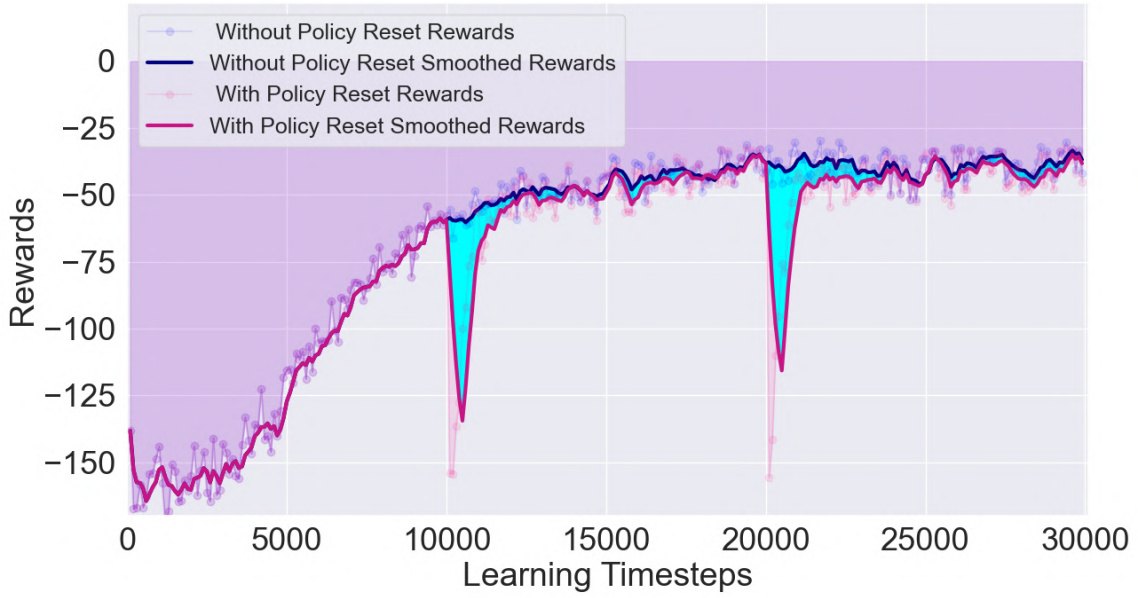
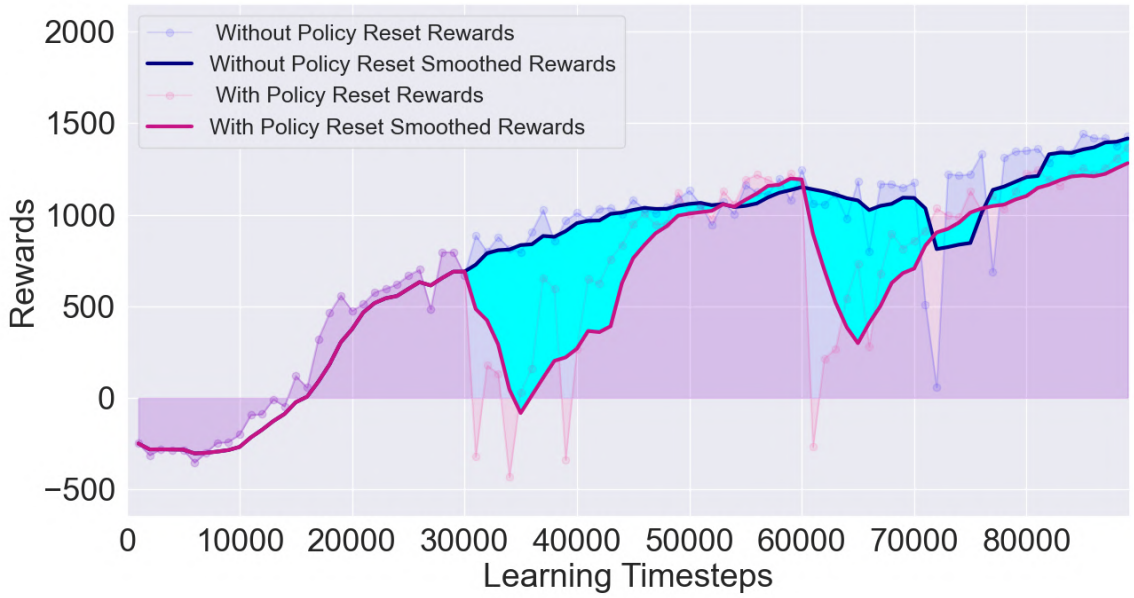


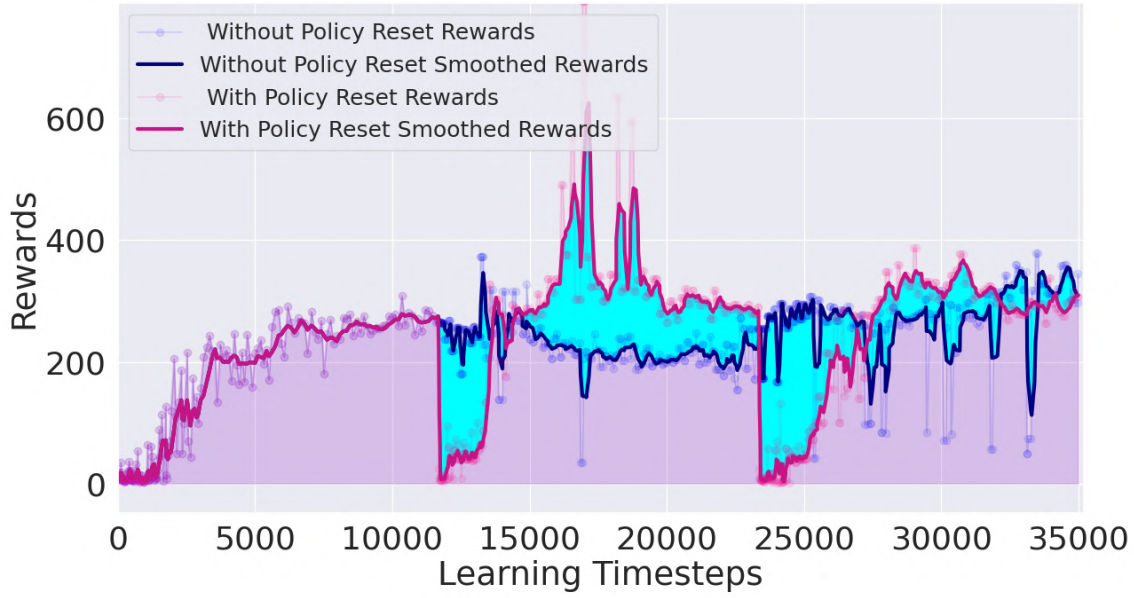
Figure 8: Policy reset results taken from the paper "The Primacy Bias in Deep Reinforcement Learning" [8]



(a) My policy reset results for SAC - Pusher



(b) My policy reset results for SAC - HalfCheetah



(c) My policy reset results for SAC - Hopper

Figure 9: Comparison of policy reset results for SAC across different environments

In PPO, however, the situation is different. Given that PPO relies heavily on recent experiences to update the policy, resetting it might not align with the algorithm's natural learning process. As a result, it was concluded that policy resets would **not** be implemented in PPO.

This means that the policy in PPO is the only source of randomness that will be investigated just once during the training process, corresponding to a single section (secionsNumber=1).

3.7 Deciding on seed range size

Next, it was necessary to determine the appropriate width for the seed range to try. The goal was to find a balance between a range broad enough to provide adequate information about performance differences, while also remaining computationally feasible.

The range was determined using four different methods, with the median value being considered for the final result. For each algorithm and each source of randomness, a range [1 .. 100] was tested.

As a preparatory step, experiments were conducted to collect the dataset. **Oracle** (explained in Section 3.3) was executed with a seed range from 1 to 100 for every source of randomness. These experiments produced **backlog_evaluation.csv** files, containing information on average rewards and corresponding model names. The average rewards were extracted from these files, and this data is referred to as **Dataset** D in the following algorithms 3 and 5.

3.7.1 Performance Range Method (Algorithm 3)

The first method, from now on referred to as the **Performance Range Method**, is a custom approach that focuses on the variation in model performance captured by different numbers of seeds. For each number of seeds n from 1 to 100, the difference between the maximum and minimum average rewards sampled from the dataset is calculated. By iterating this process multiple times and averaging the results, a measure of the captured reward difference for each n is obtained.

Stabilization Criterion: The number of seeds n at which the variation between consecutively captured differences becomes smaller than a defined threshold is considered the optimal seed count.

Algorithm 3 Performance range method

Input: Dataset with average rewards D, Maximum seeds $N_{\max} = 100$, Stabilization threshold ϵ=0.1, Maximum iterations $I_{\max} = 100$
for $n = 1$ to $N_{\max}$ **do**
  $differences = []$
  $averageDifferences = []$
  for $i = 1$ to $I_{\max}$ **do**
    Sample n rewards from D
    Compute the difference between max and min rewards
    Append difference to $differences$
  end for
  Compute average difference for n based on $differences$
  Append average difference to $averageDifferences$
end for
Find n where the change in difference $\Delta < \epsilon$ based on $averageDifferences$
Return: n

Figure 10 illustrates how the captured average reward difference increases as the number of seeds grows. The x-axis represents the number of seeds, while the y-axis shows the reward difference. Initially, the reward difference rises sharply, but it eventually stabilizes, as marked by the red dashed line (in this example around 17 seeds). After this point, adding more seeds results in only minor changes to the reward difference, indicating that increasing the seeds does not capture much more variation.

3.7.2 Statistical Bootstrapping

Statistical Bootstrapping is a technique similar to the **Performance Range Method**, but instead of sampling without replacement, it utilizes bootstrapping, where sampling is done with replacement. This method aims to provide a more robust estimate of the reward difference by incorporating the inherent variability in the dataset.

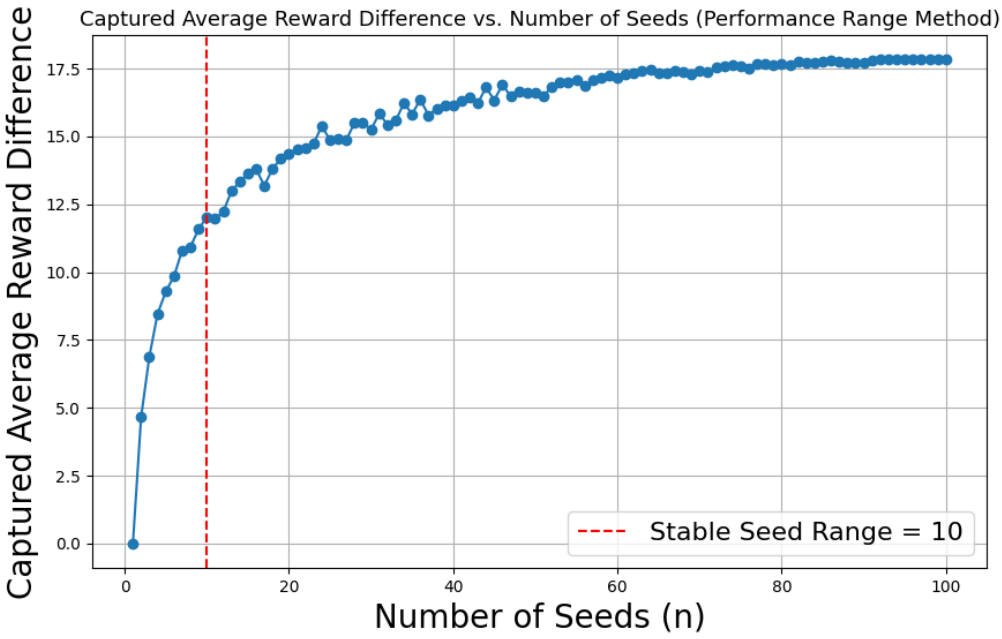


Figure 10: Performance range method example for `env` source using SAC, MountainCarContinuous-v0

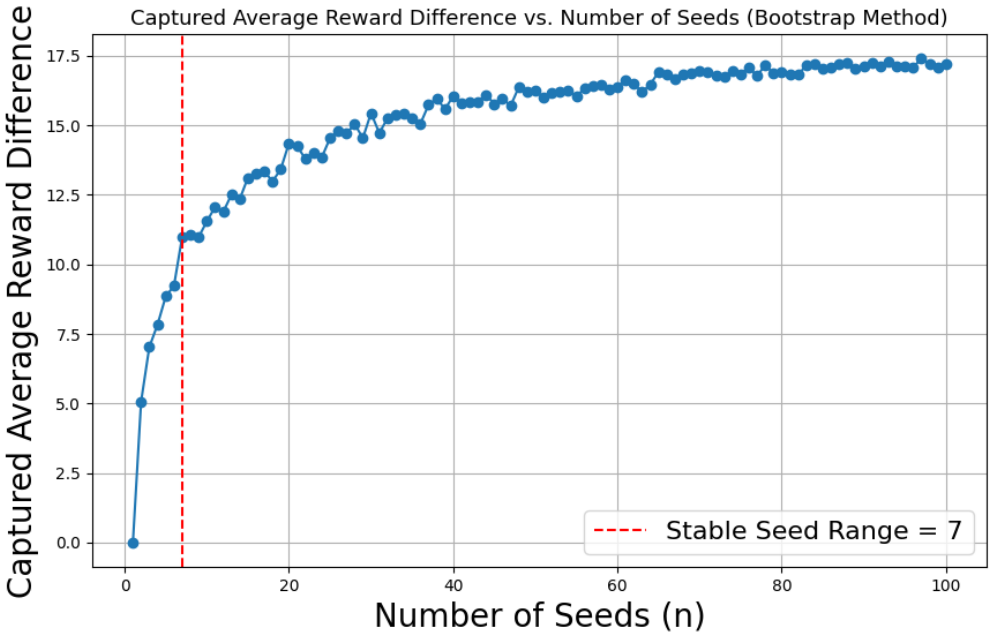


Figure 11: Statistical bootstrapping example for `env` source using SAC, MountainCarContinuous-v0

3.7.3 Moving Average Method (Algorithm 4)

This method applies a **Moving Average** over the captured differences obtained from the **Performance Range Method**. The goal is to smooth out fluctuations and identify a stabilization point more effectively.

Stabilization Criterion: The number of seeds n is identified where the change in the moving average of the captured differences falls below the defined threshold.

Algorithm 4 Moving average method

Input: List of captured average differences $averageDifferences$ from **Performance Range Method**, Window size w=5, Stabilization threshold ϵ=0.1
Compute moving average list $movingAverages$ for $averageDifferences$ with window size w
for $i = 2$ to length of $movingAverages$ **do**
  Compute the change Δ between consecutive moving average values
  if $\Delta < \epsilon$ **then**
    Return: $n = i + w$ as the optimal seed count
  end if
end for
Return: n

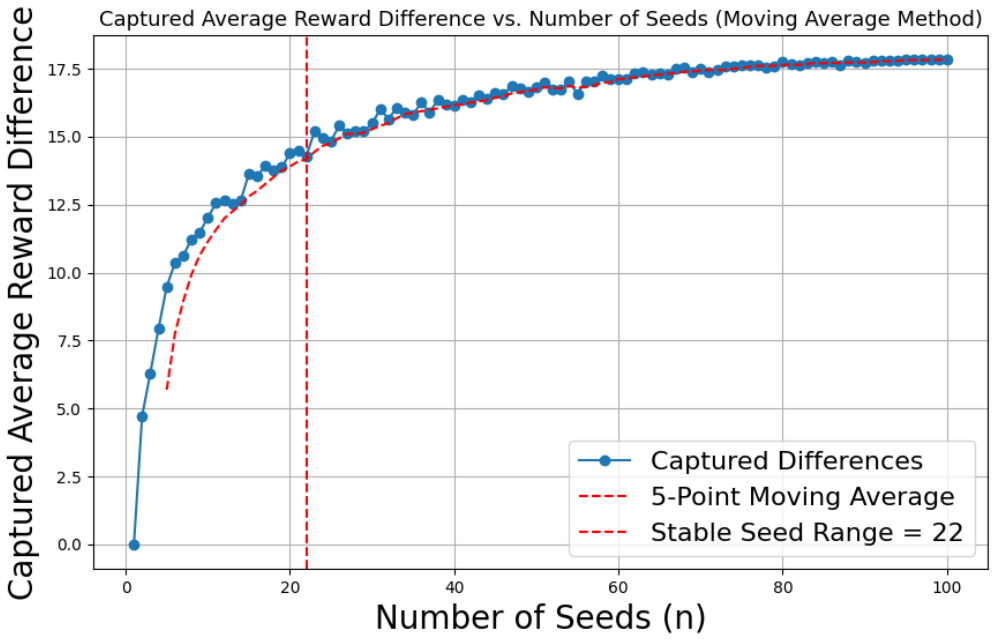


Figure 12: Moving average method example for `env` source using SAC, MountainCarContinuous-v0

3.7.4 Cumulative Mean and Standard Deviation Method (Algorithm5)

In this method, the **Cumulative Mean** and **Standard Deviation** of the rewards are calculated as the number of seeds increases. By analyzing these cumulative statistics, it is possible to determine when additional seeds contribute negligibly to the variance, indicating that most variability is already captured.

Algorithm 5 Cumulative mean and standard deviation method

Input: Dataset with average rewards D, Maximum seeds number $N_{\max}$=100, Stabilization threshold ϵ=0.1
for $n = 1$ to $N_{\max}$ **do**
    Sample n rewards from D
    Compute cumulative mean and std deviation
end for
Find n where the change in cumulative mean or std deviation $\Delta < \epsilon$
Return: n

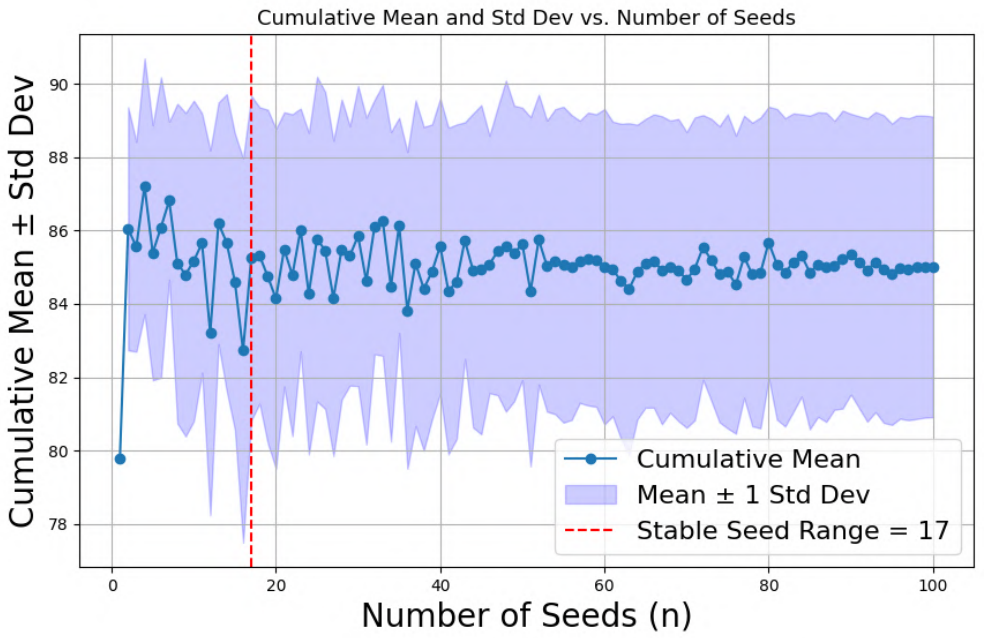


Figure 13: Cumulative mean and standard deviation method example for `env` source using SAC, MountainCarContinuous-v0

3.7.5 Final results

Each method provides a different approach to determining the optimal number of seeds. The *Performance Range* and *Statistical Bootstrapping* methods measure performance variability, while the *Moving Average* and *Cumulative* methods offer smoothed and overall views. By combining these methods, a clearer understanding of seed impact on model performance is achieved. The median of these four methods was taken as the final result.

Experiments were conducted using the two fastest environments: *MountainCarContinuous-v0* and *Pusher-v4*, and repeated across multiple runs to obtain stabilized values. The results are presented in Table 9.

	environment name	source	Performance Range	Statistical Bootstrapping	Moving Average	Cumulative Mean & Standard Deviation	Final (median)
SAC	**MountainCarContinuous-v0**	**env**	11	11	23	17	15
		buffer	8	7	17	11	10
		noise	12	12	26	15	16
		policy	7	8	15	8	9
	Pusher-v4	**env**	7	7	15	7	9
		buffer	7	7	12	11	9
		noise	7	15	7	10	9
		policy	8	14	8	12	9
PPO	**MountainCarContinuous-v0**	**env**	15	15	28	14	23
		buffer	10	10	23	10	12
		noise	11	11	24	14	15
		policy	9	9	22	12	12
	Pusher-v4	**env**	5	9	5	4	5
		buffer	3	6	3	2	3
		noise	7	13	7	6	7
		policy	2	6	2	2	2

Table 9: Optimal seed range calculated with different methods

Across these two environments, the highest values were selected, and the final decision on the seed range is presented in Table 10.

	env	**buffer**	**noise**	**policy**
SAC	[1 ... **15**]	[1 .. **10**]	[1 ... **16**]	[1 ... **9**]
PPO	[1 ... **23**]	[1 ... **12**]	[1 ... **15**]	[1 ... **12**]

Table 10: Final seed range decision

The corresponding code can be accessed at GitLab [11].

3.8 Determining the optimal number of steps in each environment

To determine the optimal number of steps for each environment, the training was executed for a large number of steps to observe the learning curve. Since 3 sections are being studied, the number of steps was chosen so that the first section represents rapid growth, the second section shows slower

growth, and the third section reflects stabilization as was discussed in 3.4. While this pattern does not always occur, it is fairly common.

The results are represented in Table 11.

env	**The optimal number of steps**	
	SAC	**PPO**
Pendulum-v1	15 000	300 000
MountainCarContinuous-v0	15 000	75 000
BipedalWalker-v3	25 000	900 000
LunarLanderContinuous-v2	125 000	300 000
BipedalWalkerHardcore-v3	40 000	1 000 000
Ant-v4	30 000	900 000
HalfCheetah-v4	90 000	900 000
Hopper-v4	35 000	900 000
Reacher-v4	30 000	600 000
InvertedPendulum-v4	60 000	300 000
InvertedDoublePendulum-v4	60 000	600 000
Pusher-v4	30 000	800 000
Walker2d-v4	60 000	500 000
CarRacing-v2	-*	150 000
HumanoidStandup-v4	60 000	800 000
Humanoid-v4	60 000	900 000

Table 11: The optimal number of steps in different environments

It was not possible to run experiments with **SAC for the **CarRacing-v2** environment due to memory allocation issues. SAC, being an **off-policy** algorithm, relies on a replay buffer to store large amounts of experience data (was explained in Section2.2), including high-dimensional image states from CarRacing-v2 (96x96x3) (mentioned in Paragraph 3). This leads to significant memory requirements, and the attempt to store one million frames resulted in a memory allocation error. In contrast, **PPO**, an **on-policy** algorithm was able to run without encountering this issue.*

4 Experimental Dataset Overview

This section presents the results of the *SuperOracle* and *AntiSuperOracle* using a few selected examples to illustrate different aspects of the data. These examples have been chosen to showcase key observations, as the full dataset is very large. Detailed results are available in the Appendix A, B and C.

The methods of data representation :

1. **SuperOracle & AntiSuperOracle graph**

   For every source of randomness, graph will display two curves: the pink one represents the *SuperOracle*, and the blue one represents the *AntiSuperOracle*. The areas where they differ will be highlighted in light blue. These graphs are based on reward logs collected during training.

   The graph will also show divisions into sections and indicate which seed was chosen for the best and the worst performance in each section.

   Due to potential visibility issues, information about rewards, seed choice, and other details can be found in Table 16 (see Appendix A), where it was originally sourced for the graph.

2. **SMAPE table**

   To quantify the impact of each source per section in percentage terms, *SMAPE (Symmetric Mean Absolute Percentage Error)* is used to calculate the difference between the best and worst performance curves.

4.1 SMAPE

In this analysis, **the Symmetric Mean Absolute Percentage Error (SMAPE)** is employed to quantify the percentage difference between the learning curves of the best and worst performance cases. Originally SMAPE is a metric used to measure the accuracy of predictive models, particularly when comparing predicted values to actual values. It is calculated as follows:

$$\text{SMAPE} = \frac{1}{n} \sum_{i=1}^{n} \frac{2 \times |y_{\text{best},i} - y_{\text{worst},i}|}{|y_{\text{best},i}| + |y_{\text{worst},i}|} \times 100\%$$

where $y_{\text{best},i}$ represents the best performance value and $y_{\text{worst},i}$ represents the worst performance value at each time step i. The resulting SMAPE value represents the average percentage difference between these values, providing an intuitive measure of accuracy that is easy to interpret.

4.2 Results and Observations

Overall, the experiments demonstrate a wide range of results regarding the intensity of the sources' impact and the variations in their individual influence.

For example, in the *BipedalWalkerHardcore-v3* environment using *SAC*, the impact from each source is small and almost identical (see Table 14a and Figure 15a). Notably, the consistently smallest impacts were recorded in the *Pusher-v4* environment with *PPO* (see Table 14b and Figure 15b). A contrasting situation was observed in the *LunarLanderContinuous-v2* environment using *SAC*, where the impacts of the sources were also close to each other but increased up to 75% (see Table 14c and Figure 15c).

However, the impact of sources being nearly equal is not always the case. For example, in the *MountainCarContinuous-v0* environment with *SAC*, the `environment` and `noise` had significantly high values, while the `buffer` and `policy` showed moderate values. The difference in impact between sources reaches more than 50% (see Table 14d and Figure 15d).

The leading source in terms of impact also varies. For example, in *Pendulum-v1* with *SAC*, the `environment` had the greatest impact (see Table 14e and Figure 15e). In *InvertedDoublePendulum-v4*

with *PPO*, `buffer` had the most pronounced effect (see Table 43 and Figure 44). In *InvertedPendulum-v4* with *PPO*, `noise` was the most influential (see Table 14g and Figure 15g), and in *BipedalWalkerHardcore-v3* with *PPO*, `policy` proved to be the most significant (see Table 37 and Figure 38).

Section	env	buffer	noise	policy
all	9.9%	8.5%	8.5%	9.6%

(a) SMAPE for BipedalWalkerHardcore, SAC: the impact is consistently small across all sources

Section	env	buffer	noise	policy
all	6.0%	1.0%	1.8%	0.6%

(b) SMAPE for Pusher, PPO: smallest impacts across all sources

Section	env	buffer	noise	policy
all	65.8%	68.7%	75.6%	68.7%

(c) SMAPE for LunarLanderContinuous, SAC: the impact is consistently large across all sources

Section	env	buffer	noise	policy
all	72.0%	16.3%	91.6%	22.2%

(d) SMAPE for MountainCarContinuous, SAC: impacts of sources are significantly different

Section	env	buffer	noise	policy
all	30.4%	9.5%	13.9%	10.8%

(e) SMAPE for Pendulum, SAC: `environment` has the greatest impact

Section	env	buffer	noise	policy
all	17.9%	41.9%	19.8%	17.2%

(f) SMAPE for InvertedDoublePendulum, PPO: `buffer` has the greatest impact

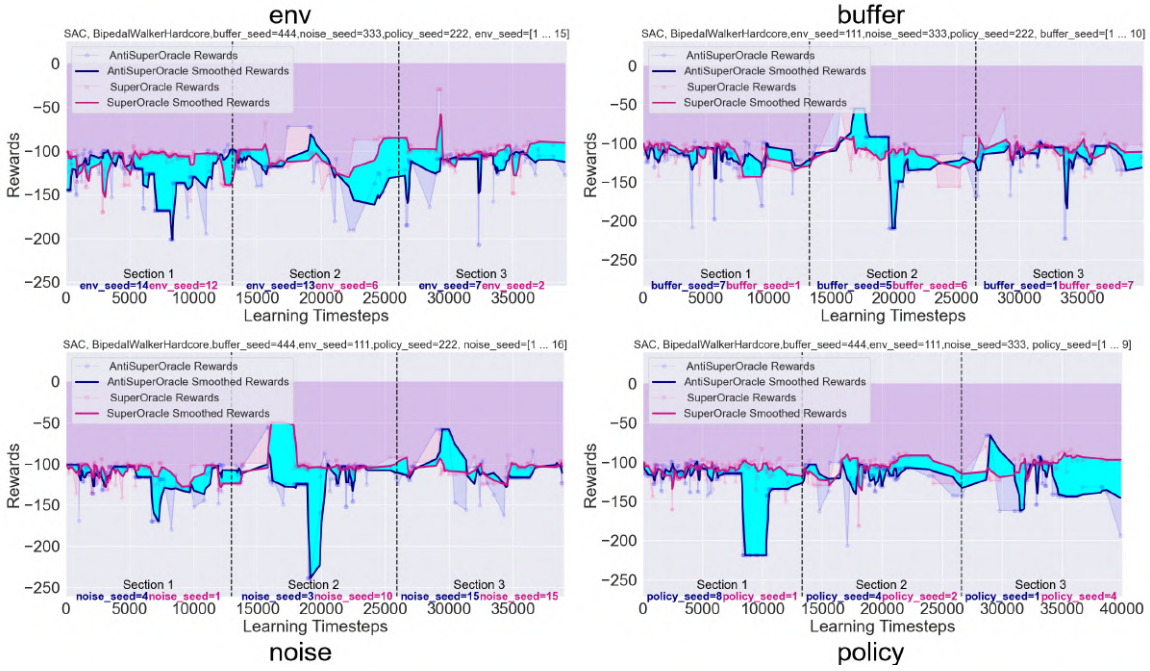
Section	env	buffer	noise	policy
all	26.0%	22.6%	32.5%	23.8%

(g) SMAPE for InvertedPendulum, PPO: `noise` has the greatest impact

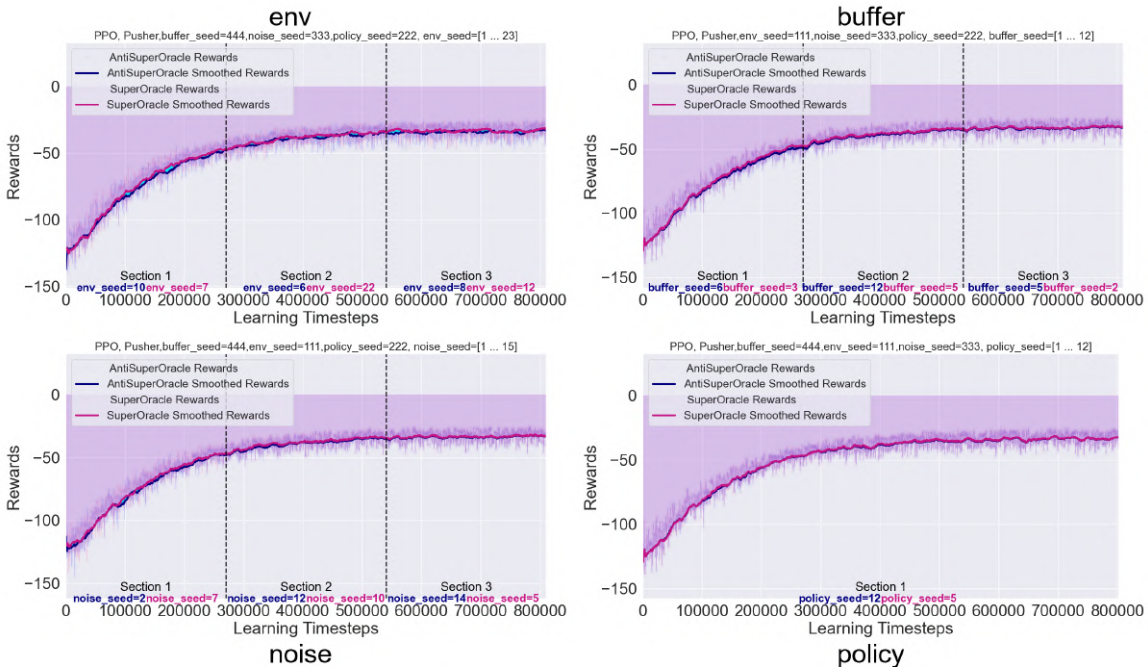
Section	env	buffer	noise	policy
all	10.6%	11.4%	10.5%	13.5%

(h) SMAPE for BipedalWalkerHardcore, PPO: `policy` has the greatest impact

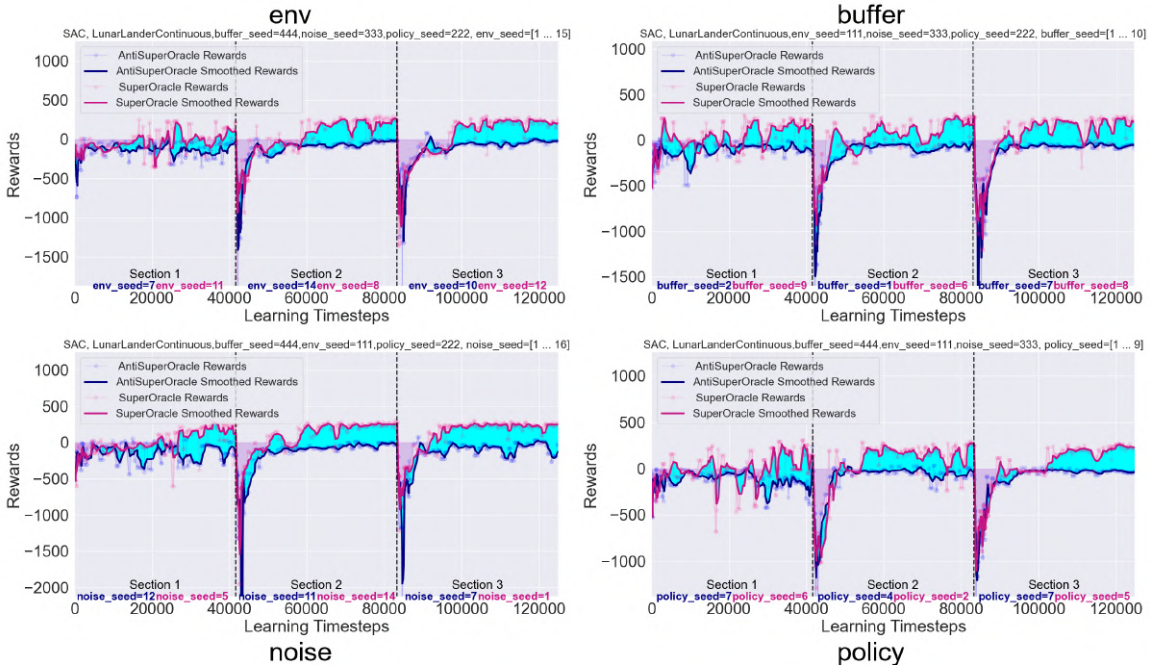
Figure 14: Comparison of SMAPE values across various environments using SAC and PPO.



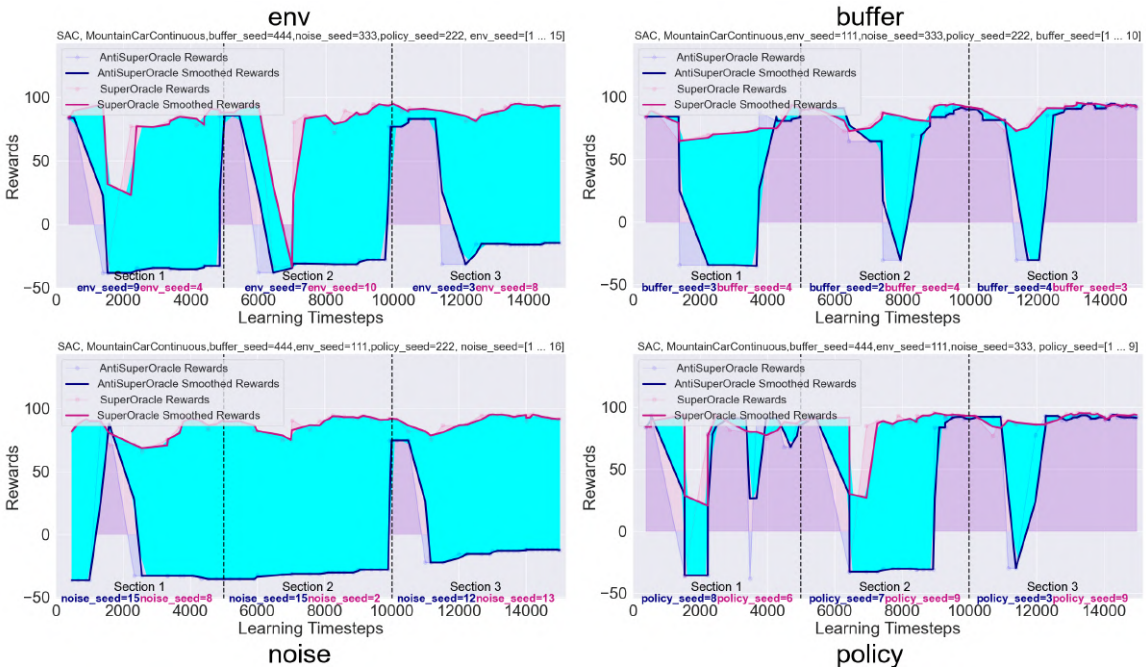
(a) Combined Graph for BipedalWalkerHardcore, SAC: the impact is consistently small across all sources



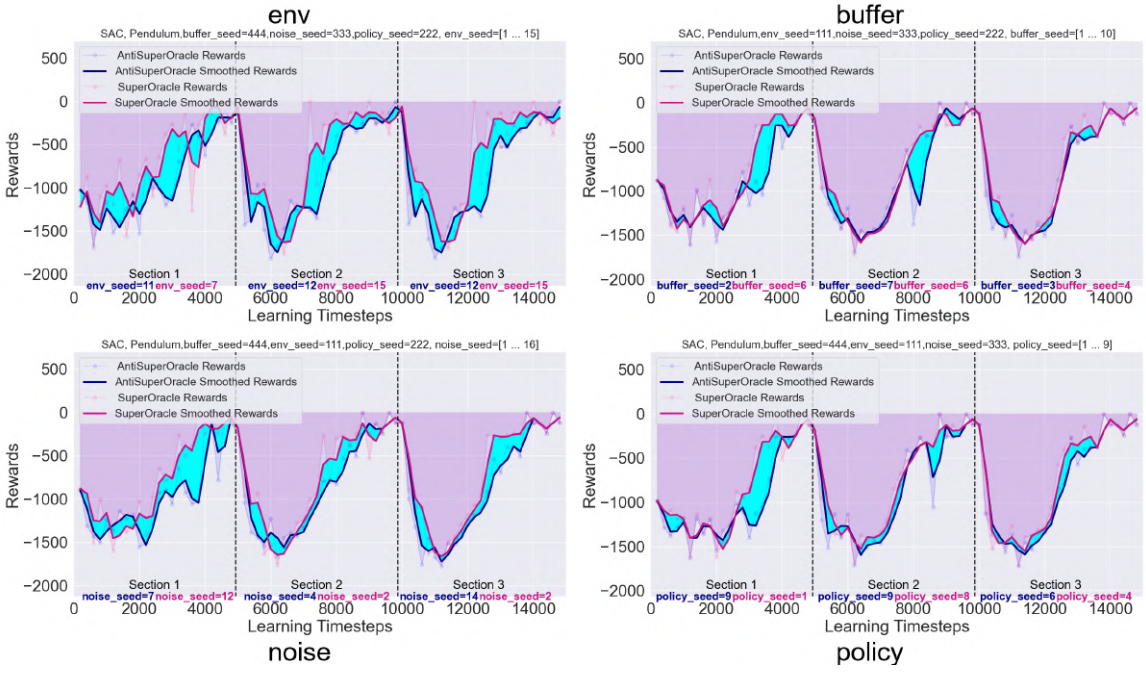
(b) Combined Graph for Pusher, PPO: smallest impacts across all sources



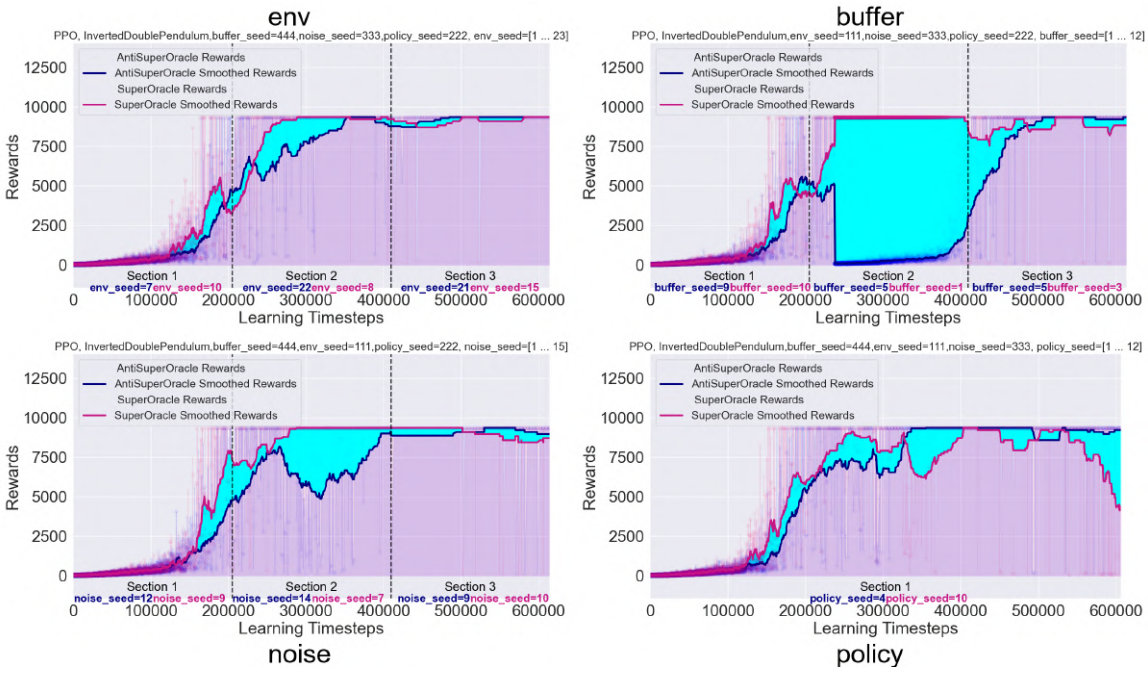
(c) Combined Graph for LunarLanderContinuous, SAC: the impact is consistently large across all sources



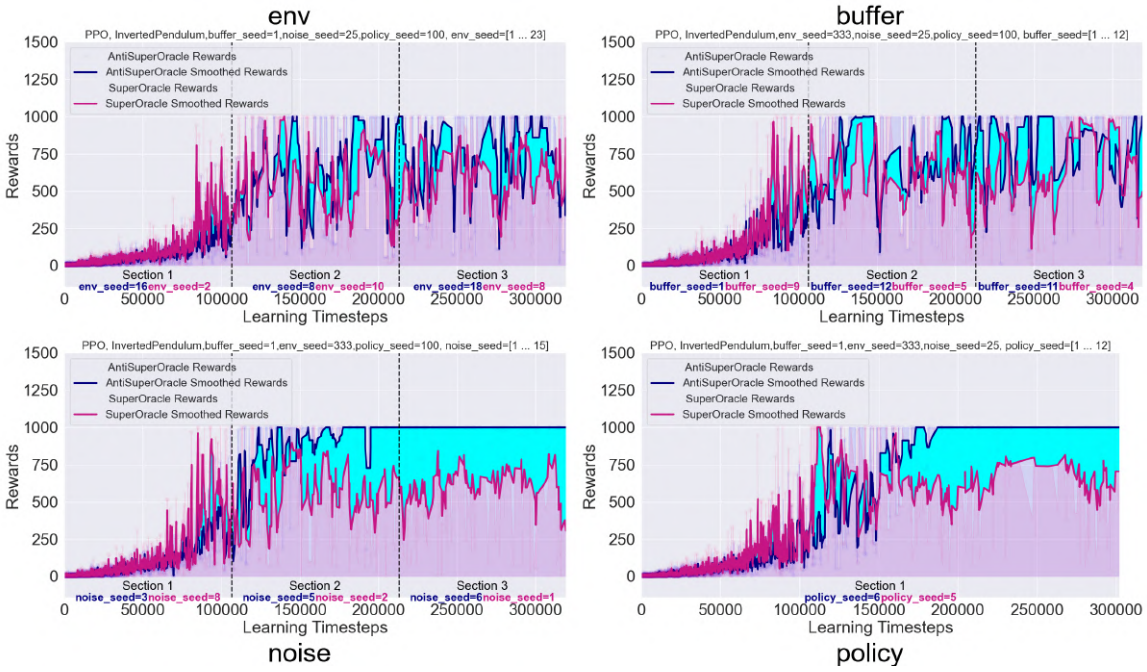
(d) Combined Graph for MountainCarContinuous, SAC: impacts of sources are significantly different



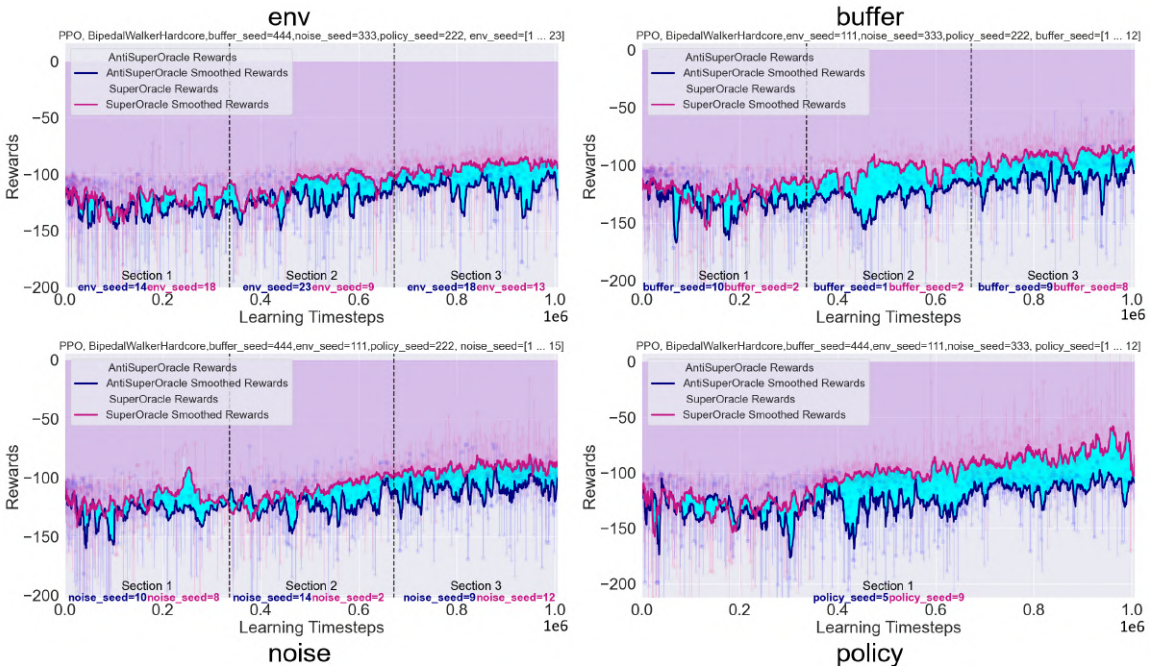
(e) Combined Graph for Pendulum, SAC: `environment` has the greatest impact



(f) Combined Graph for InvertedDoublePendulum, PPO: `buffer` has the greatest impact



(g) Combined Graph for InvertedPendulum, PPO: `noise` has the greatest impact



(h) Combined Graph for BipedalWalkerHardcore, PPO: `policy` has the greatest impact

Figure 15: Comparison of combined graphs across multiple environments and algorithms

5 Analysis of Results and Interpretation

In this section, the analysis and interpretation of results from the final SMAPE table (see Table 17, Appendix B) will be conducted using several methods: **equality of source impacts in environmental context**, **source ranking**, **data visualization**, **descriptive statistics** and **analysis of variance (ANOVA)**.

The first two methods—**equality of source impacts in environmental context** and **source ranking**—provide a brief overview of the results. Therefore, the focus will be solely on the outcomes of the *entire* training process (labeled as *section: all* in the SMAPE tables).

In contrast, **data visualization**, **descriptive statistics**, and **analysis of variance (ANOVA)** will provide a more detailed analysis, considering the division into sections. The goal of this analysis is to explore potential patterns or the absence of such patterns across different sources and sections.

5.1 Equality of Source Impacts in Environmental Context

Let us examine how often it occurs that the values of all sources are relatively equal in a given environment. Table 12 presents results for each environment and algorithm, indicating whether the impact of all four sources— `environment`, `buffer`, `noise`, and `policy` — was relatively close to equal. A value of *Equal* signifies that the sources were aligned within a 10% margin of error, while a value of *Different* indicates that they were not.

Section	Environment	SAC	PPO
all	Humanoid-v4	Equal	Equal
all	HumanoidStandup-v4	Equal	Equal
all	Walker2d-v4	Different	Equal
all	Ant-v4	Different	Equal
all	BipedalWalker-v3	Equal	Different
all	BipedalWalkerHardcore-v3	Equal	Equal
all	CarRacing-v2	-	Equal
all	HalfCheetah-v4	Equal	Different
all	Hopper-v4	Equal	Equal
all	InvertedDoublePendulum-v4	Equal	Different
all	InvertedPendulum-v4	Equal	Equal
all	LunarLanderContinuous-v2	Equal	Equal
all	MountainCarContinuous-v0	Different	Different
all	Pendulum-v1	Different	Different
all	Pusher-v4	Equal	Equal
all	Reacher-v4	Different	Different

Table 12: Equality of impacts among all sources of randomness

We can observe that a larger percentage of cases demonstrates a relatively equal impact among all sources. Specifically, the percentage of *Equal* values is 65%, while the percentage of *Different* values is 35%.

The percentage of cases where SAC and PPO produced identical outcomes in terms of the relative equality or inequality of source impacts is 66%, indicating a considerable level of similarity between these two algorithms.

The environments where the impact across sources is consistently **equal** for both *SAC* and *PPO* are:

- *Humanoid-v4*

- *HumanoidStandup-v4*
- *BipedalWalkerHardcore-v3*
- *Hopper-v4*
- *InvertedPendulum-v4*
- *LunarLanderContinuous-v2*
- *Pusher-v4*

On the other hand, the environments where the impact across sources is **different** include:

- *MountainCarContinuous-v0*
- *Pendulum-v1*
- *Reacher-v4*

5.2 Source Ranking

Next, let us examine how frequently each source took the lead in terms of impact, how often the opposite occurred, and other relevant comparisons.

In this analysis, for each algorithm and environment pair, all sources were ranked from 1 to 4 based on their impact values. A rank of 1 represents the smallest impact, while a rank of 4 represents the largest. The results are presented in Table 13, with the ranking numbers also color-coded for clarity *(1: dark blue, 2: cyan, 3: light red, 4:red)*.

Section	Environment	SAC				PPO			
		env	buffer	noise	policy	env	buffer	noise	policy
all	Humanoid-v4	2	3	1	4	3	1	4	2
all	HumanoidStandup-v4	2	1	4	3	2	1	4	3
all	Walker2d-v4	2	4	3	1	3	1	4	2
all	Ant-v4	3	4	1	2	1	2	3	4
all	BipedalWalker-v3	3	4	1	2	2	1	4	3
all	BipedalWalkerHardcore-v3	4	2	1	3	2	3	1	4
all	CarRacing-v2	-	-	-	-	4	1	2	3
all	HalfCheetah-v4	3	1	4	2	1	2	4	3
all	Hopper-v4	3	1	4	2	2	1	4	3
all	InvertedDoublePendulum-v4	2	1	4	3	2	4	3	1
all	InvertedPendulum-v4	1	2	4	3	3	1	4	2
all	LunarLanderContinuous-v2	1	3	4	2	3	1	4	2
all	MountainCarContinuous-v0	3	1	4	2	2	1	4	3
all	Pendulum-v1	4	1	3	2	4	2	3	1
all	Pusher-v4	4	1	3	2	4	2	3	1
all	Reacher-v4	4	1	3	2	4	2	3	1

Table 13: SMAPE Rankings across algorithms and environments

It is evident that the `noise` column is the most prominent red, while the `buffer` column is full of dark blue for both algorithms. This indicates that `noise` has the highest impact, whereas `buffer` has the lowest.

To summarize and streamline the table, the total count of each rank for every source is calculated, as shown in Table 14.

Section	Algorithm	Source	1	2	3	4
all	all	env	4	10	9	8
all	all	buffer	17	7	3	5
all	all	noise	5	1	9	16
all	all	policy	5	13	9	3

Table 14: Count for each rank across all sources

As previously mentioned, the most distinct pattern suggests that the `buffer` usually has the smallest impact, while `noise` consistently has the largest. `Policy` often ranks second in impact.

It is important to note that the ranking reflects only the order of impact, not the magnitude of difference between impact values. A more detailed analysis will follow in the next three sections.

5.3 Data Visualization: Scatter Plot (Figure16)

Starting from this section, we will perform a deeper analysis and expand it to include not only *Section: all*, but also the other sections.

First, let us plot the values from Table 17 which contains the complete SMAPE data and is located in Appendix B. The plot will account for the source of randomness, section, and SMAPE, which quantifies the difference between the best and worst performance.

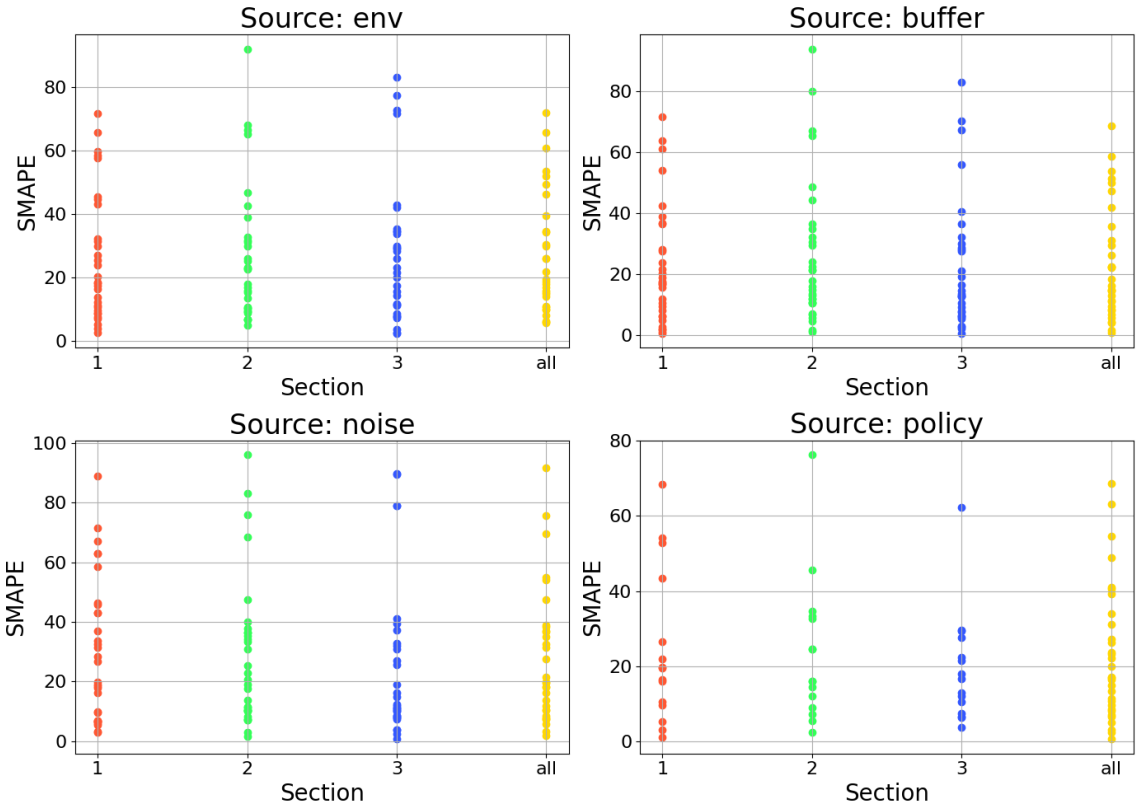


Figure 16: SMAPE values for different sources across sections.

From Figure 16 , we can observe that the majority of SMAPE values range from 0 to 40-45%, indicating that the sources of randomness typically affect model performance up to 45%

5.4 Descriptive Statistics 15

Descriptive statistics summarize the central tendencies, dispersion, and overall distribution of values across different groups. For each combination of `source` and `section`, the following calculations are performed:

- The **mean** (μ), which represents the average SMAPE value.
- The **standard deviation** (σ), which measures the spread of the values around the mean.
- The **quartiles (25th, 50th, and 75th percentiles)**. The 25th percentile marks the point below which 25% of the values lie, the 50th percentile (or median) indicates the middle value where half of the data is above and half is below, and the 75th percentile shows the value below which 75% of the data falls. Together, these quartiles offer a clearer picture of how the SMAPE values are spread across the data set.

Source	Section	Count	Mean	Std	Min	25%	50%	75%	Max
buffer	1	31	22.5	19.6	0.7	7.2	17.2	32.3	71.7
buffer	2	31	26.2	23.4	1.1	10.8	17.9	33.5	93.8
buffer	3	31	22.3	21.4	0.6	7.1	13.5	29.4	82.9
buffer	all	31	23.4	18.7	1.0	9.0	16.3	33.5	68.7
env	1	31	26.1	20.1	2.6	10.2	18.5	37.8	71.7
env	2	31	26.8	21.6	4.9	10.3	22.7	32.2	92.2
env	3	31	27.6	22.3	2.3	11.3	23.2	34.5	83.2
env	all	31	27.2	19.0	5.7	12.5	19.5	37.1	72.0
noise	1	31	30.2	23.7	3.0	8.2	26.9	44.5	88.9
noise	2	31	28.2	24.4	1.5	10.2	20.7	36.5	96.1
noise	3	31	25.2	25.8	0.6	8.5	12.3	32.5	89.7
noise	all	31	28.2	22.6	1.8	10.5	19.8	37.6	91.6
policy	1	15	24.6	20.6	1.2	10.2	19.7	35.1	68.5
policy	2	15	23.7	19.2	2.5	10.6	16.1	33.0	76.3
policy	3	15	19.2	14.7	3.7	9.0	16.7	25.0	62.2
policy	all	31	23.1	18.1	0.6	9.1	17.2	32.6	68.7

Table 15: Descriptive statistics for SMAPE by source and section

5.4.1 Interpretation

- As previously observed in Figure 16, we can confirm once again that 75% of the values fall below 45% (SMAPE), which is the upper value indicated by the 75th. Additionally, 50% of the values fall below 27% (SMAPE), indicating that with a 50% probability, performance can be affected up to 27% in SMAPE, and with a 75% probability, performance can be affected up to 45% in SMAPE.
- The highest mean value is observed in the `noise` source, particularly in Section 1, where it reaches 30.2% SMAPE. Conversely, the lowest average value is found in the `policy` source in Section 3, with 19.2% SMAPE."
- The `buffer`, `env`, and `noise` sources in Section 2 show some extreme maximum values, reaching 94%, 92%, and 96% in SMAPE, respectively.

- In the **Section 1** `noise` shows the highest mean, indicating it has the most significant impact in this section, and `buffer` - lowest.
- In the **Section 2** - `noise` again has the largest impact and `policy` - smallest.
- In the **Section 3** `env` has the largest impact and `policy` - smallest.
- Overall, `buffer` and `policy` exhibit slightly lower mean values compared to `env` and `noise`.
- Although a few leading values were identified, clear patterns were not detected, as the values are relatively close to one another and not consistent over the dataset.

5.5 Data Visualization: Boxplot (Figure 17)

Boxplot offers a way to visualize the spread and skewness of values for each combination of `source` and `section`. Key elements in a boxplot include:

- The **interquartile range** (IQR), represented by the box, which contains the middle 50% of the data. The lower border of the box represents the 25th percentile and the upper border - the 75th percentile. Thus, the height of the box covers the range from the 25th percentile to the 75th percentile.
- The **median**, marked as a line inside the box, representing the 50th percentile of the data.
- The **whiskers**, which extend to 1.5 times the IQR, highlighting the range of non-outlier values.
- Any values outside the whiskers are considered **outliers**.

5.5.1 Interpretation

** In each subsequent interpretation, only newly discovered aspects will be highlighted, avoiding repetition of earlier points.*

- **Env Source** shows a relatively consistent spread of SMAPE values. The median SMAPE values for `env` are close to each other, with slightly higher value in Section 3. This suggests that the `env` source has a somewhat greater impact in Section 3.
- **Buffer Source** demonstrates a bit higher consistency compared to **env**, has a similar impact across Sections 1 and 2, and its impact is slightly lower in Section 3.
- **Noise Source** exhibits the highest overall variability, with the highest(over all) median in Section 1 and the lowest(over all) in Section 3. This indicates that noise has the strongest impact in Section 1 and the weakest in Section 3 among all sources.
- **Policy Source** shows the smallest spread of SMAPE values overall for Section 3. The median values are close to each other with the highest in Section 1 and lowest in Section 3. Although some outliers are present, they are fewer in number, indicating stability.
- **Section 1** and **Section 3** exhibit the greatest variability in performance impacts across the sources.
- **Section 2** is the most stable section in terms of the impact of the sources.
- The overall impact does not seem to be widely distributed. The ranking from highest to lowest impact is as follows: **noise**, **env**, **policy**, **buffer**.

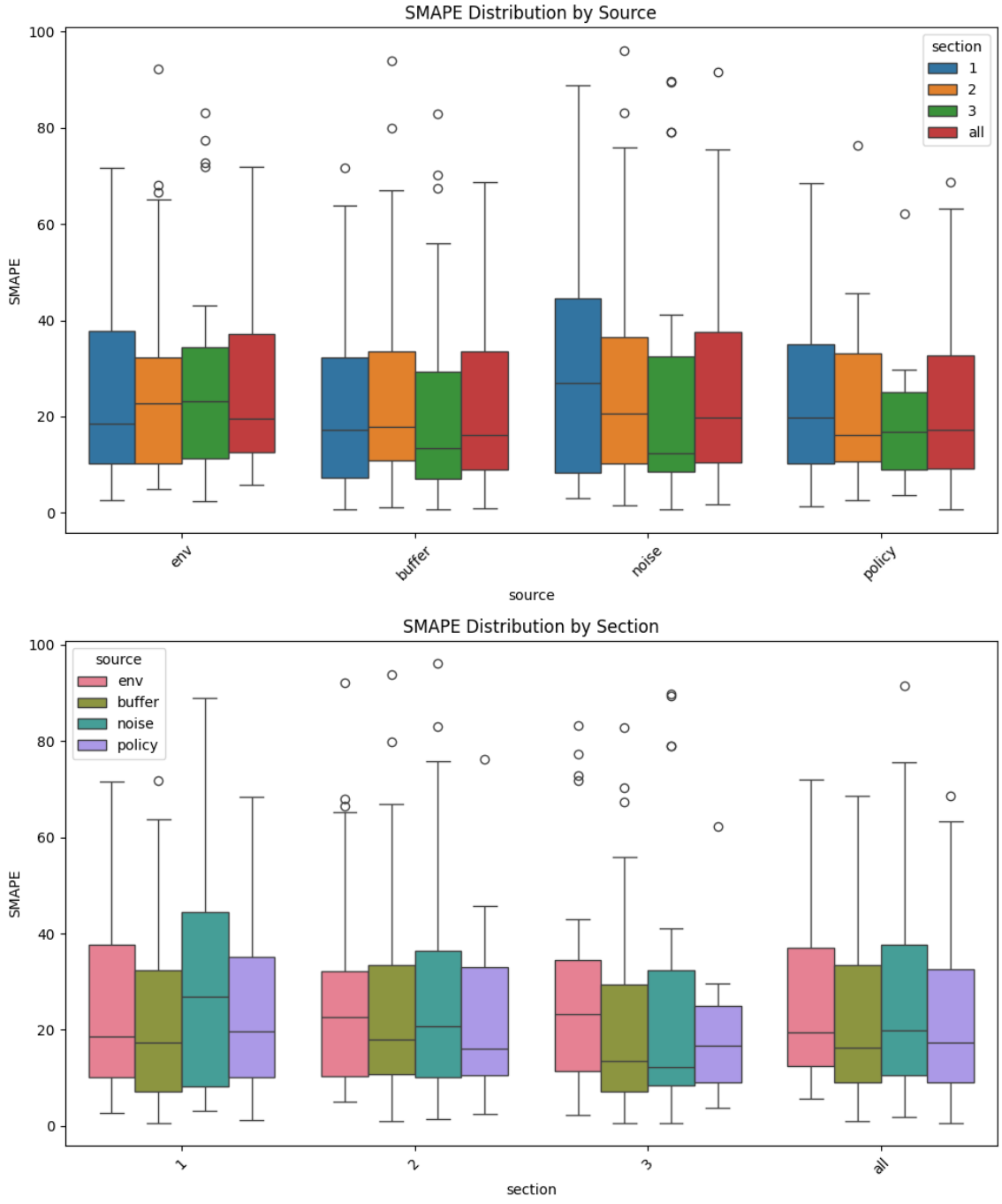


Figure 17: Boxplot showing SMAPE distributions for different `source` and `section` combinations

5.6 Data Visualization: Heatmap (Figure 18)

A **heatmap** is used to display the average SMAPE for each combination of `source` and `section`. It uses color intensity to indicate performance difference: darker colors represent higher SMAPE values (large difference in worst and best performance), while lighter colors represent lower SMAPE values (smaller difference in worst and best performance).

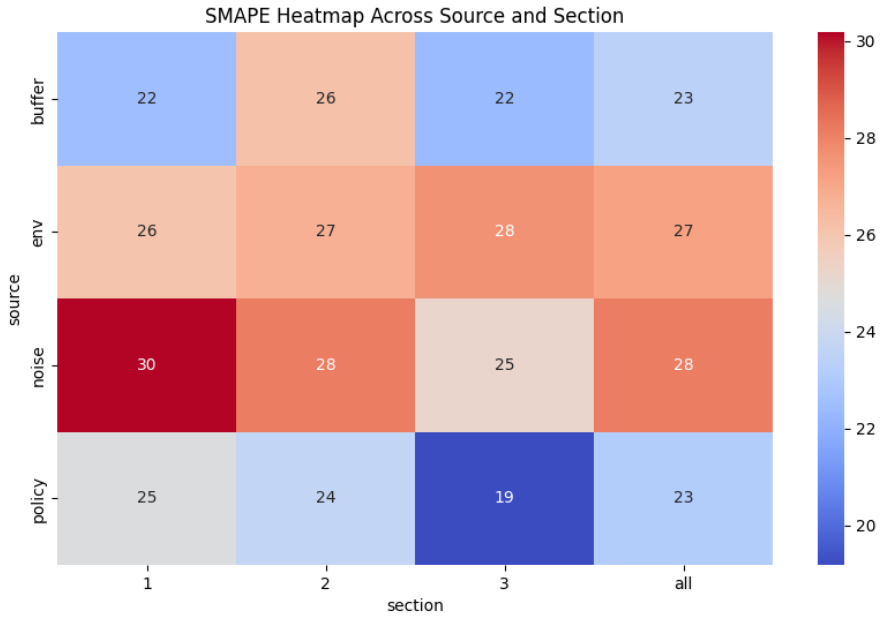


Figure 18: Heatmap of average SMAPE values for each combination of `source` and `section`

5.6.1 Interpretation

- The `env` source tends to increase its impact with each subsequent section, whereas the `noise` and `policy` show a decreasing trend in impact over time.
- Overall, `env` and `noise` have a similar impact, as do `buffer` and `policy`. However, the impact of the first group is slightly stronger.

5.7 Analysis of Variance (ANOVA)

ANOVA compares the variance between group means for two categorical variables (in this case, `source` and `section`) and their interaction. This allows us to assess whether the differences in group means are significant for each factor individually and whether there is a significant interaction between the factors.

ANOVA tests the following:

- **Main effect of `source`**: Tests whether the mean SMAPE values for different `source` categories (`env`, `buffer`, `noise`, and `policy`) are significantly different.
- **Main effect of `section`**: Tests whether the mean SMAPE values for different `section` categories (`1`, `2`, `3`, and `all`) are significantly different.
- **Interaction effect between `source` and `section`**: Tests whether specific combinations of `source` and `section` categories (e.g., `env-1`, `buffer-2`, etc.) have a unique effect on SMAPE that differs from the individual effects of `source` or `section` alone.

The ANOVA output includes the **F-statistic** and **p-value** for each factor and their interaction. The F-statistic measures the ratio of variance between groups to variance within groups, while the p-value tells us whether these differences are statistically significant. A **p-value** below 0.05 indicates

that there is a significant difference in SMAPE values between groups, meaning that the group categories or their interaction have a real impact on SMAPE.

- **ANOVA for Source**:
  F-statistic = 1.45, **p-value** = 0.23.
- **ANOVA for Section**:
  F-statistic = 0.23, **p-value** = 0.87.
- **Interaction between Source and Section**:
  F-statistic = 0.16, **p-value** = 0.997.

5.7.1 Interpretation

A **p-value** greater than 0.05 indicates that there is **no** statistically significant difference in SMAPE values across the different `source`, `section`, or their interaction.

The results show that neither the `source`, `section`, nor the combination of `source` and `section` has a statistically significant effect on SMAPE values. This suggests that, while variability exists in SMAPE values, these differences are not large enough to be considered meaningful or systematic across the categories and their interactions.

6 Conclusion

Based on the *descriptive statistical analysis*, it can be concluded that 75% of the data shows the impact of any source falls below 45% (SMAPE), which is the upper value indicated by the 75th percentile. Additionally, 50% of the data suggests the impact will not exceed 27% (SMAPE).

Using the *Equality of Source Impacts in Environmental Context* method, it was found that in 66% of cases, the impact of sources within the current environment is nearly equal, with a difference of less than 10% in SMAPE. While some environments consistently show balanced source impacts, others display significant disparities across different sources. A list of these environments can be found in section 5.1.

The highest impact record occurs in the first section, where `noise` had the greatest influence, reaching 92% (SMAPE). The lowest occurred in the third section caused by the `policy`, accounting for only 0.6% (SMAPE).

All analyses agree on the following : The sources can be grouped based on their impact values. The first group, consisting of `environment` and `noise` exert a slightly greater influence, while the second group, comprising `buffer` and `policy` show lower impacts. Ranking the sources from the highest to lowest impact: `noise`, `environment`, `buffer`, and `policy`.

The first section shows the highest impact across all sections from `noise`, and its influence consistently decreases with each subsequent section.

In contrast, the impact of `environment` increases progressively with each section.

The second section stands out as the most stable in terms of impact range across all sources.

Focusing on the middle 50% of the data, it is notable that `noise` has both the highest overall impact (in the first section) and lowest overall impact (in the third section), making it the most variable source across sections in terms of performance influence.

7 Future work

In this thesis, four sources of randomness-`environment`, `buffer`, `noise`, `policy`- were investigated, using the optimal seed range for each, determined in section 3.7. For each experiment, three of the sources of randomness were held constant (their seeds), while for the fourth source—the one under investigation—all the values out of the defined seed range were tested. The primary focus was on exploring the behavior of the *source under investigation* and assessing the impact of *its* seed values. Due to constraints related to computational resources, throughout the experimental process a single set of *fixed seeds* was used as outlined in Table 16 (see Appendix A). Specifically, the *fixed seeds* were set to 111 for the `environment`, 222 for the `buffer`, 333 for `noise`, and 444 for the `policy`. Naturally, the fixed seed for the source under investigation was ignored in favor of testing the defined range of seeds.

For future work, it would be beneficial to repeat the experiments with different sets of *fixed seeds*. This would influence the outcome of individual experiments and help to verify if the mean values and conclusions drawn in this work remain consistent. While the current results are theoretically expected to hold—since altering the *fixed seeds* should not significantly influence *overall* impact range of source being tested—validating this assumption through additional experiments would strengthen the findings and could potentially reveal underlying patterns that were not apparent in the initial setup.

As outlined in section 3.4, the learning process was divided into three sections—beginning, middle, and end—to study the effects of randomness across different stages of training. Although the division parameter is configurable, in this research it was used only with its default value of **3**. However, a more granular division could offer deeper insights into how randomness influences performance at various stages, providing a more detailed understanding of learning dynamics.

Another potential avenue for future work is to expand the range of seeds tested for each source of randomness. For instance, while the maximum range explored in this research was 23 (i.e., seeds from 1 to 23), it would be beneficial to extend this to a larger range, such as up to 100. With increased computational resources, running experiments with a broader seed range would boost confidence in the results by ensuring that the best and worst performance cases are identified reliably, or at least with a higher probability. If the results and conclusions remain consistent with a larger seed range, it would also confirm the robustness of the seed range selection methodology described in Section 3.7. Alternatively, it would reveal how much information may be overlooked when using a smaller seed range.

Furthermore, this research studied the impact of each source of randomness *isolated*. A logical extension would be to investigate the combined effects of multiple sources of randomness. This could be done by investigating all possible combinations of sources simultaneously. There could be interactions between different sources, with one source potentially influencing the impact of another. Studying this cross-impact could reveal underlying relationships and provide a more comprehensive understanding of how randomness shapes the learning process.

Finally, there is potential to further subdivide the sources of randomness. For example, in this research, the `policy` source of randomness was treated as a combination of both actor and critic weights, as was explained in Section 2.3. Future work could explore these components separately to determine if they exert distinct influences on performance.

Considering the limited research on this topic, a thorough analysis could offer valuable insights into how various sources of randomness, as well as randomness as a whole, influence the overall behavior of deep reinforcement learning systems.

References

[1] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. *Published as a conference paper at ICLR 2023*, 2023.

[2] Farama Foundation. Gymnasium: An api standard for reinforcement learning with a diverse collection of reference environments. *Retrieved from* https://gymnasium.farama.org/index.html, 2023.

[3] Jakob Hollenstein and contributors. Colored noise ppo paper code. `https://github.com/jkbjh/cn-ppo-paper-code`, 2023.

[4] Jakob Hollenstein, Georg Martius, and Justus Piater. Colored noise in ppo: Improved exploration and performance through correlated action sampling. *Universität Innsbruck, Max Planck Institute for Intelligent Systems, Eberhard Karls University*, 2024.

[5] Dhanoop Karunakaran. The actor-critic reinforcement learning algorithm, September 30 2020. Intro to Artificial Intelligence, Medium.

[6] Gym Library. Mujoco environments. *Retrieved from* https://www.gymlibrary.dev/environments/mujoco/index.html, 2022.

[7] Georg Martius and contributors. Pink noise rl: Colored noise exploration in deep reinforcement learning. `https://github.com/martius-lab/pink-noise-rl/tree/main`, 2023.

[8] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. *arXiv preprint arXiv:2206.08795*, 2022.

[9] M. Saveriano. Reinforcement learning. *Lecture slides, Machine Learning.*

[10] Stable-Baselines3. Stable-baselines3 docs - reliable reinforcement learning implementations. *https://stable-baselines3.readthedocs.io/en/master/*, 2024.

[11] Elizaveta Terente. Sources of variance in reinforcement learning. GitLab repository, September 2024. Bachelor's Thesis, Leopold-Franzens-Universität Innsbruck, Innsbruck, Austria.

A Complete results table

Here you will find a summary of all the results collected during experiments.

This file discussed earlier in 3.5 and contains information about whether it's a *SuperOracle* (*anti*=False) or *AntiSuperOracle* (*anti*=True), the fixed seeds values, the source of interest, and the selected best/worst performing seed value along with its corresponding reward.

**To improve readability in the table, vertically duplicated values in cells are written only once in the first row.*

Previously in example Table 7 it was shown that the source under investigation does not have a fixed seed value - it was* ***None*. However, for large-scale experiments, it was more convenient to set fixed values for* ***all*** *sources of randomness. The fixed seed for the source of randomness under investigation is simply ignored.*

anti	algorithm	section	learning_timesteps	environment	env_seed	policy_seed	noise_seed	buffer_seed	source	seed_value	avg_reward
True	SAC	1	5000	Pendulum-v1	111	222	333	444	env	11	-894.3
		2								12	-850.0
		3								12	-839.4
False		1								7	-714.8
		2								15	-684.2
		3								15	-670.2
True		1							buffer	2	-903.1
		2								7	-870.5
		3								3	-841.1
False		1								6	-797.6
		2								6	-787.0
		3								4	-772.0
True		1							noise	7	-968.2
		2								4	-900.5
		3								14	-881.7
False		1								12	-780.1
		2								2	-768.8
		3								2	-749.6
True		1							policy	9	-965.1
		2								9	-897.7
		3								6	-860.9
False		1								1	-840.2
		2								8	-797.8
		3								4	-775.5
True		1	8333	BipedalWalker-v3					env	2	-118.6
		2								3	-117.3
		3								3	-117.3
False		1								4	-110.8
		2								14	-107.7
		3								14	-104.9

True		1							buffer	8	-118.9
		2								3	-119.0
		3								4	-118.1
False		1								1	-112.8
		2								6	-108.9
		3								2	-106.5
True		1							noise	11	-122.6
		2								6	-121.8
		3								5	-120.1
False		1								3	-110.1
		2								16	-106.9
		3								16	-105.5
True		1							policy	9	-119.0
		2								7	-118.3
		3								1	-118.7
False		1								4	-112.4
		2								3	-109.3
		3								3	-108.3
True		1	13333	BipedalWalkerHardcore-v3					env	14	-119.4
		2								13	-120.3
		3								7	-118.1
False		1								12	-109.1
		2								6	-109.0
		3								2	-106.9
True		1							buffer	7	-117.7
		2								5	-119.2
		3								1	-120.0
False		1								1	-110.4
		2								6	-111.0
		3								7	-109.4
True		1							noise	4	-119.0
		2								3	-118.4
		3								15	-117.7
False		1								1	-108.8
		2								10	-106.0
		3								15	-105.2
True		1							policy	8	-117.7
		2								4	-118.3
		3								1	-119.1
False		1								1	-109.3
		2								2	-107.8
		3								4	-106.1
True		1	41667	LunarLanderContinuous-v2					env	7	-155.8
		2								14	-200.6
		3								10	-222.9
False		1								11	-43.6
		2								8	-44.6
		3								12	-30.0
True		1							buffer	2	-134.4

		2								1	-199.9
		3								7	-255.6
False		1								9	11.2
		2								6	-10.3
		3								8	-24.0
True		1							noise	12	-161.0
		2								11	-237.1
		3								7	-255.4
False		1								5	4.3
		2								14	20.9
		3								1	43.2
True		1							policy	7	-140.3
		2								4	-172.3
		3								7	-204.3
False		1								6	-48.1
		2								2	-25.3
		3								5	-50.7
True		1	10000	Ant-v4					env	13	-264.6
		2								5	-86.2
		3								14	92.8
False		1								10	-122.4
		2								15	-4.0
		3								9	106.4
True		1							buffer	5	-241.5
		2								8	-50.3
		3								2	127.8
False		1								10	-125.5
		2								4	-19.5
		3								2	87.5
True		1							noise	2	-249.7
		2								3	-48.1
		3								13	131.1
False		1								4	-140.8
		2								1	-21.0
		3								6	105.5
True		1							policy	3	-222.6
		2								8	-33.5
		3								1	119.3
False		1								1	-120.6
		2								4	2.6
		3								3	118.3
True		1	30000	HalfCheetah-v4					env	1	-3.2
		2								15	288.9
		3								11	376.6
False		1								6	371.9
		2								6	632.1
		3								13	725.7
True		1							buffer	6	101.7
		2								8	290.4

		3								3	385.6
False		1								1	247.8
		2								9	568.8
		3								5	693.3
True		1							noise	5	-12.4
		2								5	229.3
		3								16	486.1
False		1								16	236.0
		2								13	583.7
		3								16	919.0
True		1							policy	7	-4.1
		2								8	219.5
		3								3	352.7
False		1								6	261.1
		2								2	525.0
		3								2	594.4
True		1	11667	Hopper-v4					env	13	111.9
		2								15	118.7
		3								8	122.8
False		1								12	140.1
		2								15	156.0
		3								7	162.5
True		1							buffer	1	108.8
		2								9	113.5
		3								6	118.6
False		1								7	131.2
		2								6	142.1
		3								9	147.3
True		1							noise	1	115.0
		2								8	128.3
		3								15	124.5
False		1								10	141.6
		2								11	153.3
		3								11	157.1
True		1							policy	2	105.6
		2								2	124.5
		3								2	128.6
False		1								5	141.8
		2								6	167.0
		3								7	175.3
True		1	10000	Reacher-v4					env	9	-24.5
		2								1	-18.0
		3								1	-15.4
False		1								13	-23.9
		2								8	-17.3
		3								15	-14.7
True		1							buffer	3	-24.4
		2								4	-17.7
		3								9	-15.1

False		1								5	-24.2
		2								10	-17.4
		3								10	-14.8
True		1							noise	6	-24.8
		2								9	-18.0
		3								12	-15.4
False		1								11	-24.0
		2								3	-17.1
		3								3	-14.5
True		1							policy	3	-24.3
		2								7	-17.6
		3								7	-15.1
False		1								8	-24.0
		2								5	-17.2
		3								8	-14.5
True		1	20000	InvertedPendulum-v4					env	10	32.6
		2								7	40.5
		3								9	43.3
False		1								5	39.3
		2								10	56.8
		3								14	55.5
True		1							buffer	1	32.4
		2								2	42.4
		3								6	42.7
False		1								3	39.1
		2								5	47.2
		3								10	47.8
True		1							noise	2	29.5
		2								1	37.5
		3								12	40.8
False		1								8	44.0
		2								14	52.5
		3								13	59.1
True		1							policy	1	32.2
		2								7	41.1
		3								3	44.9
False		1								9	38.9
		2								9	52.6
		3								7	63.9
True		1	20000	InvertedDoublePendulum-v4					env	2	392.7
		2								15	410.2
		3								2	409.9
False		1								5	451.1
		2								10	460.0
		3								13	454.0
True		1							buffer	8	376.4
		2								1	382.6
		3								9	388.8
False		1								10	427.1

		2								2	425.0
		3								7	419.6
True		1							noise	9	383.8
		2								9	397.9
		3								10	396.9
False		1								11	442.2
		2								14	441.4
		3								5	437.3
True		1							policy	4	364.8
		2								4	402.8
		3								4	416.0
False		1								8	426.9
		2								9	456.5
		3								4	469.8
True		1	10000	Pusher-v4					env	4	-120.0
		2								12	-87.6
		3								12	-74.6
False		1								3	-118.1
		2								2	-85.3
		3								2	-72.1
True		1							buffer	2	-118.5
		2								10	-85.9
		3								2	-73.0
False		1								9	-117.5
		2								5	-85.2
		3								1	-71.9
True		1							noise	14	-121.1
		2								15	-87.9
		3								1	-74.8
False		1								12	-118.0
		2								1	-85.7
		3								14	-72.7
True		1							policy	2	-118.8
		2								8	-86.2
		3								5	-73.0
False		1								5	-117.5
		2								1	-84.6
		3								3	-71.5
True		1	5000	MountainCarContinuous-v0					env	9	4.7
		2								7	5.1
		3								3	8.0
False		1								4	74.3
		2								10	77.3
		3								8	83.7
True		1							buffer	3	40.3
		2								2	58.3
		3								4	70.2
False		1								4	80.9
		2								4	85.5

		3								3	88.1
True		1							noise	15	-10.1
		2								15	-12.1
		3								12	-7.9
False		1								8	84.2
		2								2	87.5
		3								13	89.0
True		1							policy	8	57.6
		2								7	55.3
		3								3	67.4
False		1								6	73.9
		2								9	80.6
		3								9	85.8
True		1	20000	Walker2d-v4					env	7	80.8
		2								11	140.0
		3								4	187.7
False		1								14	128.3
		2								11	217.1
		3								15	294.7
True		1							buffer	7	84.3
		2								3	151.3
		3								3	190.6
False		1								4	154.3
		2								6	285.9
		3								6	411.2
True		1							noise	4	88.3
		2								5	150.4
		3								2	200.1
False		1								5	142.7
		2								7	239.0
		3								4	324.1
True		1							policy	5	83.7
		2								7	131.3
		3								4	192.2
False		1								2	128.0
		2								4	185.9
		3								6	223.1
True		1	20000	HumanoidStandup-v4					env	13	69415.5
		2								1	75868.6
		3								12	80189.9
False		1								3	79318.5
		2								15	105236.2
		3								6	115603.4
True		1							buffer	9	65336.0
		2								10	71219.4
		3								2	75948.8
False		1								5	92305.9
		2								10	103460.3
		3								10	108538.7

True		1							noise	3	63322.3
		2								6	67774.4
		3								15	70491.8
False		1								1	91482.1
		2								5	99827.1
		3								13	110652.8
True		1							policy	4	58415.7
		2								9	69734.4
		3								1	73809.7
False		1								6	87317.1
		2								4	98861.8
		3								7	108868.7
True		1	20000	Humanoid-v4					env	11	163.4
		2								3	231.4
		3								9	272.9
False		1								5	179.9
		2								13	249.1
		3								5	296.0
True		1							buffer	3	164.4
		2								4	229.2
		3								6	268.3
False		1								6	175.6
		2								6	245.7
		3								8	288.5
True		1							noise	14	169.5
		2								6	234.6
		3								1	276.4
False		1								9	180.9
		2								5	253.8
		3								4	300.8
True		1							policy	3	170.2
		2								6	234.9
		3								8	275.1
False		1								5	194.1
		2								4	262.8
		3								8	308.0
True	PPO	1	100000	Pendulum-v1					env	11	-1076.0
		2								1	-703.1
		3								13	-536.4
False		1								14	-1025.9
		2								10	-655.3
		3								7	-491.3
True		1							buffer	3	-1075.9
		2								6	-704.9
		3								11	-527.7
False		1								9	-1040.1
		2								12	-659.8
		3								5	-492.6
True		1							noise	2	-1094.1

		2								13	-733.0
		3								14	-545.6
False		1								9	-1042.8
		2								13	-665.0
		3								6	-495.1
True		1	300000						policy	12	-537.1
False										7	-502.8
True			25000	MountainCarContinuous-v0					env	11	29.1
		2								9	60.5
		3								19	74.3
False		1								10	47.1
		2								1	66.5
		3								1	77.8
True		1							buffer	7	32.9
		2								12	60.4
		3								8	74.4
False		1								1	46.7
		2								3	66.5
		3								11	78.4
True		1							noise	1	-2.1
		2								4	43.6
		3								15	67.4
False		1								4	49.4
		2								11	66.0
		3								11	77.8
True		1	75000						policy	12	68.0
False									policy	5	73.0
True			300000	BipedalWalker-v3					env	21	-122.5
		2								4	-104.5
		3								17	-71.2
False		1								1	-85.4
		2								18	17.8
		3								17	109.2
True		1							buffer	5	-119.9
		2								8	-109.8
		3								2	-86.9
False		1								1	-90.0
		2								3	-9.9
		3								11	66.9
True		1							noise	5	-117.0
		2								12	-108.9
		3								15	-94.7
False		1								8	-77.0
		2								1	19.4
		3								11	96.5
True		1	900000						policy	5	-74.9
False										8	69.6
True			100000	LunarLanderContinuous-v2					env	5	-173.4
		2								3	-139.4

		3								23	-126.4
False		1								11	-145.9
		2								4	-109.4
		3								8	-86.1
True		1							buffer	1	-169.5
		2								2	-141.3
		3								12	-125.8
False		1								6	-152.6
		2								7	-119.2
		3								10	-96.7
True		1							noise	5	-172.2
		2								11	-135.8
		3								2	-126.1
False		1								15	-146.8
		2								8	-109.7
		3								11	-81.6
True		1	300000						policy	2	-120.7
False										3	-81.7
True		1	300000	Ant-v4					env	5	-108.2
		2								18	6.9
		3								9	109.0
False		1								12	-91.7
		2								8	6.2
		3								8	107.9
True		1							buffer	2	-107.9
		2								3	4.2
		3								9	100.7
False		1								8	-90.7
		2								2	5.7
		3								1	120.4
True		1							noise	9	-106.8
		2								10	-4.0
		3								14	91.1
False		1								15	-92.0
		2								2	0.5
		3								7	100.4
True		1	900000						policy	8	91.5
False		1								12	126.9
True			333333	BipedalWalkerHardcore-v3					env	14	-129.5
		2								23	-126.5
		3								18	-122.6
False		1								18	-122.2
		2								9	-114.3
		3								13	-105.1
True		1							buffer	10	-131.5
		2								1	-129.1
		3								9	-124.2
False		1								2	-120.6
		2								2	-114.7

		3								8	-105.0
True		1							noise	10	-127.8
		2								14	-126.0
		3								9	-120.7
False		1								8	-122.2
		2								2	-114.3
		3								12	-103.1
True		1	1000000						policy	5	-123.8
False										9	-101.1
True			300000	HalfCheetah-v4					env	5	-129.4
		2								17	331.9
		3								8	633.2
False		1								17	-42.2
		2								9	439.6
		3								1	730.4
True		1							buffer	4	-149.9
		2								6	246.5
		3								8	545.7
False		1								2	-47.8
		2								3	435.4
		3								6	720.9
True		1							noise	7	-176.0
		2								1	112.5
		3								3	417.7
False		1								2	-74.3
		2								11	378.1
		3								10	678.0
True		1	900000						policy	7	476.7
False										9	741.7
True			300000	Hopper-v4					env	2	166.4
		2								5	235.1
		3								2	265.2
False		1								8	181.2
		2								11	263.2
		3								9	313.9
True		1							buffer	4	165.7
		2								11	251.3
		3								3	299.2
False		1								8	177.5
		2								8	268.9
		3								4	321.6
True		1							noise	1	165.6
		2								6	248.1
		3								7	295.0
False		1								4	181.9
		2								2	257.3
		3								10	302.8
True		1	900000						policy	12	273.1
False										11	317.9

True			200000	Reacher-v4					env	13	-27.0
		2								4	-16.8
		3								13	-12.7
False		1								16	-26.5
		2								20	-16.0
		3								20	-12.0
True		1							buffer	11	-26.8
		2								8	-16.8
		3								7	-12.7
False		1								2	-26.5
		2								5	-16.3
		3								11	-12.3
True		1							noise	2	-26.8
		2								15	-16.4
		3								15	-12.4
False		1								7	-26.2
		2								8	-16.3
		3								13	-12.3
True		1	600000						policy	8	-12.6
False										7	-12.3
True			100000	InvertedPendulum-v4					env	16	32.5
		2								8	53.7
		3								18	70.4
False		1								2	37.9
		2								10	68.0
		3								8	94.3
True		1							buffer	1	34.3
		2								12	53.5
		3								11	67.1
False		1								9	38.5
		2								5	68.0
		3								4	93.0
True		1							noise	3	34.1
		2								5	46.3
		3								6	46.6
False		1								8	39.1
		2								2	68.7
		3								1	94.3
True		1	300000						policy	6	46.2
False										5	71.7
True			200000	InvertedDoublePendulum-v4					env	7	216.7
		2								22	414.8
		3								21	520.8
False		1								10	231.6
		2								8	464.2
		3								15	675.6
True		1							buffer	9	218.3
		2								5	254.4
		3								5	379.5

False		1								10	237.0
		2								1	476.6
		3								3	702.2
True		1							noise	12	219.6
		2								14	438.0
		3								9	547.4
False		1								9	231.6
		2								7	467.5
		3								10	600.9
True		1	600000						policy	4	640.9
False										10	683.0
True			266667	Pusher-v4					env	10	-76.0
		2								6	-57.6
		3								8	-49.8
False		1								7	-74.5
		2								22	-56.5
		3								12	-48.8
True		1							buffer	6	-76.1
		2								12	-57.6
		3								5	-49.7
False		1								3	-74.8
		2								5	-56.6
		3								2	-48.9
True		1							noise	2	-75.6
		2								12	-57.2
		3								14	-49.4
False		1								7	-74.4
		2								10	-56.2
		3								5	-48.6
True		1	800000						policy	12	-49.3
False										5	-48.9
True			166667	Walker2d-v4					env	8	51.7
		2								21	119.3
		3								18	156.4
False		1								1	55.8
		2								4	138.9
		3								5	208.7
True		1							buffer	12	51.1
		2								1	124.1
		3								3	180.7
False		1								7	57.2
		2								1	144.8
		3								10	220.5
True		1							noise	7	48.9
		2								1	120.3
		3								11	173.2
False		1								5	56.3
		2								5	128.3
		3								3	183.2

True		1	500000						policy	10	156.5
False										1	201.2
True			50000	CarRacing-v2					env	16	-62.1
		2								15	-66.6
		3								2	-68.0
False		1								8	-54.3
		2								17	-49.1
		3								14	-49.2
True		1							buffer	12	-60.1
		2								12	-66.0
		3								11	-66.2
False		1								4	-55.3
		2								2	-52.0
		3								3	-51.6
True		1							noise	15	-70.2
		2								14	-71.6
		3								1	-70.7
False		1								11	-45.9
		2								9	-45.7
		3								12	-42.6
True		1	150000						policy	1	-63.3
False										7	-47.4
True			266667	HumanoidStandup-v4					env	11	62268.3
		2								9	70090.9
		3								6	73305.6
False		1								18	65482.0
		2								17	73268.7
		3								3	79143.4
True		1							buffer	6	63145.3
		2								9	70846.3
		3								9	74598.0
False		1								12	65475.9
		2								3	73556.7
		3								12	77920.8
True		1							noise	6	60008.5
		2								6	67144.7
		3								8	70564.3
False		1								15	65042.0
		2								11	74506.0
		3								11	79604.0
True		1	800000						policy	4	73620.9
False										5	78910.1
True			300000	Humanoid-v4					env	17	213.9
		2								8	266.7
		3								17	297.9
False		1								1	221.6
		2								15	275.8
		3								2	309.0
True		1							buffer	3	218.7

		2								2	270.6
		3								11	300.0
False		1								7	222.9
		2								9	277.2
		3								2	310.4
True		1							noise	4	217.6
		2								9	267.0
		3								5	297.2
False		1								8	226.0
		2								2	279.9
		3								2	310.4
True		1	900000						policy	3	298.6
False										11	306.3

Table 16: Final Results

The table is available in the file stored on GitLab [11].

B Complete SMAPE table

- **Complete SMAPE table calculated from reward logs recorded during training sessions:**

algorithm	environment	source	section	SMAPE (%)
SAC	Humanoid-v4	env	1	11.1
SAC	Humanoid-v4	env	2	10.2
SAC	Humanoid-v4	env	3	11.4
SAC	Humanoid-v4	env	all	10.9
SAC	Humanoid-v4	buffer	1	10.5
SAC	Humanoid-v4	buffer	2	10.8
SAC	Humanoid-v4	buffer	3	12.9
SAC	Humanoid-v4	buffer	all	11.1
SAC	Humanoid-v4	noise	1	9.9
SAC	Humanoid-v4	noise	2	10.0
SAC	Humanoid-v4	noise	3	12.3
SAC	Humanoid-v4	noise	all	10.4
SAC	Humanoid-v4	policy	1	10.7
SAC	Humanoid-v4	policy	2	12.1
SAC	Humanoid-v4	policy	3	13.1
SAC	Humanoid-v4	policy	all	11.6
SAC	HumanoidStandup-v4	env	1	13.7
SAC	HumanoidStandup-v4	env	2	22.9
SAC	HumanoidStandup-v4	env	3	21.6
SAC	HumanoidStandup-v4	env	all	19.5
SAC	HumanoidStandup-v4	buffer	1	16.9
SAC	HumanoidStandup-v4	buffer	2	21.4
SAC	HumanoidStandup-v4	buffer	3	16.6
SAC	HumanoidStandup-v4	buffer	all	18.3
SAC	HumanoidStandup-v4	noise	1	16.2
SAC	HumanoidStandup-v4	noise	2	20.7
SAC	HumanoidStandup-v4	noise	3	27.0
SAC	HumanoidStandup-v4	noise	all	21.4
SAC	HumanoidStandup-v4	policy	1	21.9
SAC	HumanoidStandup-v4	policy	2	16.1
SAC	HumanoidStandup-v4	policy	3	22.3
SAC	HumanoidStandup-v4	policy	all	20.1
SAC	Walker2d-v4	env	1	57.8
SAC	Walker2d-v4	env	2	39.0
SAC	Walker2d-v4	env	3	35.2
SAC	Walker2d-v4	env	all	46.4
SAC	Walker2d-v4	buffer	1	54.1
SAC	Walker2d-v4	buffer	2	44.2
SAC	Walker2d-v4	buffer	3	56.0
SAC	Walker2d-v4	buffer	all	51.3
SAC	Walker2d-v4	noise	1	58.6
SAC	Walker2d-v4	noise	2	37.7
SAC	Walker2d-v4	noise	3	39.4
SAC	Walker2d-v4	noise	all	47.5

SAC	Walker2d-v4	policy	1	53.0
SAC	Walker2d-v4	policy	2	34.7
SAC	Walker2d-v4	policy	3	29.7
SAC	Walker2d-v4	policy	all	41.0
SAC	Ant-v4	env	1	65.9
SAC	Ant-v4	env	2	31.3
SAC	Ant-v4	env	3	23.2
SAC	Ant-v4	env	all	49.4
SAC	Ant-v4	buffer	1	63.8
SAC	Ant-v4	buffer	2	36.6
SAC	Ant-v4	buffer	3	27.6
SAC	Ant-v4	buffer	all	50.0
SAC	Ant-v4	noise	1	45.7
SAC	Ant-v4	noise	2	33.3
SAC	Ant-v4	noise	3	16.3
SAC	Ant-v4	noise	all	36.9
SAC	Ant-v4	policy	1	54.1
SAC	Ant-v4	policy	2	33.4
SAC	Ant-v4	policy	3	21.5
SAC	Ant-v4	policy	all	40.7
PPO	Ant-v4	env	1	59.7
PPO	Ant-v4	env	2	46.8
PPO	Ant-v4	env	3	34.4
PPO	Ant-v4	env	all	52.1
PPO	Ant-v4	buffer	1	61.2
PPO	Ant-v4	buffer	2	48.7
PPO	Ant-v4	buffer	3	36.5
PPO	Ant-v4	buffer	all	53.7
PPO	Ant-v4	noise	1	62.8
PPO	Ant-v4	noise	2	47.6
PPO	Ant-v4	noise	3	30.9
PPO	Ant-v4	noise	all	54.0
PPO	Ant-v4	policy	all	54.6
PPO	BipedalWalker-v3	env	1	31.5
PPO	BipedalWalker-v3	env	2	92.2
PPO	BipedalWalker-v3	env	3	72.8
PPO	BipedalWalker-v3	env	all	60.9
PPO	BipedalWalker-v3	buffer	1	27.6
PPO	BipedalWalker-v3	buffer	2	79.9
PPO	BipedalWalker-v3	buffer	3	82.9
PPO	BipedalWalker-v3	buffer	all	58.6
PPO	BipedalWalker-v3	noise	1	37.0
PPO	BipedalWalker-v3	noise	2	83.1
PPO	BipedalWalker-v3	noise	3	89.7
PPO	BipedalWalker-v3	noise	all	69.7
PPO	BipedalWalker-v3	policy	all	63.3
SAC	BipedalWalker-v3	env	1	8.7
SAC	BipedalWalker-v3	env	2	9.4
SAC	BipedalWalker-v3	env	3	11.2

SAC	BipedalWalker-v3	env	all	9.9
SAC	BipedalWalker-v3	buffer	1	6.3
SAC	BipedalWalker-v3	buffer	2	21.3
SAC	BipedalWalker-v3	buffer	3	21.2
SAC	BipedalWalker-v3	buffer	all	13.0
SAC	BipedalWalker-v3	noise	1	6.8
SAC	BipedalWalker-v3	noise	2	7.3
SAC	BipedalWalker-v3	noise	3	10.3
SAC	BipedalWalker-v3	noise	all	7.8
SAC	BipedalWalker-v3	policy	1	5.3
SAC	BipedalWalker-v3	policy	2	14.4
SAC	BipedalWalker-v3	policy	3	10.7
SAC	BipedalWalker-v3	policy	all	8.6
PPO	BipedalWalkerHardcore-v3	env	1	9.6
PPO	BipedalWalkerHardcore-v3	env	2	10.3
PPO	BipedalWalkerHardcore-v3	env	3	11.6
PPO	BipedalWalkerHardcore-v3	env	all	10.6
PPO	BipedalWalkerHardcore-v3	buffer	1	9.6
PPO	BipedalWalkerHardcore-v3	buffer	2	12.0
PPO	BipedalWalkerHardcore-v3	buffer	3	12.7
PPO	BipedalWalkerHardcore-v3	buffer	all	11.4
PPO	BipedalWalkerHardcore-v3	noise	1	9.5
PPO	BipedalWalkerHardcore-v3	noise	2	10.3
PPO	BipedalWalkerHardcore-v3	noise	3	11.2
PPO	BipedalWalkerHardcore-v3	noise	all	10.5
PPO	BipedalWalkerHardcore-v3	policy	all	13.5
SAC	BipedalWalkerHardcore-v3	env	1	12.2
SAC	BipedalWalkerHardcore-v3	env	2	9.3
SAC	BipedalWalkerHardcore-v3	env	3	7.2
SAC	BipedalWalkerHardcore-v3	env	all	9.9
SAC	BipedalWalkerHardcore-v3	buffer	1	6.2
SAC	BipedalWalkerHardcore-v3	buffer	2	16.0
SAC	BipedalWalkerHardcore-v3	buffer	3	6.3
SAC	BipedalWalkerHardcore-v3	buffer	all	8.5
SAC	BipedalWalkerHardcore-v3	noise	1	6.1
SAC	BipedalWalkerHardcore-v3	noise	2	11.3
SAC	BipedalWalkerHardcore-v3	noise	3	8.6
SAC	BipedalWalkerHardcore-v3	noise	all	8.5
SAC	BipedalWalkerHardcore-v3	policy	1	9.8
SAC	BipedalWalkerHardcore-v3	policy	2	7.4
SAC	BipedalWalkerHardcore-v3	policy	3	12.1
SAC	BipedalWalkerHardcore-v3	policy	all	9.6
PPO	CarRacing-v2	env	1	10.7
PPO	CarRacing-v2	env	2	42.6
PPO	CarRacing-v2	env	3	42.1
PPO	CarRacing-v2	env	all	39.6
PPO	CarRacing-v2	buffer	1	8.1
PPO	CarRacing-v2	buffer	2	34.8
PPO	CarRacing-v2	buffer	3	40.5

PPO	CarRacing-v2	buffer	all	35.8
PPO	CarRacing-v2	noise	1	33.7
PPO	CarRacing-v2	noise	2	36.5
PPO	CarRacing-v2	noise	3	41.1
PPO	CarRacing-v2	noise	all	38.3
PPO	CarRacing-v2	policy	all	39.4
PPO	HalfCheetah-v4	env	1	29.9
PPO	HalfCheetah-v4	env	2	8.9
PPO	HalfCheetah-v4	env	3	3.6
PPO	HalfCheetah-v4	env	all	14.1
PPO	HalfCheetah-v4	buffer	1	36.8
PPO	HalfCheetah-v4	buffer	2	24.2
PPO	HalfCheetah-v4	buffer	3	6.5
PPO	HalfCheetah-v4	buffer	all	22.5
PPO	HalfCheetah-v4	noise	1	43.2
PPO	HalfCheetah-v4	noise	2	40.0
PPO	HalfCheetah-v4	noise	3	11.1
PPO	HalfCheetah-v4	noise	all	31.4
PPO	HalfCheetah-v4	policy	all	27.2
SAC	HalfCheetah-v4	env	1	44.8
SAC	HalfCheetah-v4	env	2	25.3
SAC	HalfCheetah-v4	env	3	33.7
SAC	HalfCheetah-v4	env	all	34.5
SAC	HalfCheetah-v4	buffer	1	28.2
SAC	HalfCheetah-v4	buffer	2	32.1
SAC	HalfCheetah-v4	buffer	3	28.1
SAC	HalfCheetah-v4	buffer	all	29.5
SAC	HalfCheetah-v4	noise	1	43.2
SAC	HalfCheetah-v4	noise	2	34.1
SAC	HalfCheetah-v4	noise	3	32.9
SAC	HalfCheetah-v4	noise	all	36.7
SAC	HalfCheetah-v4	policy	1	43.5
SAC	HalfCheetah-v4	policy	2	32.7
SAC	HalfCheetah-v4	policy	3	18.0
SAC	HalfCheetah-v4	policy	all	31.3
PPO	Hopper-v4	env	1	20.3
PPO	Hopper-v4	env	2	10.7
PPO	Hopper-v4	env	3	17.4
PPO	Hopper-v4	env	all	16.9
PPO	Hopper-v4	buffer	1	19.1
PPO	Hopper-v4	buffer	2	10.7
PPO	Hopper-v4	buffer	3	10.6
PPO	Hopper-v4	buffer	all	14.6
PPO	Hopper-v4	noise	1	28.3
PPO	Hopper-v4	noise	2	10.5
PPO	Hopper-v4	noise	3	8.0
PPO	Hopper-v4	noise	all	18.3
PPO	Hopper-v4	policy	all	16.9
SAC	Hopper-v4	env	1	24.0

SAC	Hopper-v4	env	2	31.7
SAC	Hopper-v4	env	3	34.5
SAC	Hopper-v4	env	all	29.9
SAC	Hopper-v4	buffer	1	20.2
SAC	Hopper-v4	buffer	2	29.5
SAC	Hopper-v4	buffer	3	28.7
SAC	Hopper-v4	buffer	all	26.1
SAC	Hopper-v4	noise	1	31.4
SAC	Hopper-v4	noise	2	36.5
SAC	Hopper-v4	noise	3	37.3
SAC	Hopper-v4	noise	all	35.1
SAC	Hopper-v4	policy	1	26.6
SAC	Hopper-v4	policy	2	24.7
SAC	Hopper-v4	policy	3	27.7
SAC	Hopper-v4	policy	all	26.4
PPO	Humanoid-v4	env	1	17.4
PPO	Humanoid-v4	env	2	13.6
PPO	Humanoid-v4	env	3	14.3
PPO	Humanoid-v4	env	all	15.4
PPO	Humanoid-v4	buffer	1	16.8
PPO	Humanoid-v4	buffer	2	13.0
PPO	Humanoid-v4	buffer	3	13.5
PPO	Humanoid-v4	buffer	all	14.8
PPO	Humanoid-v4	noise	1	18.9
PPO	Humanoid-v4	noise	2	13.8
PPO	Humanoid-v4	noise	3	15.0
PPO	Humanoid-v4	noise	all	16.3
PPO	Humanoid-v4	policy	all	14.9
PPO	HumanoidStandup-v4	env	1	5.1
PPO	HumanoidStandup-v4	env	2	4.9
PPO	HumanoidStandup-v4	env	3	8.6
PPO	HumanoidStandup-v4	env	all	6.2
PPO	HumanoidStandup-v4	buffer	1	5.0
PPO	HumanoidStandup-v4	buffer	2	4.7
PPO	HumanoidStandup-v4	buffer	3	7.7
PPO	HumanoidStandup-v4	buffer	all	5.8
PPO	HumanoidStandup-v4	noise	1	7.0
PPO	HumanoidStandup-v4	noise	2	7.1
PPO	HumanoidStandup-v4	noise	3	8.0
PPO	HumanoidStandup-v4	noise	all	7.4
PPO	HumanoidStandup-v4	policy	all	6.6
PPO	InvertedDoublePendulum-v4	env	1	18.3
PPO	InvertedDoublePendulum-v4	env	2	15.3
PPO	InvertedDoublePendulum-v4	env	3	2.5
PPO	InvertedDoublePendulum-v4	env	all	17.9
PPO	InvertedDoublePendulum-v4	buffer	1	17.2
PPO	InvertedDoublePendulum-v4	buffer	2	93.8
PPO	InvertedDoublePendulum-v4	buffer	3	13.1
PPO	InvertedDoublePendulum-v4	buffer	all	41.9

PPO	InvertedDoublePendulum-v4	noise	1	19.9
PPO	InvertedDoublePendulum-v4	noise	2	22.8
PPO	InvertedDoublePendulum-v4	noise	3	8.4
PPO	InvertedDoublePendulum-v4	noise	all	19.8
PPO	InvertedDoublePendulum-v4	policy	all	17.2
SAC	InvertedDoublePendulum-v4	env	1	16.3
SAC	InvertedDoublePendulum-v4	env	2	15.9
SAC	InvertedDoublePendulum-v4	env	3	15.6
SAC	InvertedDoublePendulum-v4	env	all	15.9
SAC	InvertedDoublePendulum-v4	buffer	1	15.7
SAC	InvertedDoublePendulum-v4	buffer	2	14.9
SAC	InvertedDoublePendulum-v4	buffer	3	14.6
SAC	InvertedDoublePendulum-v4	buffer	all	15.0
SAC	InvertedDoublePendulum-v4	noise	1	17.9
SAC	InvertedDoublePendulum-v4	noise	2	17.6
SAC	InvertedDoublePendulum-v4	noise	3	19.0
SAC	InvertedDoublePendulum-v4	noise	all	18.2
SAC	InvertedDoublePendulum-v4	policy	1	16.4
SAC	InvertedDoublePendulum-v4	policy	2	16.1
SAC	InvertedDoublePendulum-v4	policy	3	16.7
SAC	InvertedDoublePendulum-v4	policy	all	16.4
PPO	InvertedPendulum-v4	env	1	25.4
PPO	InvertedPendulum-v4	env	2	32.8
PPO	InvertedPendulum-v4	env	3	29.8
PPO	InvertedPendulum-v4	env	all	26.0
PPO	InvertedPendulum-v4	buffer	1	21.6
PPO	InvertedPendulum-v4	buffer	2	30.7
PPO	InvertedPendulum-v4	buffer	3	32.2
PPO	InvertedPendulum-v4	buffer	all	22.6
PPO	InvertedPendulum-v4	noise	1	32.4
PPO	InvertedPendulum-v4	noise	2	35.4
PPO	InvertedPendulum-v4	noise	3	32.0
PPO	InvertedPendulum-v4	noise	all	32.5
PPO	InvertedPendulum-v4	policy	all	23.8
SAC	InvertedPendulum-v4	env	1	18.5
SAC	InvertedPendulum-v4	env	2	22.7
SAC	InvertedPendulum-v4	env	3	28.3
SAC	InvertedPendulum-v4	env	all	21.9
SAC	InvertedPendulum-v4	buffer	1	17.6
SAC	InvertedPendulum-v4	buffer	2	22.4
SAC	InvertedPendulum-v4	buffer	3	30.1
SAC	InvertedPendulum-v4	buffer	all	22.3
SAC	InvertedPendulum-v4	noise	1	26.9
SAC	InvertedPendulum-v4	noise	2	30.9
SAC	InvertedPendulum-v4	noise	3	25.5
SAC	InvertedPendulum-v4	noise	all	27.6
SAC	InvertedPendulum-v4	policy	1	19.7
SAC	InvertedPendulum-v4	policy	2	24.7
SAC	InvertedPendulum-v4	policy	3	29.4

SAC	InvertedPendulum-v4	policy	all	23.4
PPO	LunarLanderContinuous-v2	env	1	43.2
PPO	LunarLanderContinuous-v2	env	2	68.0
PPO	LunarLanderContinuous-v2	env	3	83.2
PPO	LunarLanderContinuous-v2	env	all	53.6
PPO	LunarLanderContinuous-v2	buffer	1	39.0
PPO	LunarLanderContinuous-v2	buffer	2	65.5
PPO	LunarLanderContinuous-v2	buffer	3	70.3
PPO	LunarLanderContinuous-v2	buffer	all	47.3
PPO	LunarLanderContinuous-v2	noise	1	46.3
PPO	LunarLanderContinuous-v2	noise	2	68.5
PPO	LunarLanderContinuous-v2	noise	3	79.1
PPO	LunarLanderContinuous-v2	noise	all	54.9
PPO	LunarLanderContinuous-v2	policy	all	48.9
SAC	LunarLanderContinuous-v2	env	1	58.5
SAC	LunarLanderContinuous-v2	env	2	66.5
SAC	LunarLanderContinuous-v2	env	3	71.8
SAC	LunarLanderContinuous-v2	env	all	65.8
SAC	LunarLanderContinuous-v2	buffer	1	71.7
SAC	LunarLanderContinuous-v2	buffer	2	67.0
SAC	LunarLanderContinuous-v2	buffer	3	67.3
SAC	LunarLanderContinuous-v2	buffer	all	68.7
SAC	LunarLanderContinuous-v2	noise	1	71.5
SAC	LunarLanderContinuous-v2	noise	2	75.9
SAC	LunarLanderContinuous-v2	noise	3	79.0
SAC	LunarLanderContinuous-v2	noise	all	75.6
SAC	LunarLanderContinuous-v2	policy	1	68.5
SAC	LunarLanderContinuous-v2	policy	2	76.3
SAC	LunarLanderContinuous-v2	policy	3	62.2
SAC	LunarLanderContinuous-v2	policy	all	68.7
PPO	MountainCarContinuous-v0	env	1	26.9
PPO	MountainCarContinuous-v0	env	2	6.7
PPO	MountainCarContinuous-v0	env	3	2.3
PPO	MountainCarContinuous-v0	env	all	8.0
PPO	MountainCarContinuous-v0	buffer	1	23.8
PPO	MountainCarContinuous-v0	buffer	2	7.1
PPO	MountainCarContinuous-v0	buffer	3	2.4
PPO	MountainCarContinuous-v0	buffer	all	6.9
PPO	MountainCarContinuous-v0	noise	1	67.2
PPO	MountainCarContinuous-v0	noise	2	25.3
PPO	MountainCarContinuous-v0	noise	3	2.4
PPO	MountainCarContinuous-v0	noise	all	19.4
PPO	MountainCarContinuous-v0	policy	all	8.1
SAC	MountainCarContinuous-v0	env	1	71.7
SAC	MountainCarContinuous-v0	env	2	65.2
SAC	MountainCarContinuous-v0	env	3	77.4
SAC	MountainCarContinuous-v0	env	all	72.0
SAC	MountainCarContinuous-v0	buffer	1	36.4
SAC	MountainCarContinuous-v0	buffer	2	11.9

SAC	MountainCarContinuous-v0	buffer	3	9.4
SAC	MountainCarContinuous-v0	buffer	all	16.3
SAC	MountainCarContinuous-v0	noise	1	88.9
SAC	MountainCarContinuous-v0	noise	2	96.1
SAC	MountainCarContinuous-v0	noise	3	89.4
SAC	MountainCarContinuous-v0	noise	all	91.6
SAC	MountainCarContinuous-v0	policy	1	19.8
SAC	MountainCarContinuous-v0	policy	2	45.6
SAC	MountainCarContinuous-v0	policy	3	6.8
SAC	MountainCarContinuous-v0	policy	all	22.2
PPO	Pendulum-v1	env	1	8.6
PPO	Pendulum-v1	env	2	26.1
PPO	Pendulum-v1	env	3	43.0
PPO	Pendulum-v1	env	all	25.9
PPO	Pendulum-v1	buffer	1	2.9
PPO	Pendulum-v1	buffer	2	13.5
PPO	Pendulum-v1	buffer	3	7.8
PPO	Pendulum-v1	buffer	all	8.1
PPO	Pendulum-v1	noise	1	5.4
PPO	Pendulum-v1	noise	2	19.1
PPO	Pendulum-v1	noise	3	10.8
PPO	Pendulum-v1	noise	all	11.8
PPO	Pendulum-v1	policy	all	7.2
SAC	Pendulum-v1	env	1	32.3
SAC	Pendulum-v1	env	2	29.9
SAC	Pendulum-v1	env	3	29.1
SAC	Pendulum-v1	env	all	30.4
SAC	Pendulum-v1	buffer	1	11.8
SAC	Pendulum-v1	buffer	2	10.6
SAC	Pendulum-v1	buffer	3	6.1
SAC	Pendulum-v1	buffer	all	9.5
SAC	Pendulum-v1	noise	1	18.5
SAC	Pendulum-v1	noise	2	11.6
SAC	Pendulum-v1	noise	3	11.8
SAC	Pendulum-v1	noise	all	13.9
SAC	Pendulum-v1	policy	1	16.0
SAC	Pendulum-v1	policy	2	9.1
SAC	Pendulum-v1	policy	3	7.4
SAC	Pendulum-v1	policy	all	10.8
PPO	Pusher-v4	env	1	3.8
PPO	Pusher-v4	env	2	6.6
PPO	Pusher-v4	env	3	7.7
PPO	Pusher-v4	env	all	6.0
PPO	Pusher-v4	buffer	1	1.2
PPO	Pusher-v4	buffer	2	1.1
PPO	Pusher-v4	buffer	3	0.6
PPO	Pusher-v4	buffer	all	1.0
PPO	Pusher-v4	noise	1	3.2
PPO	Pusher-v4	noise	2	1.5

PPO	Pusher-v4	noise	3	0.6
PPO	Pusher-v4	noise	all	1.8
PPO	Pusher-v4	policy	all	0.6
SAC	Pusher-v4	env	1	2.6
SAC	Pusher-v4	env	2	6.6
SAC	Pusher-v4	env	3	7.9
SAC	Pusher-v4	env	all	5.7
SAC	Pusher-v4	buffer	1	0.7
SAC	Pusher-v4	buffer	2	1.8
SAC	Pusher-v4	buffer	3	2.7
SAC	Pusher-v4	buffer	all	1.7
SAC	Pusher-v4	noise	1	3.0
SAC	Pusher-v4	noise	2	3.1
SAC	Pusher-v4	noise	3	3.8
SAC	Pusher-v4	noise	all	3.3
SAC	Pusher-v4	policy	1	1.2
SAC	Pusher-v4	policy	2	2.5
SAC	Pusher-v4	policy	3	3.7
SAC	Pusher-v4	policy	all	2.5
PPO	Reacher-v4	env	1	7.4
PPO	Reacher-v4	env	2	23.0
PPO	Reacher-v4	env	3	26.0
PPO	Reacher-v4	env	all	18.8
PPO	Reacher-v4	buffer	1	2.1
PPO	Reacher-v4	buffer	2	6.8
PPO	Reacher-v4	buffer	3	3.2
PPO	Reacher-v4	buffer	all	4.0
PPO	Reacher-v4	noise	1	6.5
PPO	Reacher-v4	noise	2	7.2
PPO	Reacher-v4	noise	3	3.8
PPO	Reacher-v4	noise	all	5.8
PPO	Reacher-v4	policy	all	3.2
SAC	Reacher-v4	env	1	7.0
SAC	Reacher-v4	env	2	17.8
SAC	Reacher-v4	env	3	19.9
SAC	Reacher-v4	env	all	14.9
SAC	Reacher-v4	buffer	1	1.9
SAC	Reacher-v4	buffer	2	5.7
SAC	Reacher-v4	buffer	3	5.4
SAC	Reacher-v4	buffer	all	4.3
SAC	Reacher-v4	noise	1	6.7
SAC	Reacher-v4	noise	2	8.2
SAC	Reacher-v4	noise	3	7.4
SAC	Reacher-v4	noise	all	7.4
SAC	Reacher-v4	policy	1	3.1
SAC	Reacher-v4	policy	2	5.5
SAC	Reacher-v4	policy	3	6.5
SAC	Reacher-v4	policy	all	5.1
PPO	Walker2d-v4	env	1	45.4

PPO	Walker2d-v4	env	2	16.8
PPO	Walker2d-v4	env	3	29.4
PPO	Walker2d-v4	env	all	34.4
PPO	Walker2d-v4	buffer	1	42.4
PPO	Walker2d-v4	buffer	2	17.9
PPO	Walker2d-v4	buffer	3	19.3
PPO	Walker2d-v4	buffer	all	31.3
PPO	Walker2d-v4	noise	1	63.0
PPO	Walker2d-v4	noise	2	10.0
PPO	Walker2d-v4	noise	3	10.0
PPO	Walker2d-v4	noise	all	38.8
PPO	Walker2d-v4	policy	all	34.0

Table 17: Final SMAPE

The table is available in the file stored on GitLab [11].

C Graphs

In this section, the results of the *SuperOracle* and *AntiSuperOracle* will be presented. For each algorithm and environment pair, the following will be attached:

1. **SuperOracle & AntiSuperOracle graph**

   For every source of randomness, graph will display two curves: the pink one represents the *SuperOracle*, and the blue one represents the *AntiSuperOracle*. The areas where they differ will be highlighted in light blue. These graphs are based on reward logs collected during training.

   The graph will also show divisions into sections and indicate which seed was chosen for the best and the worst performance in each section.

   Due to potential visibility issues, information about rewards, seed choice, and other details can be found in Table 16 (see Appendix A), where it was originally sourced for the graph.

2. **SMAPE table**

   To quantify the impact of each source per section in percentage terms, *SMAPE (Symmetric Mean Absolute Percentage Error)* is used to calculate the difference between the best and worst performance curves.

C.1 SAC

C.1.1 Pendulum-v1

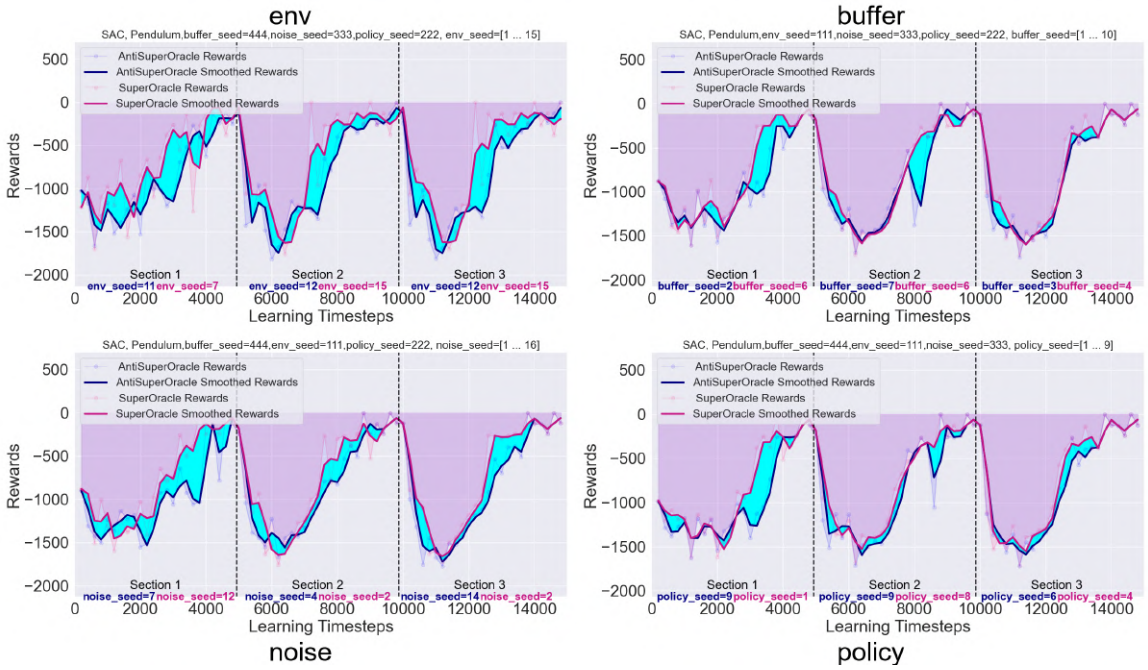


Figure 19: Combined Graph for Pendulum, SAC

Section	env	buffer	noise	policy
1	32.3%	11.8%	18.5%	16.0%
2	29.9%	10.6%	11.6%	9.1%
3	29.1%	6.1%	11.8%	7.4%
all	30.4%	9.5%	13.9%	10.8%

Table 18: SMAPE for Pendulum, SAC

C.1.2 MountainCarContinuous-v0

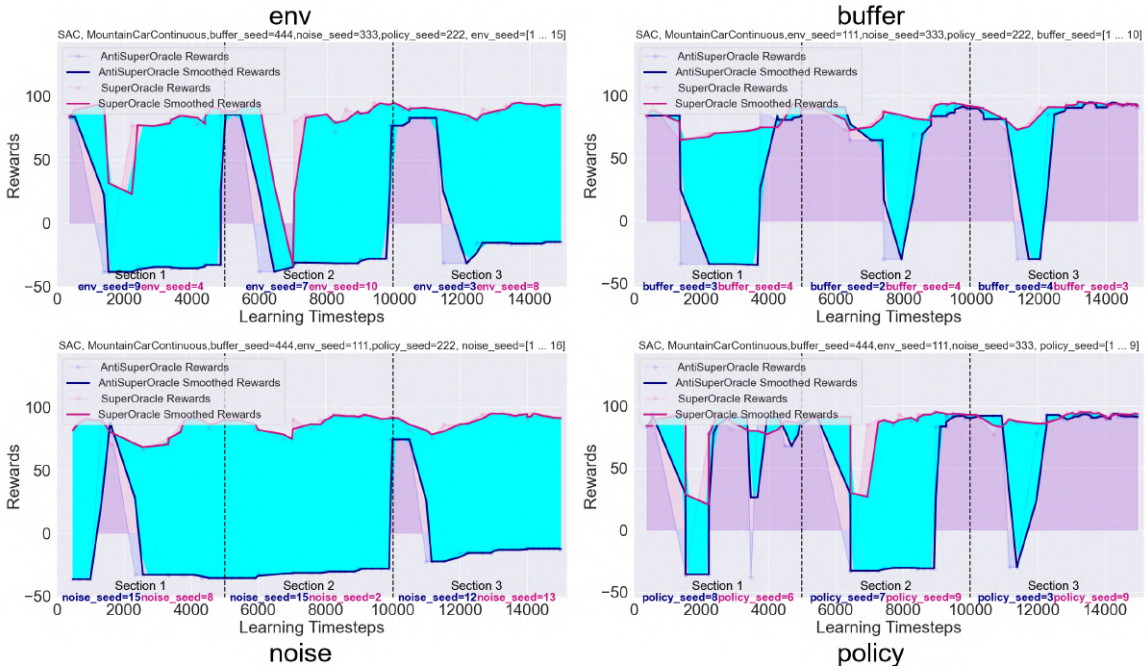


Figure 20: Combined Graph for MountainCarContinuous, SAC

Section	env	buffer	noise	policy
1	71.7%	36.4%	88.9%	19.8%
2	65.2%	11.9%	96.1%	45.6%
3	77.4%	9.4%	89.4%	6.8%
all	72.0%	16.3%	91.6%	22.2%

Table 19: SMAPE for MountainCarContinuous, SAC

C.1.3 BipedalWalker-v3

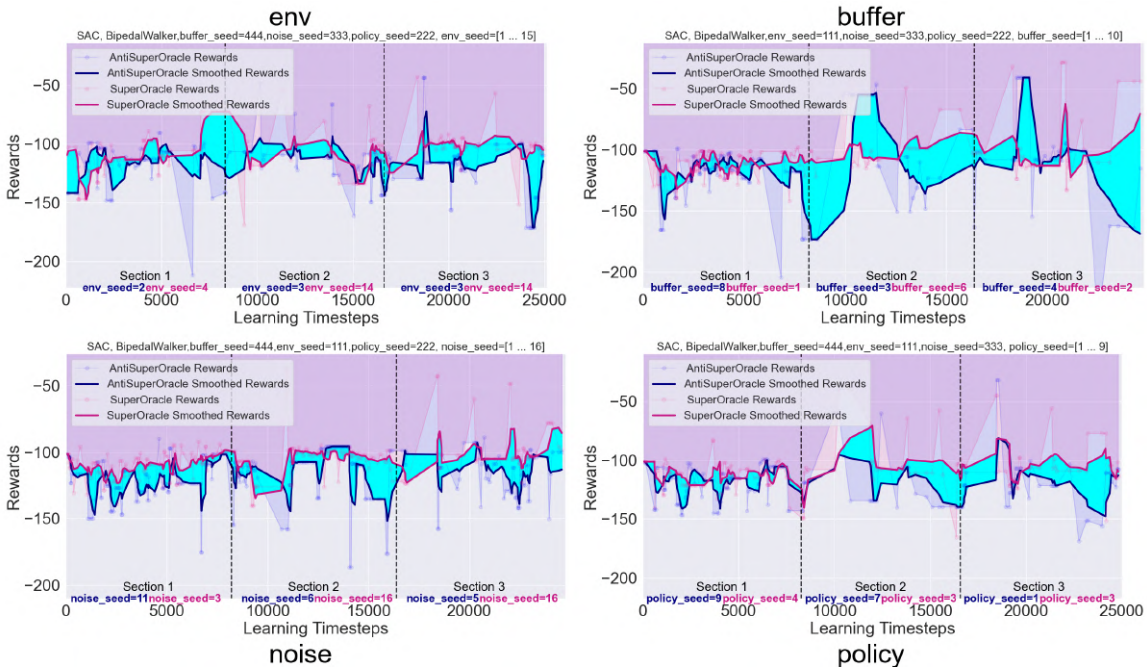


Figure 21: Combined Graph for BipedalWalker, SAC

Section	env	buffer	noise	policy
1	8.7%	6.3%	6.8%	5.3%
2	9.4%	21.3%	7.3%	14.4%
3	11.2%	21.2%	10.3%	10.7%
all	9.9%	13.0%	7.8%	8.6%

Table 20: SMAPE for BipedalWalker, SAC

C.1.4 BipedalWalkerHardcore-v3

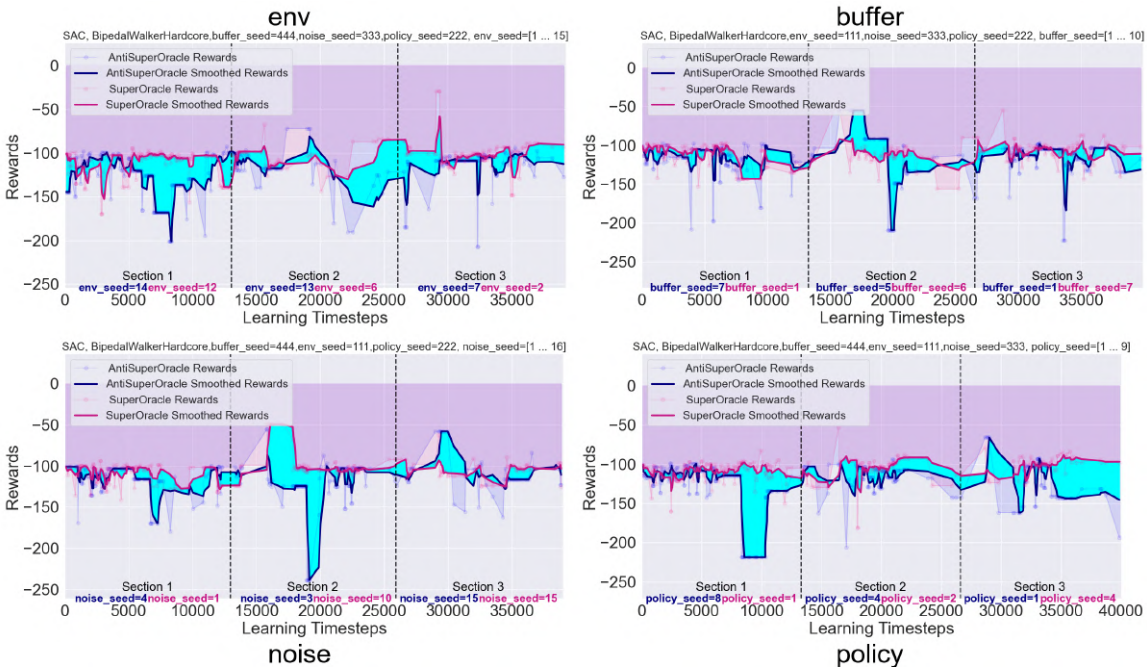


Figure 22: Combined Graph for BipedalWalkerHardcore, SAC

Section	env	buffer	noise	policy
1	12.2%	6.2%	6.1%	9.8%
2	9.3%	16.0%	11.3%	7.4%
3	7.2%	6.3%	8.6%	12.1%
all	9.9%	8.5%	8.5%	9.6%

Table 21: SMAPE for BipedalWalkerHardcore, SAC

C.1.5 LunarLanderContinuous-v2

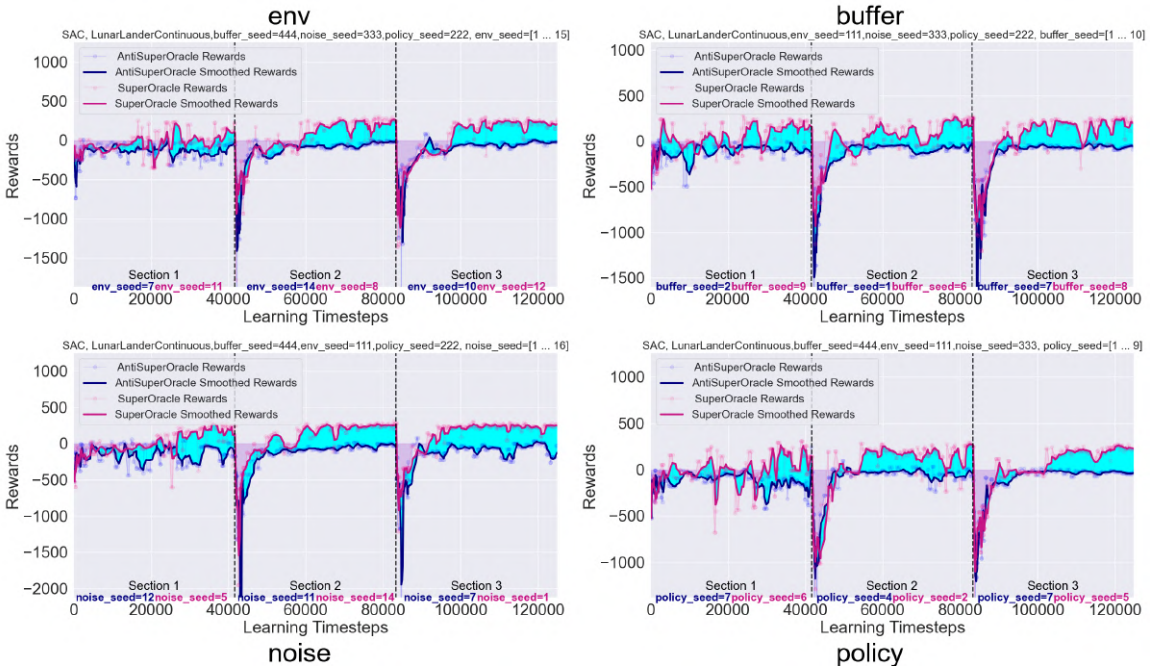


Figure 23: Combined Graph for LunarLanderContinuous, SAC

Section	env	buffer	noise	policy
1	58.5%	71.7%	71.5%	68.5%
2	66.5%	67.0%	75.9%	76.3%
3	71.8%	67.3%	79.0%	62.2%
all	65.8%	68.7%	75.6%	68.7%

Table 22: SMAPE for LunarLanderContinuous, SAC

C.1.6 Ant-v4

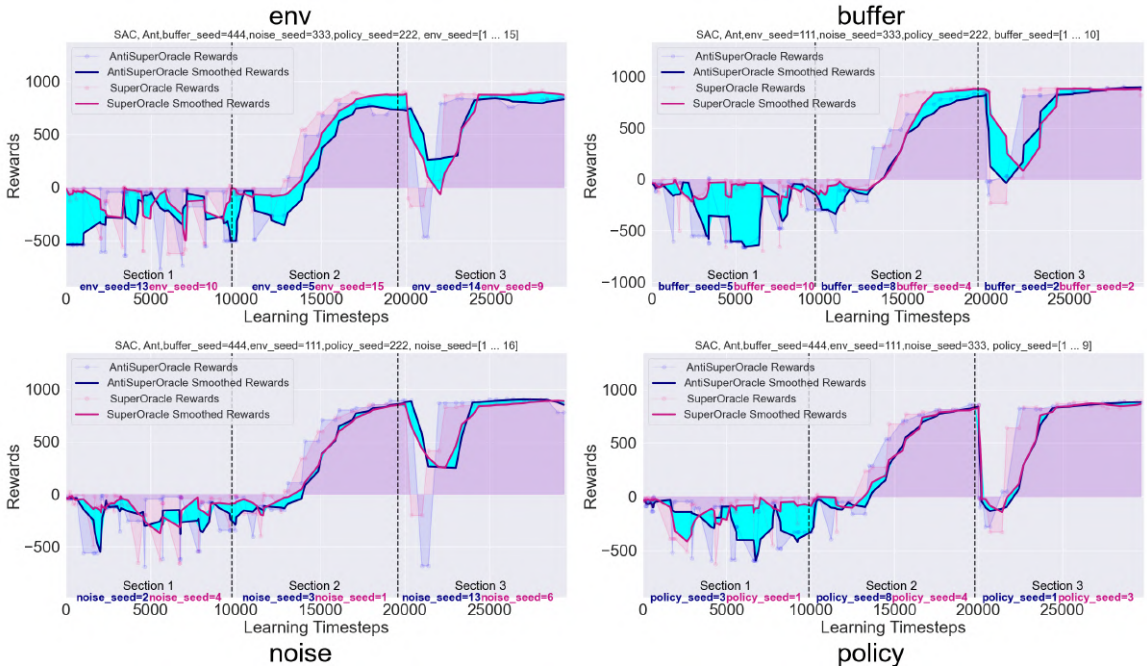


Figure 24: Combined Graph for Ant, SAC

Section	env	buffer	noise	policy
1	65.9%	63.8%	45.7%	54.1%
2	31.3%	36.6%	33.3%	33.4%
3	23.2%	27.6%	16.3%	21.5%
all	49.4%	50.0%	36.9%	40.7%

Table 23: SMAPE for Ant, SAC

C.1.7 HalfCheetah-v4

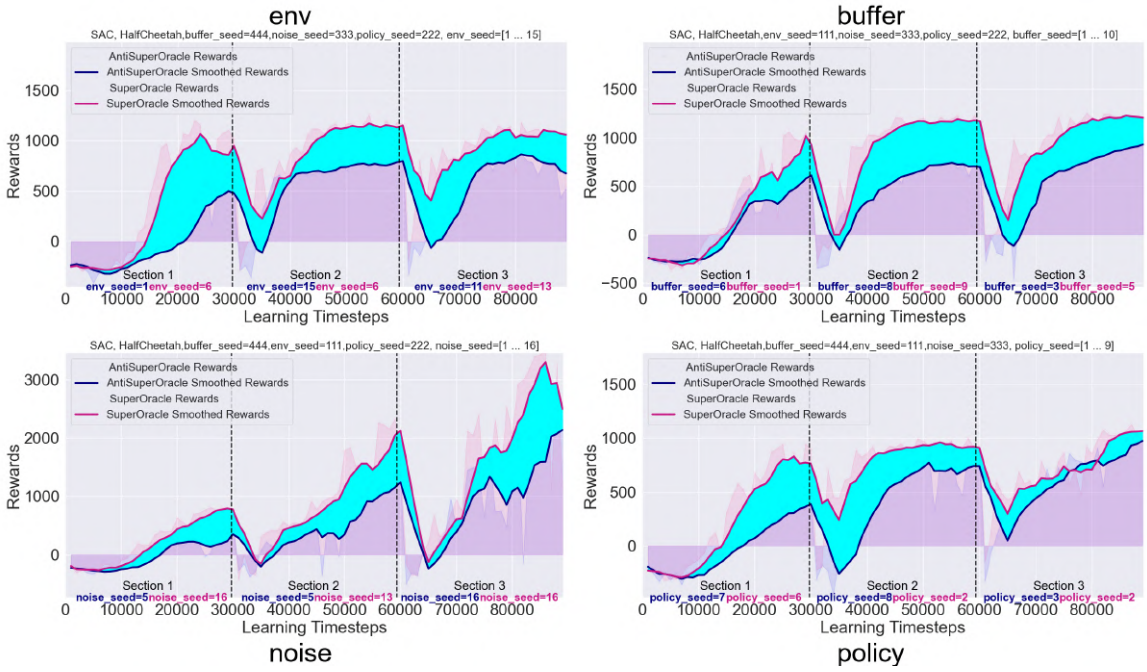


Figure 25: Combined Graph for HalfCheetah, SAC

Section	env	buffer	noise	policy
1	44.8%	28.2%	43.2%	43.5%
2	25.3%	32.1%	34.1%	32.7%
3	33.7%	28.1%	32.9%	18.0%
all	34.5%	29.5%	36.7%	31.3%

Table 24: SMAPE for HalfCheetah, SAC

C.1.8 Hopper-v4

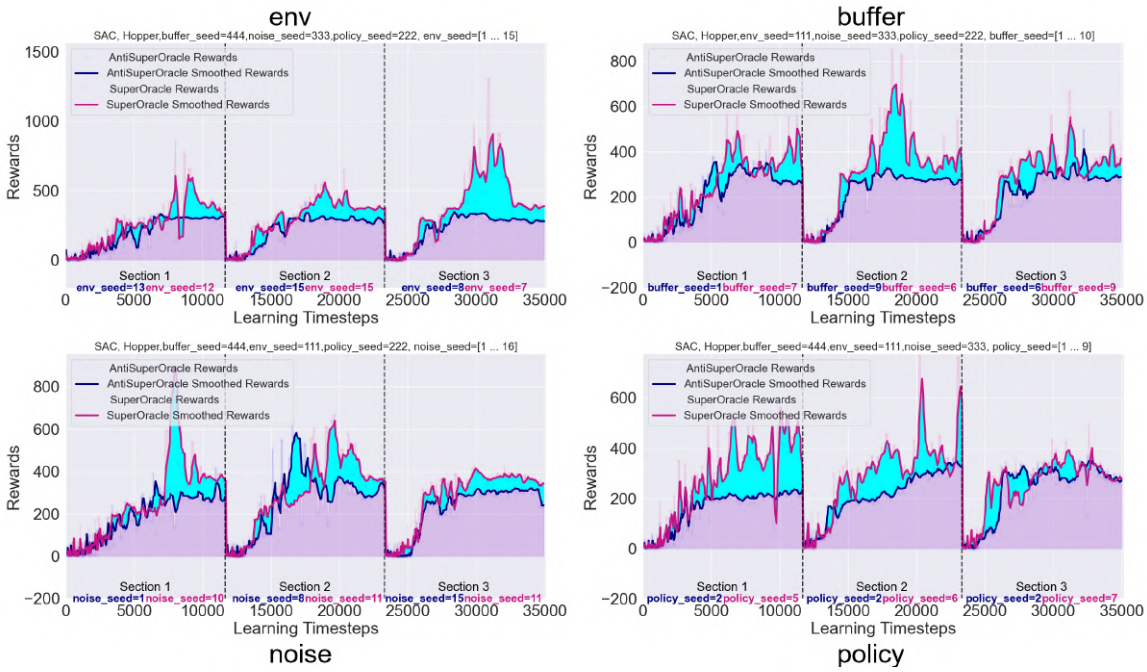


Figure 26: Combined Graph for Hopper, SAC

Section	env	buffer	noise	policy
1	24.0%	20.2%	31.4%	26.6%
2	31.7%	29.5%	36.5%	24.7%
3	34.5%	28.7%	37.3%	27.7%
all	29.9%	26.1%	35.1%	26.4%

Table 25: SMAPE for Hopper, SAC

C.1.9 Reacher-v4

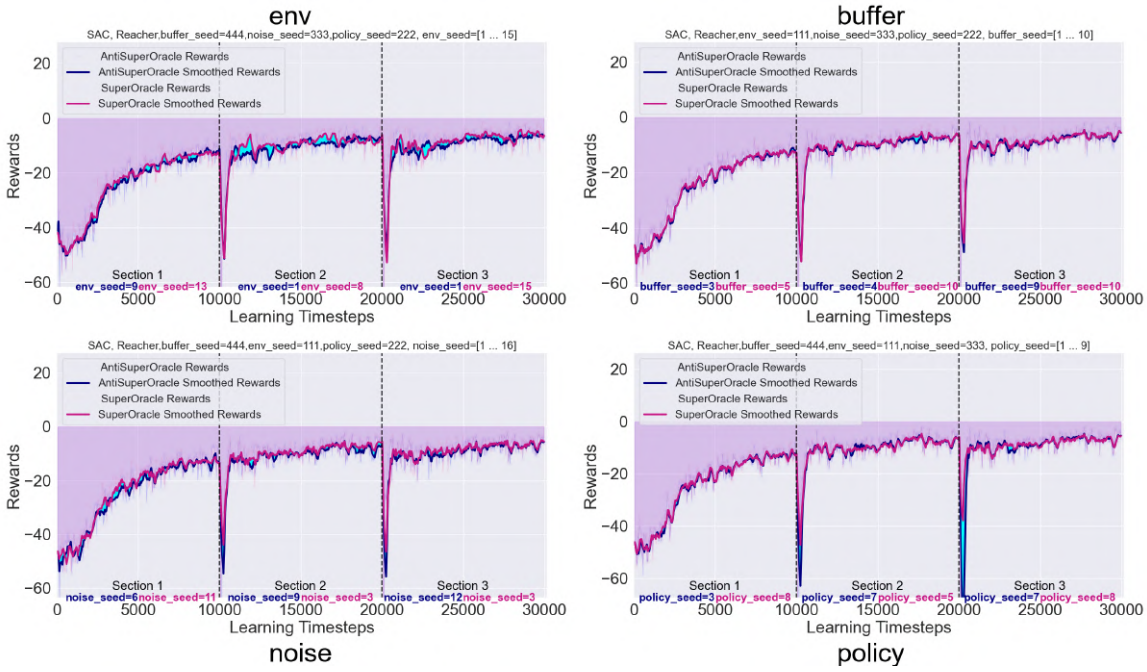


Figure 27: Combined Graph for Reacher, SAC

Section	env	buffer	noise	policy
1	7.0%	1.9%	6.7%	3.1%
2	17.8%	5.7%	8.2%	5.5%
3	19.9%	5.4%	7.4%	6.5%
all	14.9%	4.3%	7.4%	5.1%

Table 26: SMAPE for Reacher, SAC

C.1.10 InvertedPendulum-v4

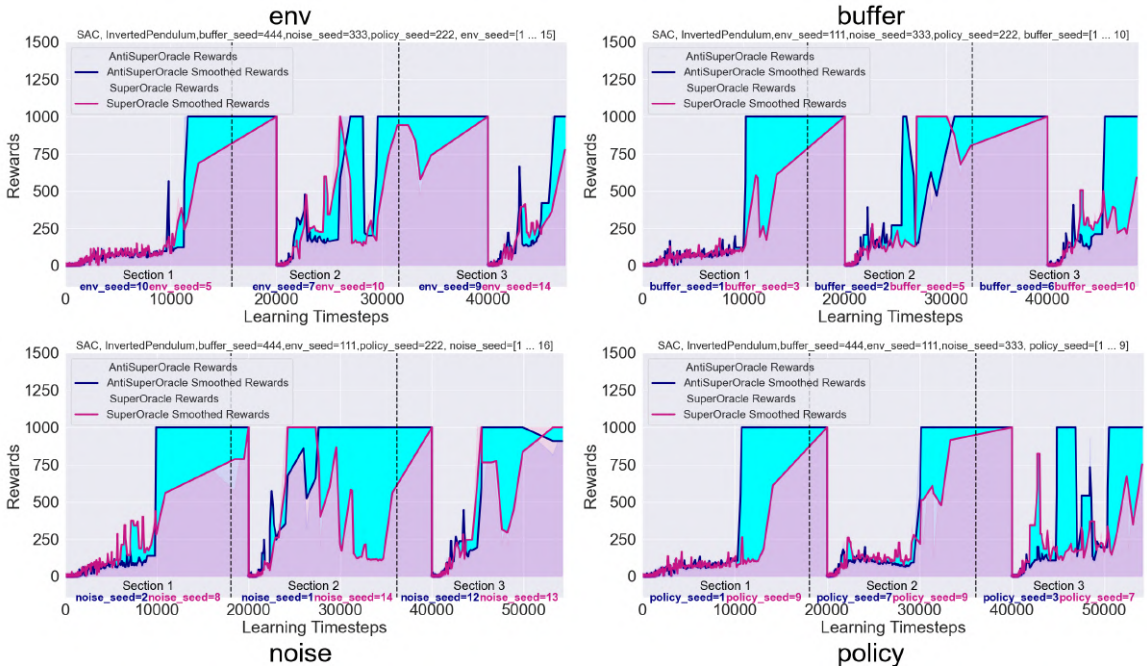


Figure 28: Combined Graph for InvertedPendulum, SAC

Section	env	buffer	noise	policy
1	18.5%	17.6%	26.9%	19.7%
2	22.7%	22.4%	30.9%	24.7%
3	28.3%	30.1%	25.5%	29.4%
all	21.9%	22.3%	27.6%	23.4%

Table 27: SMAPE for InvertedPendulum, SAC

C.1.11 InvertedDoublePendulum-v4

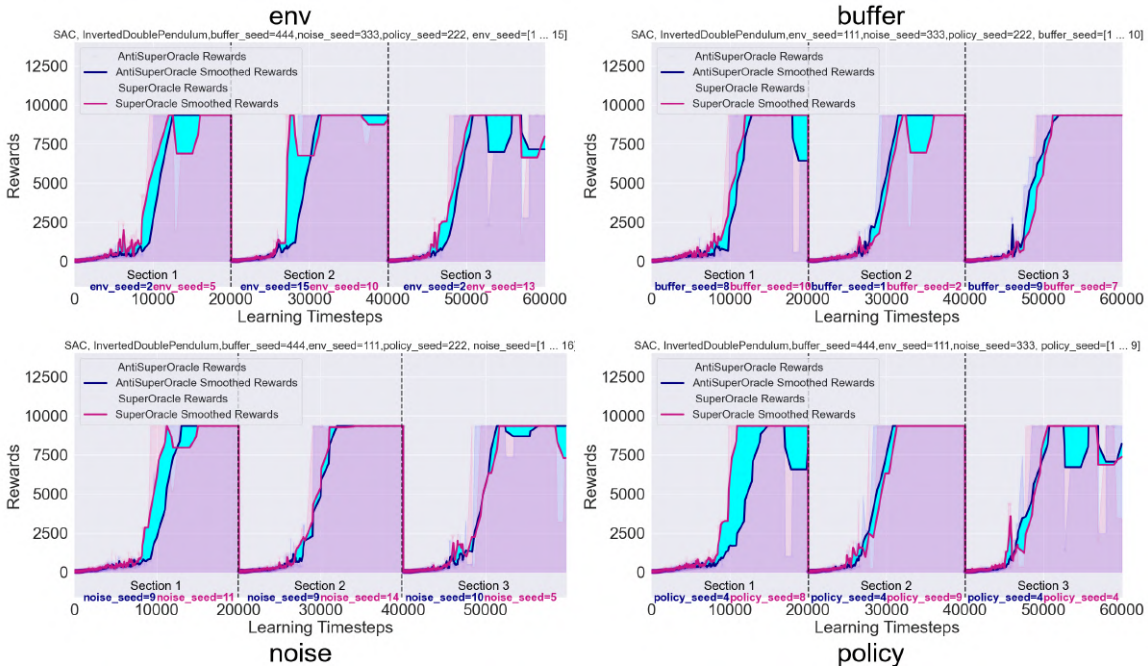


Figure 29: Combined Graph for InvertedDoublePendulum, SAC

Section	env	buffer	noise	policy
1	16.3%	15.7%	17.9%	16.4%
2	15.9%	14.9%	17.6%	16.1%
3	15.6%	14.6%	19.0%	16.7%
all	15.9%	15.0%	18.2%	16.4%

Table 28: SMAPE for InvertedDoublePendulum, SAC

C.1.12 Pusher-v4

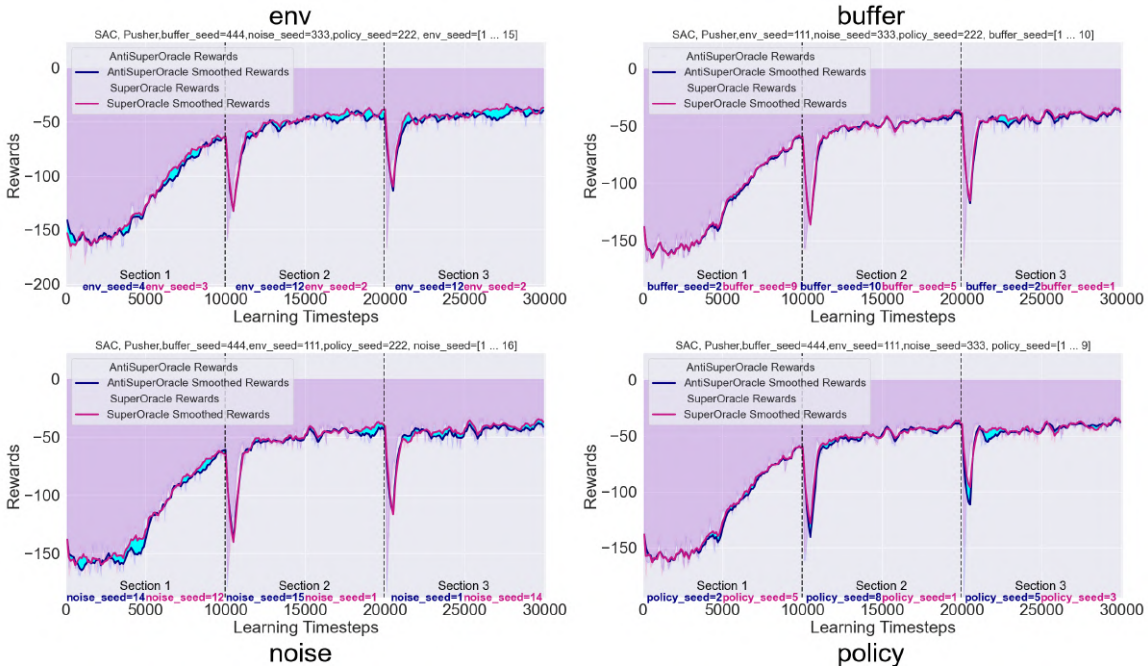


Figure 30: Combined Graph for Pusher, SAC

Section	env	buffer	noise	policy
1	2.6%	0.7%	3.0%	1.2%
2	6.6%	1.8%	3.1%	2.5%
3	7.9%	2.7%	3.8%	3.7%
all	5.7%	1.7%	3.3%	2.5%

Table 29: SMAPE for Pusher, SAC

C.1.13 Walker2d-v4

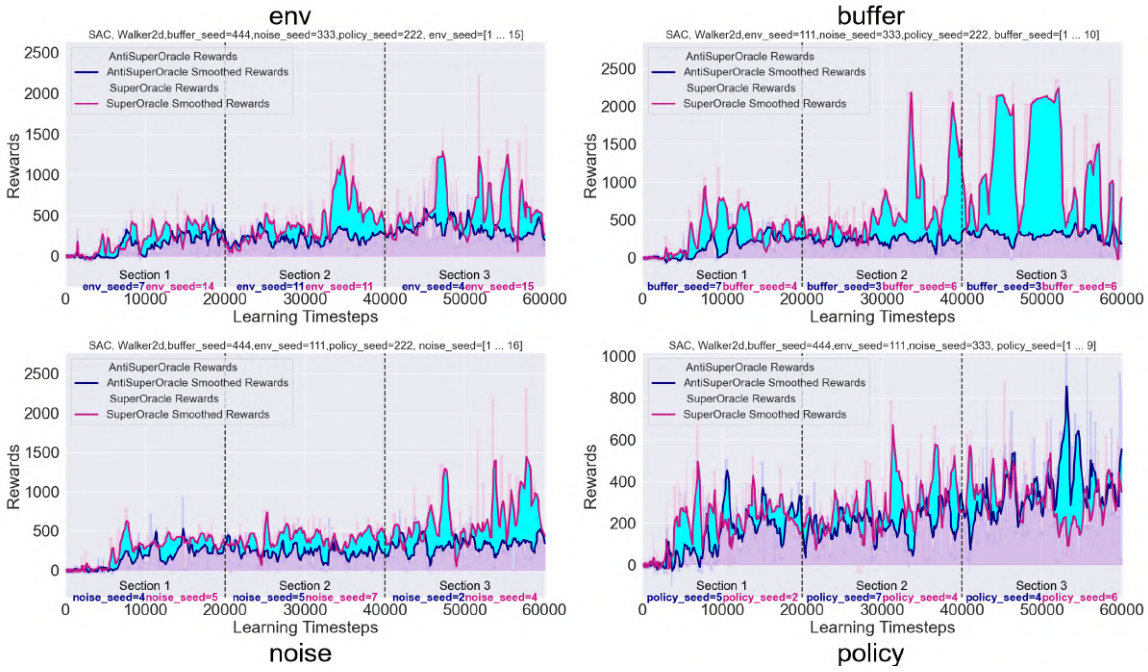


Figure 31: Combined Graph for Walker2d, SAC

Section	env	buffer	noise	policy
1	57.8%	54.1%	58.6%	53.0%
2	39.0%	44.2%	37.7%	34.7%
3	35.2%	56.0%	39.4%	29.7%
all	46.4%	51.3%	47.5%	41.0%

Table 30: SMAPE for Walker2d, SAC

C.1.14 HumanoidStandup-v4

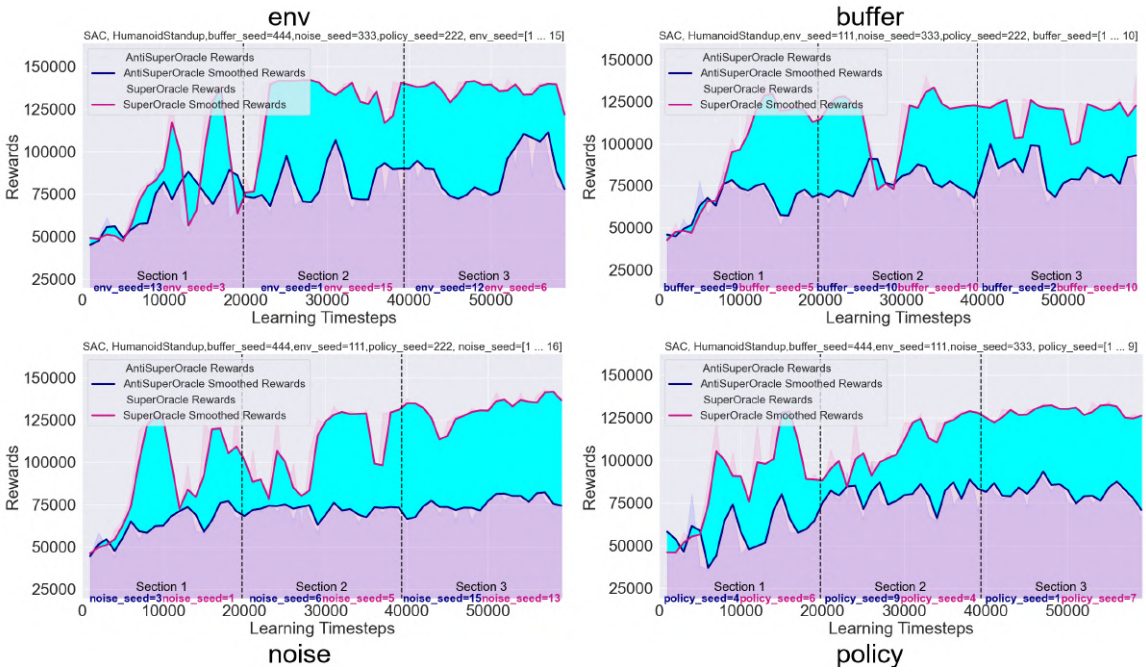


Figure 32: Combined Graph for HumanoidStandup, SAC

Section	env	buffer	noise	policy
1	13.7%	16.9%	16.2%	21.9%
2	22.9%	21.4%	20.7%	16.1%
3	21.6%	16.6%	27.0%	22.3%
all	19.5%	18.3%	21.4%	20.1%

Table 31: SMAPE for HumanoidStandup, SAC

C.1.15 Humanoid-v4

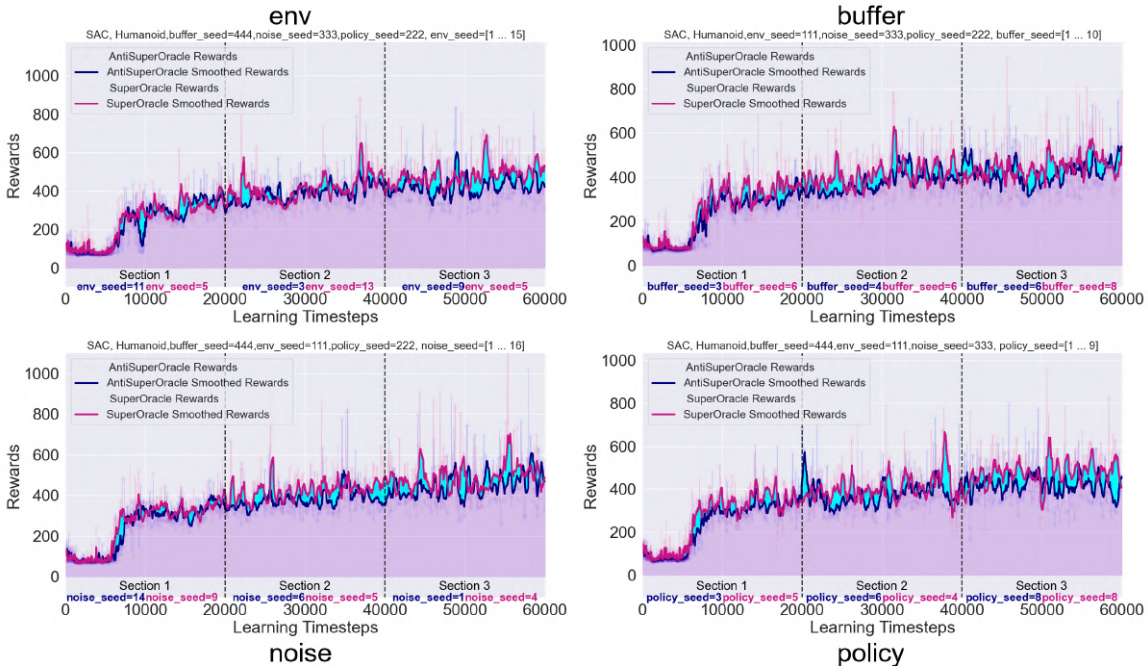


Figure 33: Combined Graph for Humanoid, SAC

Section	env	buffer	noise	policy
1	11.1%	10.5%	9.9%	10.7%
2	10.2%	10.8%	10.0%	12.1%
3	11.4%	12.9%	12.3%	13.1%
all	10.9%	11.1%	10.4%	11.6%

Table 32: SMAPE for Humanoid, SAC

C.2 PPO

C.2.1 Pendulum-v1

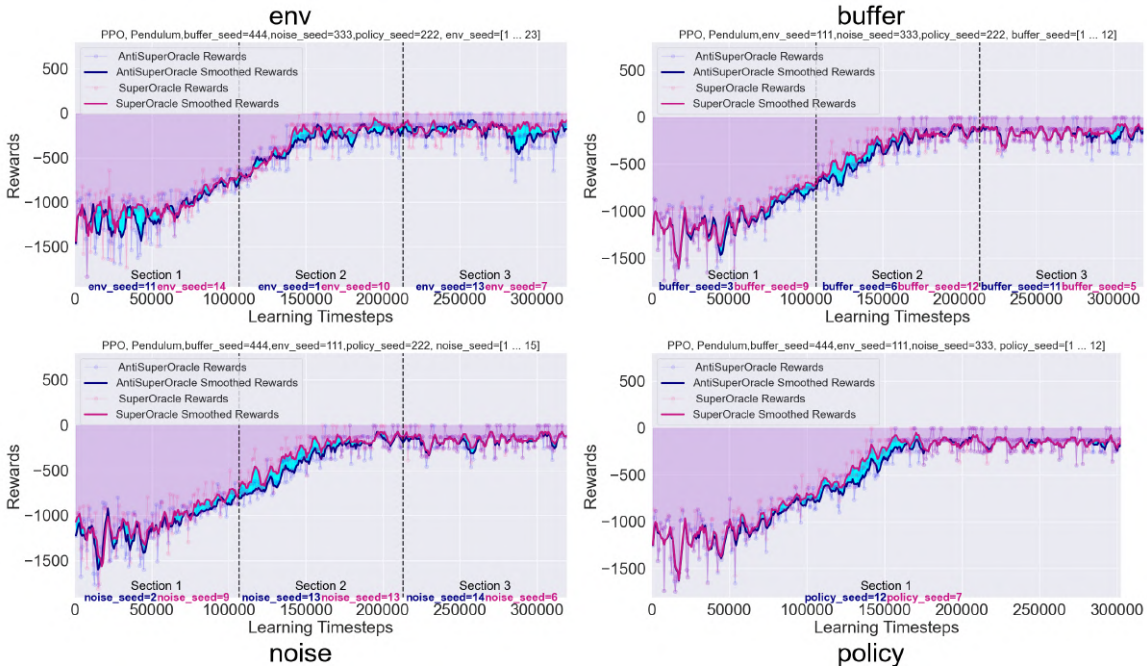


Figure 34: Combined Graph for Pendulum, PPO

Section	env	buffer	noise	policy
1	8.6%	2.9%	5.4%	-
2	26.1%	13.5%	19.1%	-
3	43.0%	7.8%	10.8%	-
all	25.9%	8.1%	11.8%	7.2%

Table 33: SMAPE for Pendulum, PPO

C.2.2 MountainCarContinuous-v0

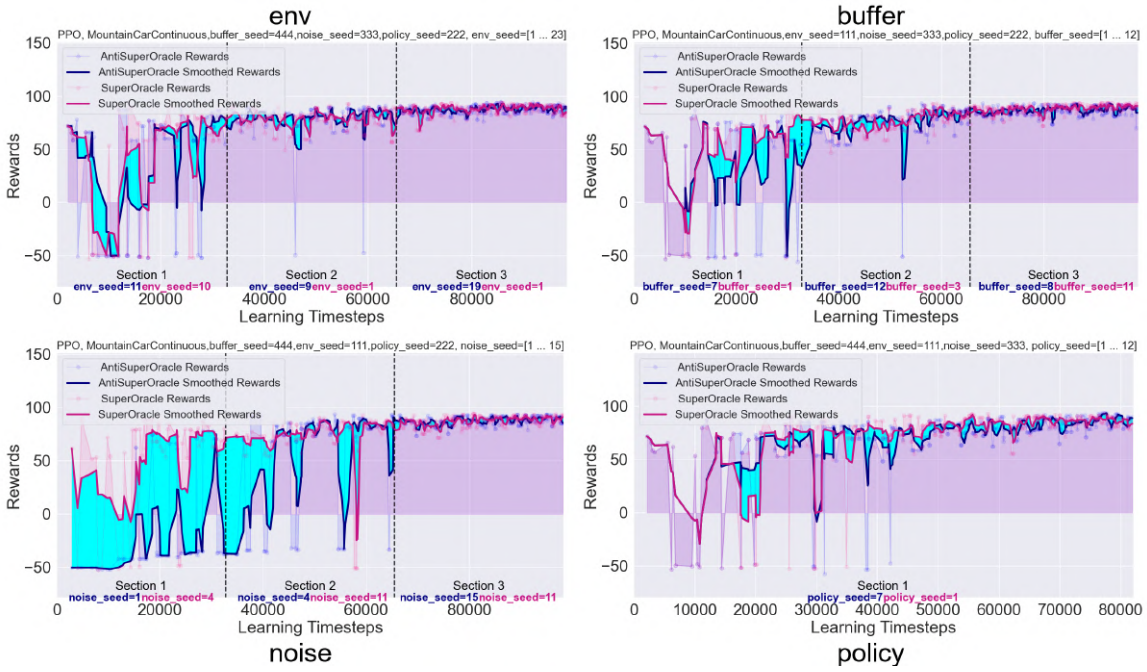


Figure 35: Combined Graph for MountainCarContinuous, PPO

Section	env	buffer	noise	policy
1	26.9%	23.8%	67.2%	-
2	6.7%	7.1%	25.3%	-
3	2.3%	2.4%	2.4%	-
all	8.0%	6.9%	19.4%	8.1%

Table 34: SMAPE for MountainCarContinuous, PPO

C.2.3 BipedalWalker-v3

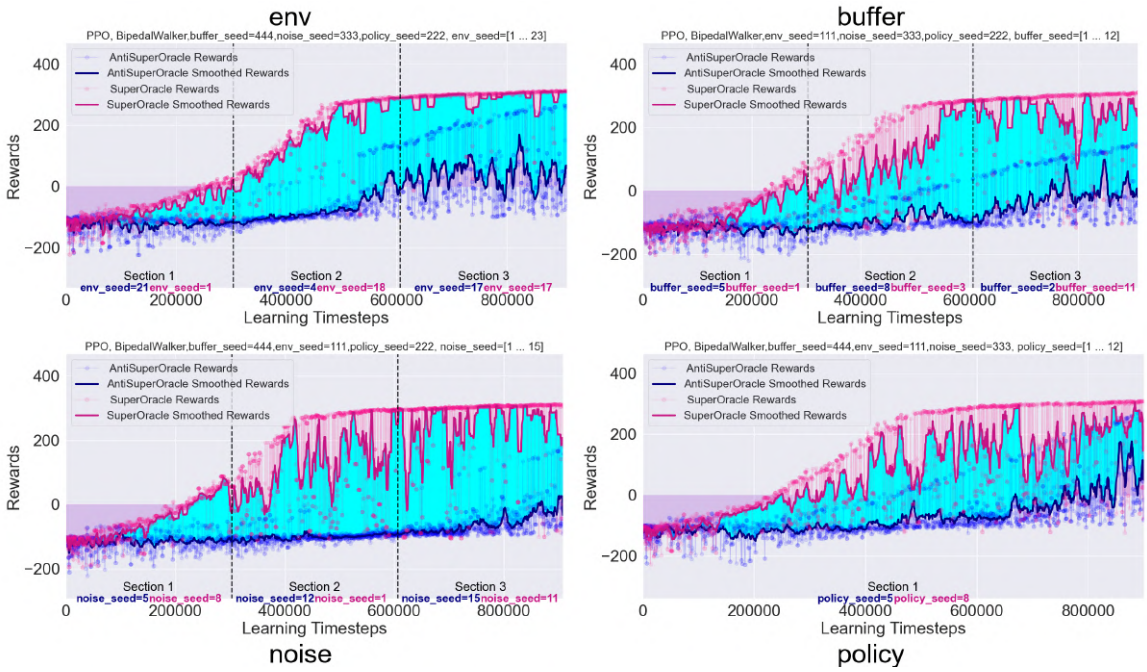


Figure 36: Combined Graph for BipedalWalker, PPO

Section	env	buffer	noise	policy
1	31.5%	27.6%	37.0%	-
2	92.2%	79.9%	83.1%	-
3	72.8%	82.9%	89.7%	-
all	60.9%	58.6%	69.7%	63.3%

Table 35: SMAPE for BipedalWalker, PPO

C.2.4 LunarLanderContinuous-v2

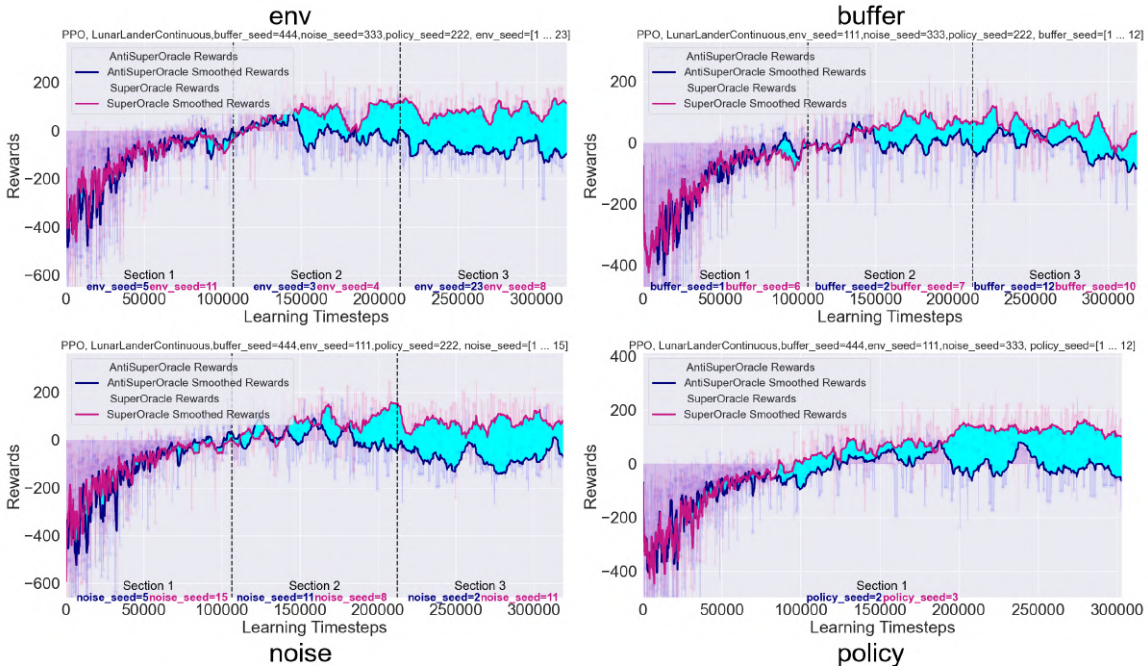


Figure 37: Combined Graph for LunarLanderContinuous, PPO

Section	env	buffer	noise	policy
1	43.2%	39.0%	46.3%	-
2	68.0%	65.5%	68.5%	-
3	83.2%	70.3%	79.1%	-
all	53.6%	47.3%	54.9%	48.9%

Table 36: SMAPE for LunarLanderContinuous, PPO

C.2.5 BipedalWalkerHardcore-v3

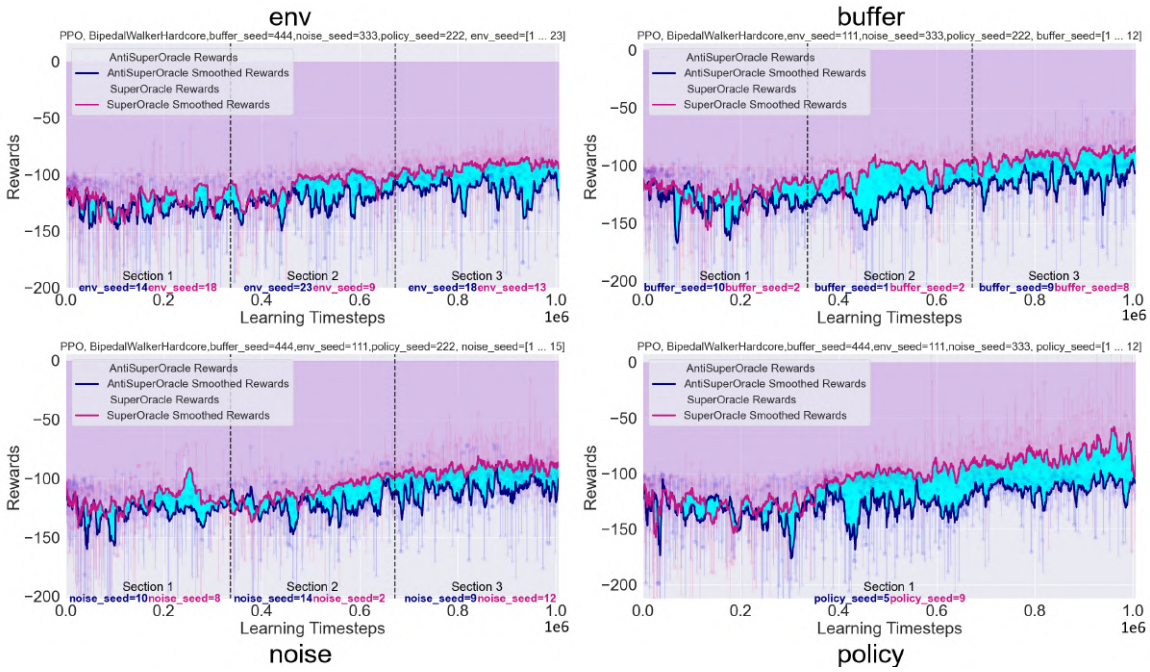


Figure 38: Combined Graph for BipedalWalkerHardcore, PPO

Section	env	buffer	noise	policy
1	9.6%	9.6%	9.5%	-
2	10.3%	12.0%	10.3%	-
3	11.6%	12.7%	11.2%	-
all	10.6%	11.4%	10.5%	13.5%

Table 37: SMAPE for BipedalWalkerHardcore, PPO

C.2.6 Ant-v4

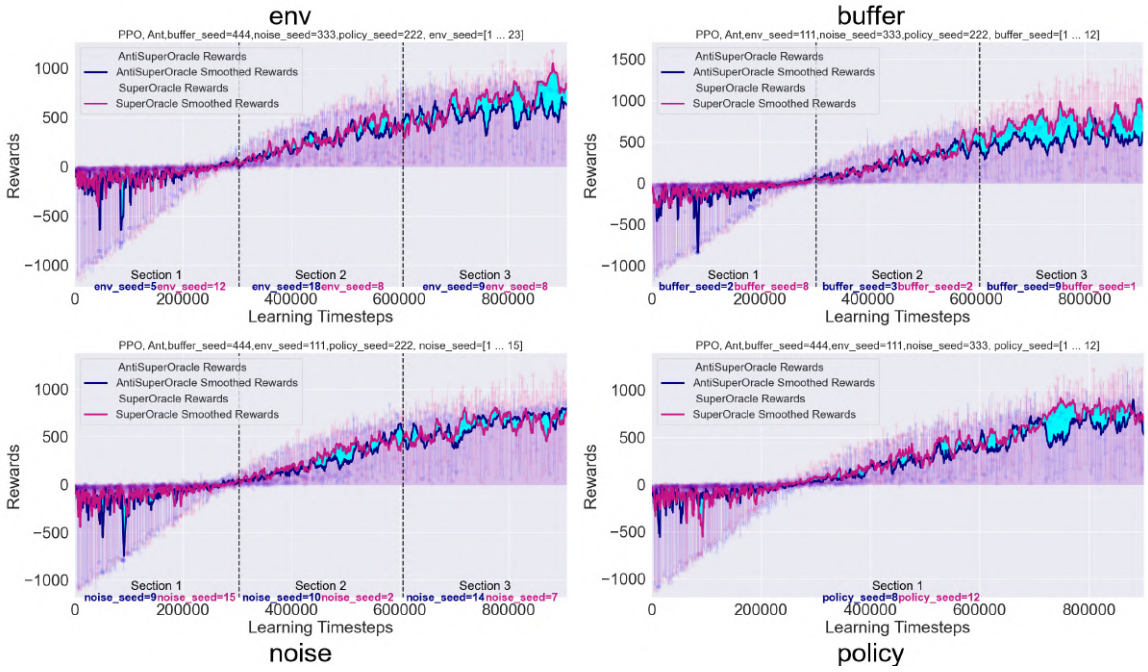


Figure 39: Combined Graph for Ant , PPO

Section	env	buffer	noise	policy
1	59.7%	61.2%	62.8%	-
2	46.8%	48.7%	47.6%	-
3	34.4%	36.5%	30.9%	-
all	52.1%	53.7%	54.0%	54.6%

Table 38: SMAPE for Ant PPO

C.2.7 HalfCheetah-v4

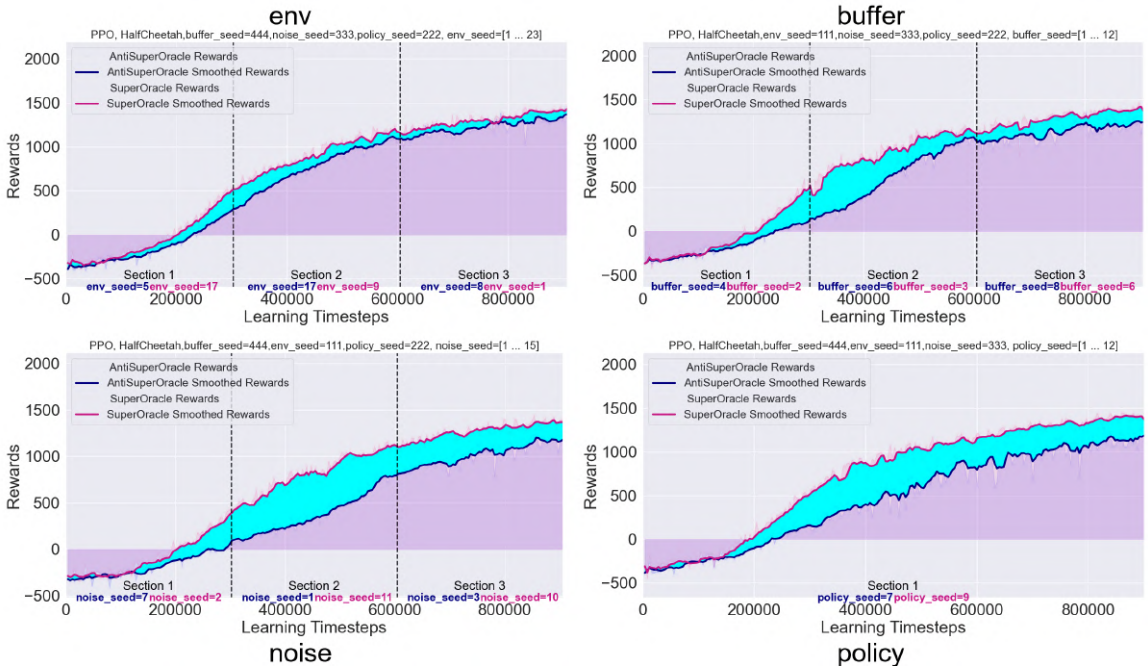


Figure 40: Combined Graph for HalfCheetah, PPO

Section	env	buffer	noise	policy
1	29.9%	36.8%	43.2%	-
2	8.9%	24.2%	40.0%	-
3	3.6%	6.5%	11.1%	-
all	14.1%	22.5%	31.4%	27.2%

Table 39: SMAPE for HalfCheetah, PPO

C.2.8 Hopper-v4

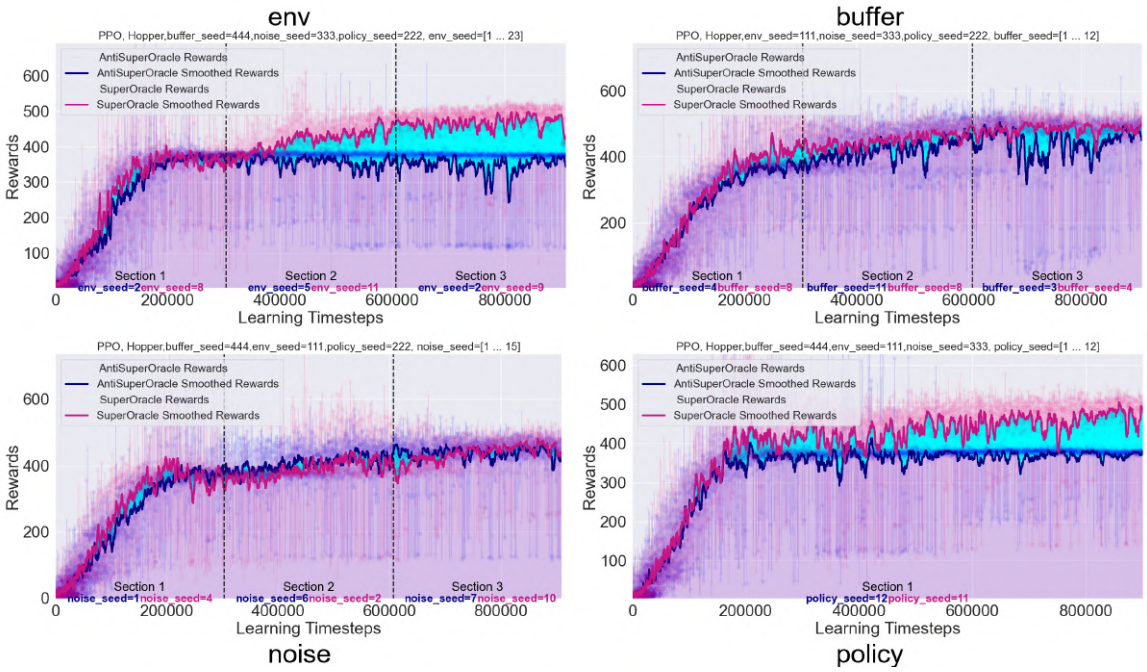


Figure 41: Combined Graph for Hopper, PPO

Section	env	buffer	noise	policy
1	20.3%	19.1%	28.3%	-
2	10.7%	10.7%	10.5%	-
3	17.4%	10.6%	8.0%	-
all	16.9%	14.6%	18.3%	16.9%

Table 40: SMAPE for Hopper, PPO

C.2.9 Reacher-v4

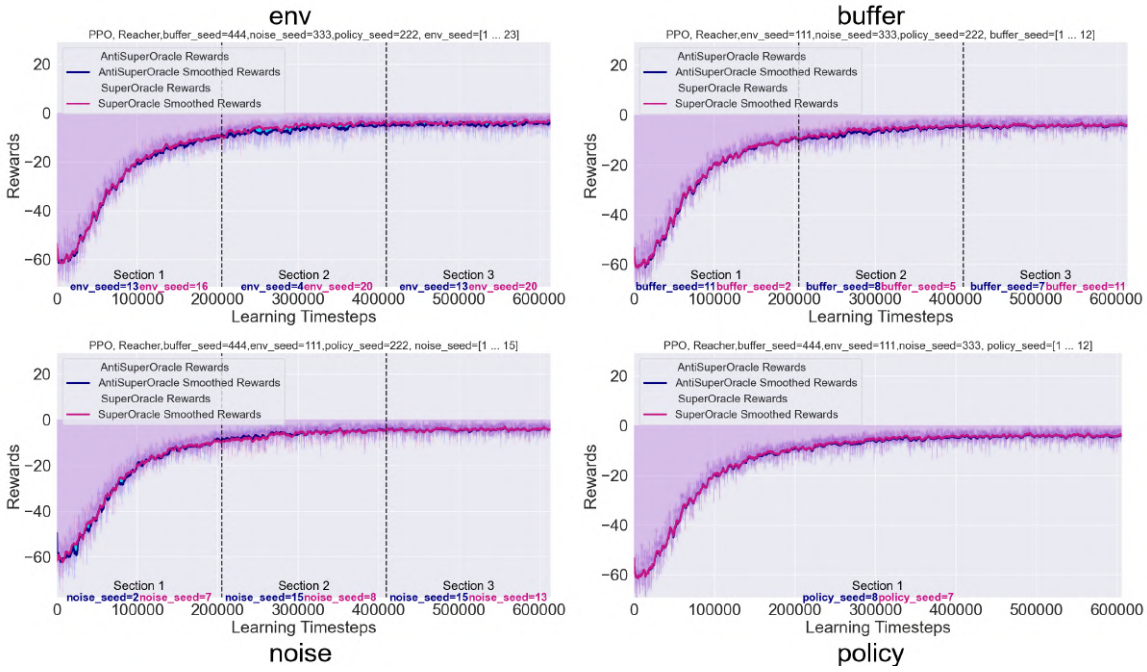


Figure 42: Combined Graph for Reacher, PPO

Section	env	buffer	noise	policy
1	7.4%	2.1%	6.5%	-
2	23.0%	6.8%	7.2%	-
3	26.0%	3.2%	3.8%	-
all	18.8%	4.0%	5.8%	3.2%

Table 41: SMAPE for Reacher, PPO

C.2.10 InvertedPendulum-v4

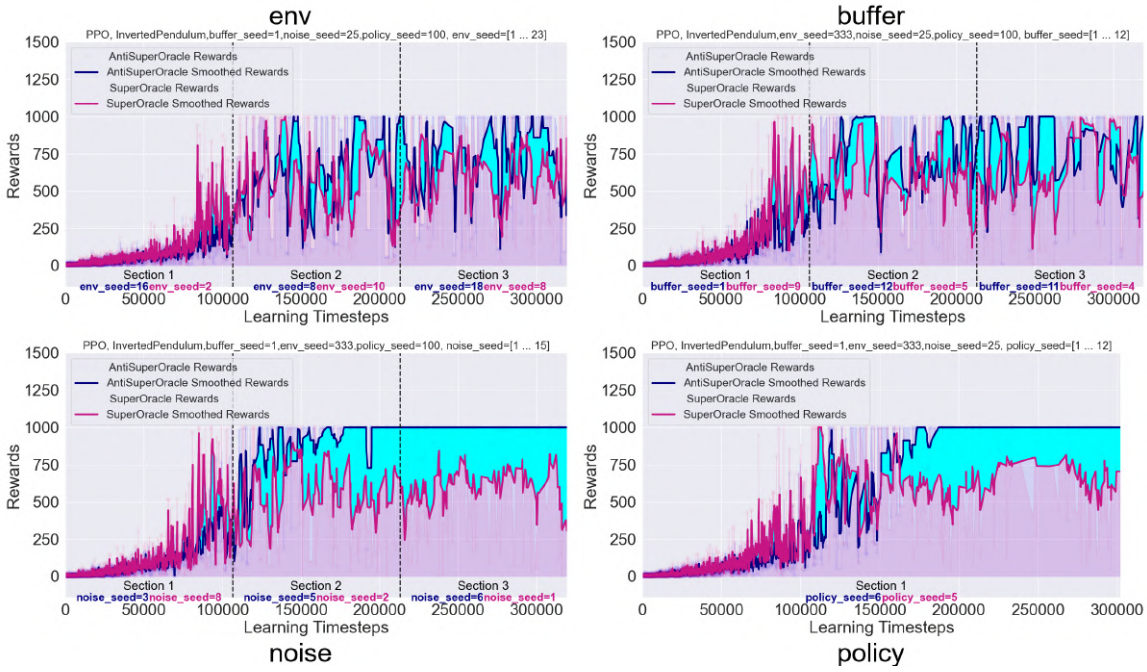


Figure 43: Combined Graph for InvertedPendulum, PPO

Section	env	buffer	noise	policy
1	25.4%	21.6%	32.4%	-
2	32.8%	30.7%	35.4%	-
3	29.8%	32.2%	32.0%	-
all	26.0%	22.6%	32.5%	23.8%

Table 42: SMAPE for InvertedPendulum, PPO

C.2.11 InvertedDoublePendulum-v4

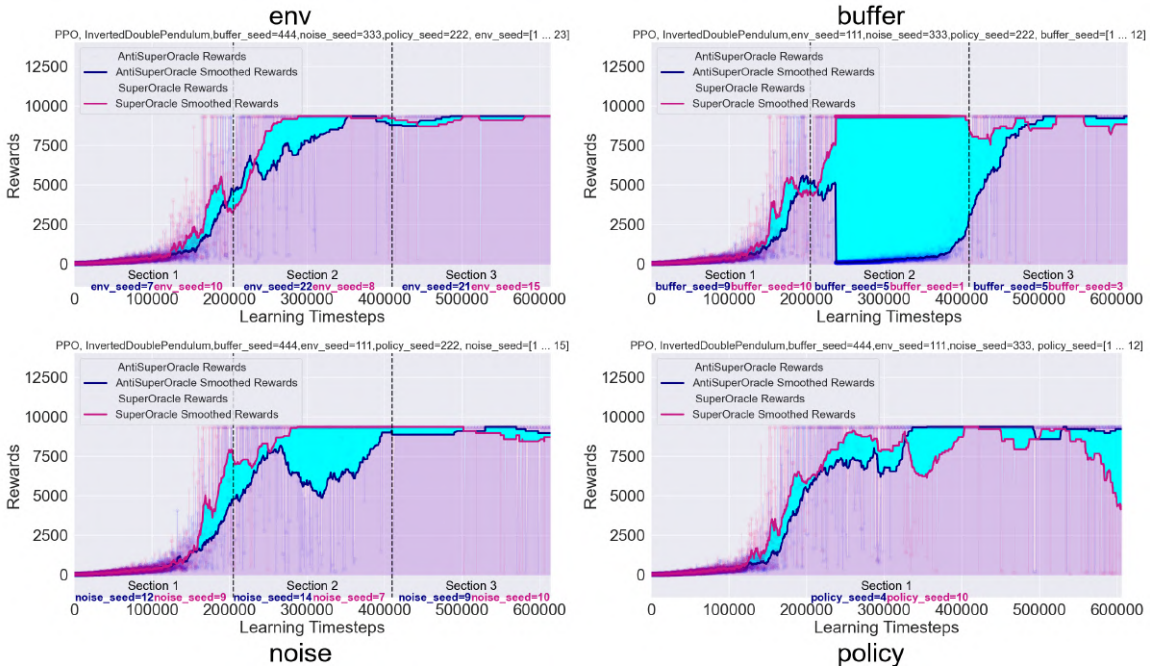


Figure 44: Combined Graph for InvertedDoublePendulum, PPO

Section	env	buffer	noise	policy
1	18.3%	17.2%	19.9%	-
2	15.3%	93.8%	22.8%	-
3	2.5%	13.1%	8.4%	-
all	17.9%	41.9%	19.8%	17.2%

Table 43: SMAPE for InvertedDoublePendulum, PPO

C.2.12 Pusher-v4

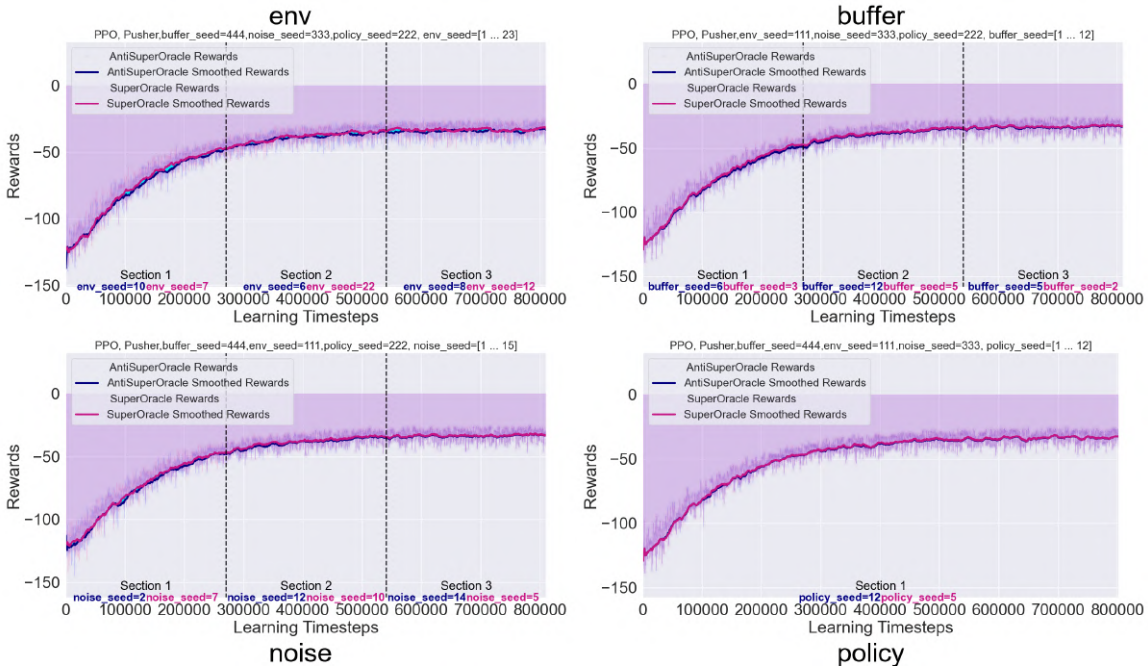


Figure 45: Combined Graph for Pusher, PPO

Section	env	buffer	noise	policy
1	3.8%	1.2%	3.2%	-
2	6.6%	1.1%	1.5%	-
3	7.7%	0.6%	0.6%	-
all	6.0%	1.0%	1.8%	0.6%

Table 44: SMAPE for Pusher, PPO

C.2.13 Walker2d-v4

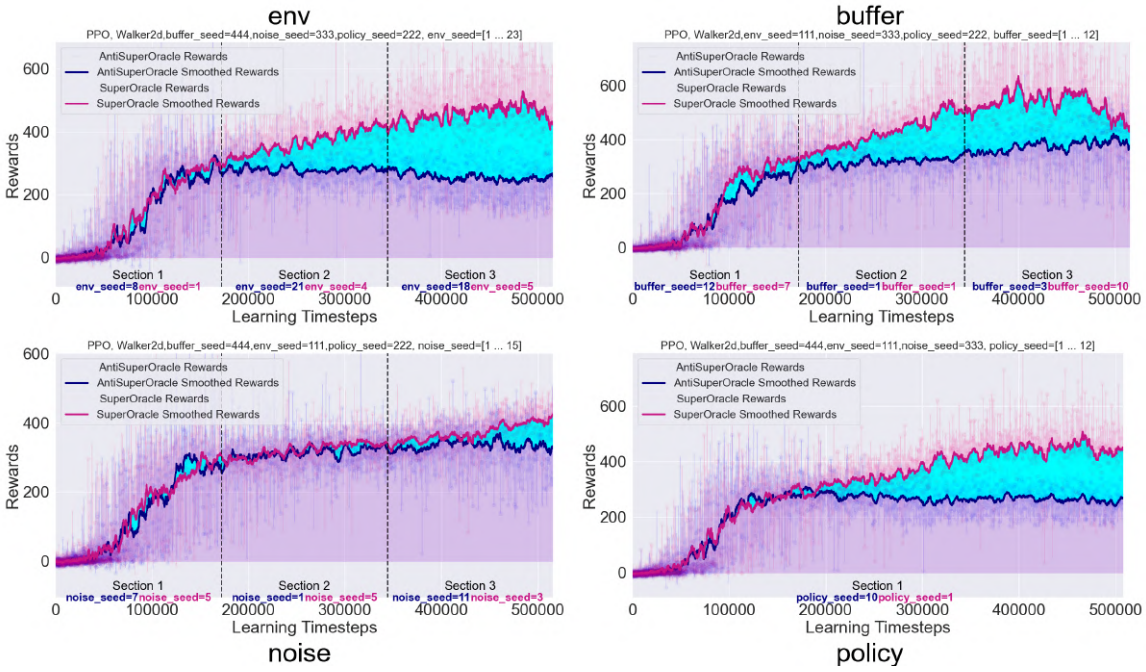


Figure 46: Combined Graph for Walker2d, PPO

Section	env	buffer	noise	policy
1	45.4%	42.4%	63.0%	-
2	16.8%	17.9%	10.0%	-
3	29.4%	19.3%	10.0%	-
all	34.4%	31.3%	38.8%	34.0%

Table 45: SMAPE for Walker2d, PPO

C.2.14 CarRacing-v2

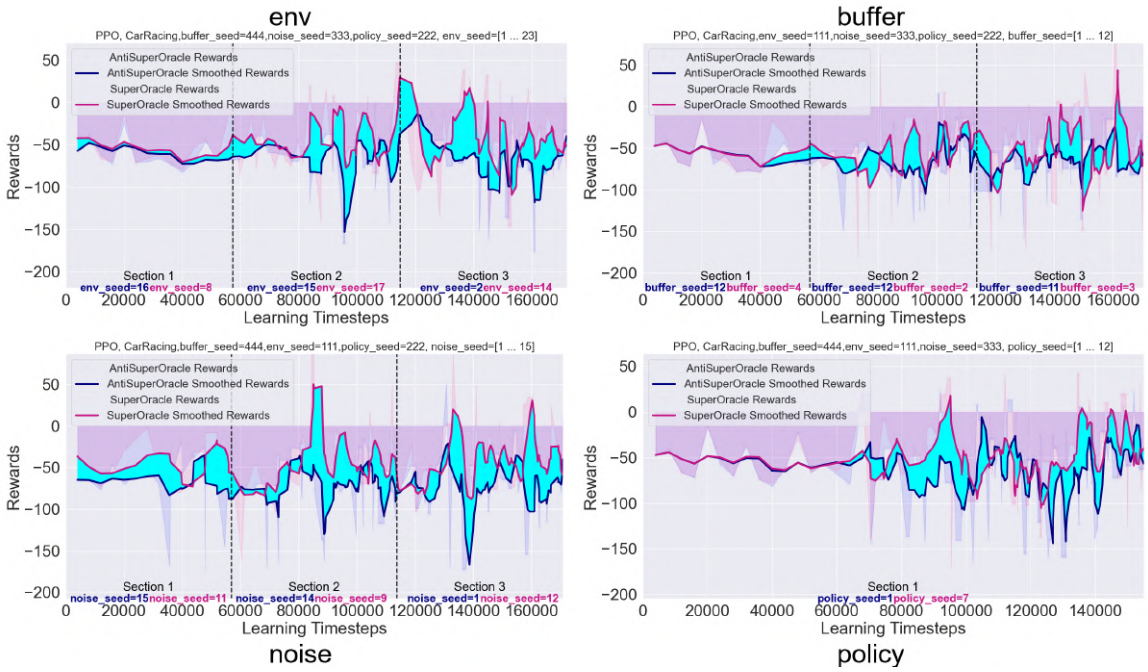


Figure 47: Combined Graph for CarRacing, PPO

Section	env	buffer	noise	policy
1	10.7%	8.1%	33.7%	-
2	42.6%	34.8%	36.5%	-
3	42.1%	40.5%	41.1%	-
all	39.6%	35.8%	38.3%	39.4%

Table 46: SMAPE for CarRacing, PPO

C.2.15 HumanoidStandup-v4

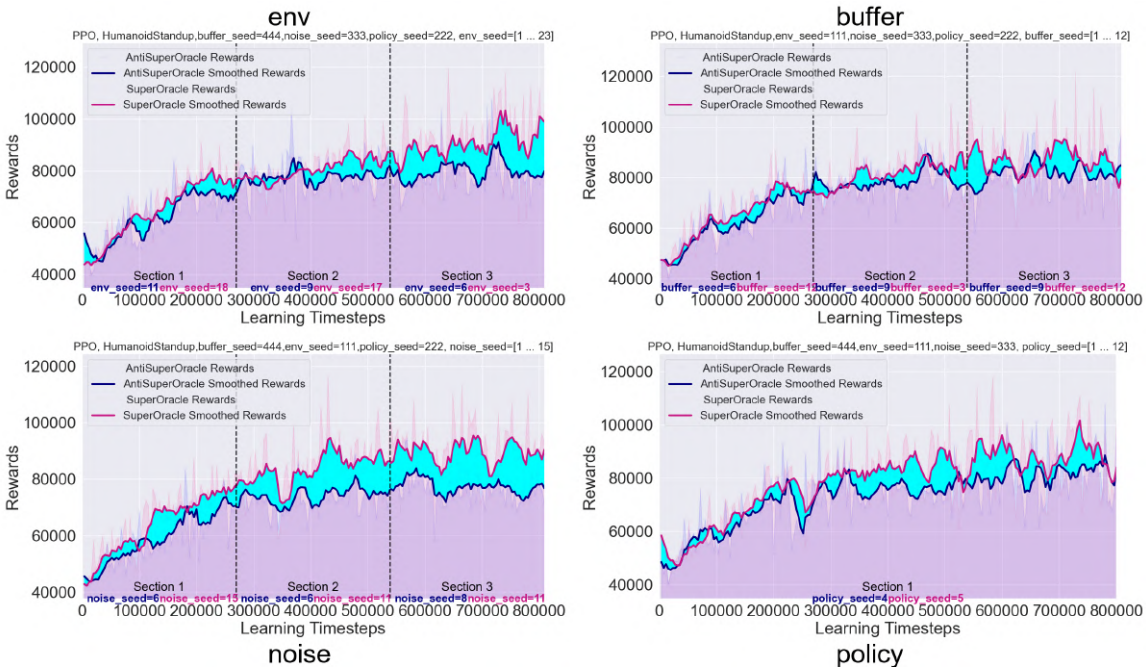


Figure 48: Combined Graph for HumanoidStandup, PPO

Section	env	buffer	noise	policy
1	5.1%	5.0%	7.0%	-
2	4.9%	4.7%	7.1%	-
3	8.6%	7.7%	8.0%	-
all	6.2%	5.8%	7.4%	6.6%

Table 47: SMAPE for HumanoidStandup, PPO

C.2.16 Humanoid-v4

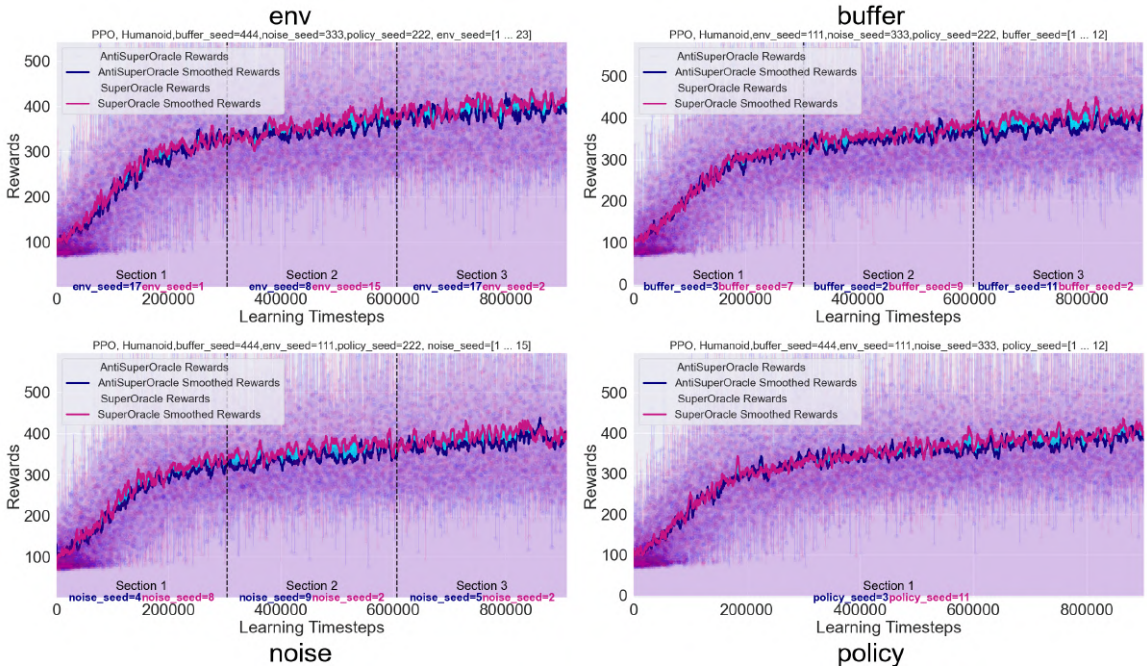


Figure 49: Combined Graph for Humanoid, PPO

Section	env	buffer	noise	policy
1	17.4%	16.8%	18.9%	-
2	13.6%	13.0%	13.8%	-
3	14.3%	13.5%	15.0%	-
all	15.4%	14.8%	16.3%	14.9%

Table 48: SMAPE for Humanoid, PPO

These images are available on GitLab [11].