



Empirical Evaluation of Random Network Distillation for DQN Agents

Alberto Moreno Sánchez¹

Supervisor(s): Neil Yorke-Smith¹, Pascal van der Vaart¹

¹EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Alberto Moreno Sánchez

Final project course: CSE3000 Research Project

Thesis committee: Neil Yorke-Smith, Pascal van der Vaart, Matthijs Spaan

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Abstract

This paper investigates how Random Network Distillation (RND), coupled with Boltzmann exploration, influences exploration behaviour and learning dynamics in value-based agents such as Deep Q-Learning (DQN) across a range of environments, from classic control tasks to behaviour suite benchmarks and contextual bandits. The study addresses the sensitivity of RND to key hyperparameters, the impact of exploration strategy design, and the transferability of settings across tasks. The results reveal that RND remains beneficial within DQN in both sequential and non-sequential tasks, but requires careful tuning of reward scaling, temperature, and network capacity to be effective. No universal hyperparameter configuration generalises across environments, and inappropriate tuning can lead to unstable learning or suboptimal outcomes. These findings provide practical insights into the strengths and limitations of applying RND within value-based reinforcement learning frameworks.

1 Introduction

Machine learning [1] has become a central approach for building systems that can learn from data and improve over time. Among its various subfields, reinforcement learning [2] focuses on training agents to make sequential decisions through interaction with an environment and feedback in the form of rewards. Deep Q-Learning (DQN) [3] is a value-based reinforcement learning algorithm that learns to approximate the optimal action-value function by maximizing expected cumulative rewards. DQN has demonstrated strong performance in a variety of high-dimensional, dense-reward environments, such as Atari 2600 games, where frequent feedback enables efficient learning through random exploration [4]. However, in sparse-reward settings, where rewards are rarely encountered, DQN often struggles to learn effective policies [5]. In practice, engineering dense reward functions [6] is often impractical or leads to unintended behaviour. In such scenarios, directed exploration methods that leverage epistemic uncertainty [7], the agent’s uncertainty about the environment dynamics due to limited experience, become essential to strategically guide exploration toward less understood states where informative feedback can be acquired.

One of the most popular approaches addressing this challenge is Random Network Distillation (RND) [8], which introduces an intrinsic reward based on the prediction error of a fixed randomly initialized network. This method provides a simple and scalable mechanism for detecting novelty, and achieved state-of-the-art performance on exploration-challenging Atari environments, such as Montezuma’s Revenge, where rewards are extremely sparse and difficult to discover through random exploration.

Despite the growing popularity of RND as a method for intrinsic motivation, several important questions about its behaviour in value-based reinforcement learning remain unresolved. First, there is limited empirical evidence on whether a robust set of hyperparameter configurations can be identified that generalize across different environments. Second, while prior work introduces the method, little is known about which parameters are most influential and how they affect exploration or learning performance. Third, most experimentation with RND has been conducted in continuous-control or policy-gradient settings, leaving its interaction with Deep Q-Learning (DQN) underexplored. Additionally, the method has received limited evaluation in simplified settings such as contextual bandits, where behavioural dynamics are easier to interpret.

To address these gaps, this study seeks to answer the following research questions:

1. **Are robust settings transferable across environments?** The investigation focuses on whether hyperparameters or design choices that perform well in one environment generalize effectively to others.
2. **Which hyperparameters are most influential?** The sensitivity of RND-augmented DQN to different hyperparameter configurations is analyzed, assessing their impact on both exploration and learning stability.
3. **How well does Random Network Distillation integrate with Deep Q-Learning in practice?** The empirical behaviour of RND within a value-based framework is evaluated, addressing a gap in prior work that has primarily focused on policy-gradient and continuous control settings.
4. **Does the method remain beneficial in well-defined contextual bandit settings?** It is examined whether the exploration benefits of RND persist in simplified settings where optimal behaviour is easier to identify.

Starting from the background, this paper introduces the theoretical foundations of value-based reinforcement learning and RND. The following section details how RND is integrated into DQN, including key architectural and design choices. The experimental setup includes the motivation behind the selected environments, a description of the baseline DQN agent, and hyperparameter sweeps in simple environments. The results and discussion section provides empirical evidence on the influence of RND across diverse tasks, highlighting key sensitivities and limitations. The conclusions and future work section summarizes the main findings and proposes directions for further research. Finally, the responsible research section addresses reproducibility, ethical considerations, and use of LLMs.

2 Background

2.1 Reinforcement Learning

The agent’s environment is modeled as a Markov Decision Process (MDP) [9], defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s' | s, a)$ is the transition probability from state s to state s' given action a , $R(s, a)$ is the extrinsic reward function, and $\gamma \in [0, 1]$ is the discount factor. At each timestep t , the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to its policy $\pi(a | s)$, receives a scalar reward r_t , and transitions to the next state $s_{t+1} \sim P(\cdot | s_t, a_t)$.

In this work, Deep Q-Learning (DQN) algorithm is adopted as the value-based learning backbone. The agent aims to approximate the optimal action-value function $Q^*(s, a)$, which satisfies the Bellman optimality condition:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \quad (1)$$

2.2 Random Network Distillation

To improve exploration in environments where external rewards are sparse or uninformative, intrinsic rewards are computed through Random Network Distillation (RND) and integrated, resulting in a composite reward signal that augments the extrinsic objective with novelty-based incentives. The learning objective thus becomes maximizing the expected cumulative discounted sum of combined rewards.

RND employs a fixed a randomly initialized and fixed target network f_{target} and a trainable predictor network $f_{\text{predictor}}$. At each time step t , the intrinsic reward is defined as the squared error between the two network outputs:

$$r_t^{\text{int}} = \|f_{\text{target}}(\hat{s}_t) - f_{\text{predictor}}(\hat{s}_t)\|^2 \quad (2)$$

This intrinsic signal is combined with the environment-provided extrinsic reward to obtain the total reward used for learning:

$$r_t = \beta_{\text{ext}} \cdot r_t^{\text{ext}} + \beta_{\text{int}} \cdot r_t^{\text{int}} \quad (3)$$

To ensure stable training, observation normalization is applied. Since the parameters of f_{target} are frozen, it cannot adapt to varying input scales. Without normalization, its outputs can collapse to low-variance representations, leading to diminished novelty signals [8]. Each observation is normalized using running estimates of mean and standard deviation, followed by clipping:

$$\hat{s}_t = \text{clip} \left(\frac{s_t - \mu_t}{\sigma_t + \epsilon}, -5, 5 \right) \quad (4)$$

where s_t is the raw observation at time t , μ_t and σ_t are the running estimates of the mean and standard deviation for each input dimension, ϵ is a small constant added for numerical stability, and clip restricts each dimension to the range $[-5, 5]$.

The normalization statistics are initialized by stepping a random agent in the environment for a fixed number of steps¹ before training begins. The same normalization is applied to both the predictor and target networks, but not to the policy network. This design follows conventions established in continuous control settings and ensures that the RND module receives consistently scaled inputs across different environments.

2.3 Environments

The environments selected for this study span a diverse range of structural and reward characteristics, enabling a nuanced analysis of how RND-integrated DQN agents behave under different conditions. By comparing across both simple and challenging tasks, it is possible to obtain a better understanding of the contexts in which novelty-based intrinsic motivation yields meaningful exploration benefits.

CartPole [10] and **Acrobot** [11] are standard control environments with continuous state spaces and discrete action spaces. Both offer dense reward functions, but differ significantly in their exploration dynamics and reward shaping. In CartPole, the agent receives a reward of +1 for every timestep it keeps the pole balanced, making the reward signal immediate and strongly correlated with progress. In contrast, Acrobot only provides a reward of -1 at every step until the agent successfully swings up and reaches a height threshold, resulting in a sparse and delayed reward signal despite the environment being low-dimensional.

DeepSea [12] is a synthetic environment designed specifically to study exploration under sparse and deceptive rewards. The agent must navigate to the bottom-right corner of a grid, incurring a small negative reward at each step, with a large positive reward only available at the goal state. Crucially, reaching this goal requires a highly specific sequence of actions with no intermediate rewards to guide the agent.

¹The number of steps configured across the different experiments was 1000.

MNISTBandit [12] is a contextual bandit environment where each context is a randomly selected MNIST digit image. The agent must classify the digit by selecting an appropriate action corresponding to its label. The environment provides an immediate binary reward indicating whether the classification was correct.

Together, these environments offer variety in difficulty and reward sparsity, enabling controlled experimentation on the conditions under which RND improves learning and exploration in value-based reinforcement learning.

While RND was originally demonstrated with policy-based methods (e.g., PPO), its theoretical framework applies generically to any reinforcement learning agent, including value-based approaches like DQN. The following section details the specific integration of RND into the DQN framework

3 Random Network Distillation in Deep Q-Network

Integrating RND into DQN requires a series of design decisions to ensure compatibility between the intrinsic reward mechanism and the value-based learning framework. In particular, adaptations are needed in how exploration is handled, since DQN typically relies on external strategies for exploration, and how intrinsic rewards are incorporated into the replay buffer. These choices significantly influence learning dynamics and must be carefully considered to fully leverage the benefits of RND in discrete-action settings.

3.1 Exploration Strategy via Boltzmann Sampling

In RND-augmented Deep Q-Learning, the exploration strategy plays a crucial role in determining how effectively intrinsic rewards translate into useful behaviour [13]. While traditional DQN implementations often select actions via a hard $\arg \max$ over Q-values, this approach results in deterministic policies and can severely limit exploration, particularly when intrinsic rewards are sparse or nonstationary [14]. An alternative is ϵ -greedy exploration [15], which adds stochasticity by choosing random actions with fixed probability, but this method does not consider the relative values of suboptimal actions and may lead to inefficient or incoherent exploration in high-dimensional settings.

To address these limitations, Boltzmann exploration [16] is adopted, where actions are sampled from a softmax distribution over Q-values:

$$\pi(a \mid s) = \frac{\exp(Q(s, a)/\tau)}{\sum_{a'} \exp(Q(s, a')/\tau)} \quad (5)$$

The temperature parameter τ modulates the trade-off between exploration and exploitation. As $\tau \rightarrow 0$, the policy becomes deterministic, favoring high-value actions. As τ increases, the distribution becomes more uniform, encouraging exploration [17]. Unlike ϵ -greedy, which samples uniformly among all actions regardless of their value, Boltzmann sampling retains sensitivity to Q-value differences, even among non-optimal actions.

This method introduces graded preferences over actions based on their estimated value, allowing the agent to prefer promising actions while still maintaining stochasticity [18]. Prior work has shown that Boltzmann policies can result in more stable and directed exploration compared to ϵ -greedy or greedy methods, especially in the early stages of training when value estimates are noisy or incomplete [19].

In contrast, the baseline DQN agent employs ϵ -greedy exploration, where the agent selects a random action with probability ϵ and otherwise chooses the action with the highest estimated Q-value. While simple and effective in some settings, ϵ -greedy exploration imposes a binary decision between random and greedy actions, which can result in abrupt and inefficient exploration, especially when Q-values are not yet well-formed. Unlike Boltzmann sampling, it does not consider the relative quality of suboptimal actions during exploration [20].

Over preliminary experiments, it was observed that in several environments, particularly *MontezumaRevenge* [21], the entropy of the action distribution under arg max selection rapidly collapsed to zero, effectively eliminating exploration. By contrast, Boltzmann exploration maintained higher entropy throughout training, enabling more effective use of intrinsic rewards provided by RND.

3.2 Replay Buffer Design and Intrinsic Reward Integration

To support off-policy learning in RND-augmented DQN, a replay buffer is used to store agent experiences for sampling during training. The buffer maintains a record of transitions, including observations, actions, rewards, and done signals.

The replay buffer implementation faces a critical design choice: whether to store only extrinsic rewards (recomputing intrinsic rewards during training) or preserve both reward types. Storing only extrinsic rewards allows intrinsic reward calculations to benefit from updated RND network parameters during training. However, this approach introduces non-stationarity in the composite reward signal, as later training phases recompute intrinsic rewards using predictor networks that have evolved beyond their initial state. Conversely, storing both rewards ensures temporal consistency in the exploration incentives and eliminates computational overhead from repeated intrinsic reward calculations, at the cost of frozen exploration priorities that don’t adapt to the agent’s evolving knowledge state.

In this work, the approach of storing both extrinsic and intrinsic rewards in the replay buffer was adopted, thereby prioritizing learning stability. This approach prevents destructive interference between the evolving RND predictor network and historical experiences. The frozen intrinsic rewards maintain consistent temporal difference targets despite the predictor network’s ongoing adaptation, avoiding the moving target problem [22] that occurs when recomputing rewards from updated network parameters.

However, this approach is not without limitations, particularly regarding the potential for outdated exploration incentives. Future research could explore more sophisticated approaches that combine the benefits of both strategies.

For a detailed overview of the algorithmic flow, refer to the pseudo-code provided in Appendix A.1, and consult the implementation at <https://github.com/albertomoreno04/EpistemicUncertaintyRL>.

4 Experimental Setup

Now that the key design choices for adapting Random Network Distillation (RND) to Deep Q-Learning (DQN) have been established, this section outlines the experimental framework used to evaluate its performance. The baseline agent and implementation tools are first described, followed by justification of the selected environments. Hyperparameter sweep

structures for simple control settings are then detailed, concluding with an evaluation protocol for contextual bandits.

4.1 Baseline Agent and Implementation Framework

To provide a fair comparison and isolate the effects of RND integration, a baseline DQN agent is implemented using the CleanRL [23] framework². CleanRL offers a high-quality, minimal, and reproducible implementation of reinforcement learning algorithms, and is widely adopted in the community for benchmarking and prototyping. The baseline agent follows standard DQN practices, including target network updates and experience replay. The implementation is based on the JAX variant of CleanRL, leveraging its efficiency and composability for high-performance learning workflows.

For exploration, the baseline agent employs the classic ϵ -greedy strategy, where actions are selected randomly with probability ϵ and greedily otherwise.

The value of ϵ is computed using a linear schedule, which gradually anneals the exploration probability from its initial value to a final value over a predefined portion of training. Specifically, ϵ is updated at each step t as:

$$\epsilon(t) = \max \left\{ \frac{\epsilon_{\text{final}} - \epsilon_{\text{start}}}{e_f \times t_t} \cdot t, \epsilon_{\text{final}} \right\} \quad (6)$$

where ϵ_{start} and ϵ_{final} denote the initial and final exploration probabilities, respectively, e_f defines the fraction of total timesteps over which the annealing takes place, and t_t is the number of total timesteps. This scheduling ensures a smooth transition from exploratory to exploitative behaviour as training progresses, establishing a popular and representative baseline for value-based reinforcement learning [4] that enables direct assessment of the contribution of intrinsic rewards.

4.2 Environments Motivation

The environments selected for this study are designed to expose different dimensions of exploration behaviour in value-based agents. The environments span a range of reward sparsity, decision complexity, and temporal structure, enabling evaluation of how RND-based intrinsic motivation impacts learning across diverse settings.

CartPole and **Acrobot** offer a contrast between environments with dense and sparse reward feedback. This contrast provides insight into how RND contributes to learning when external feedback is either frequent and shaped (**CartPole**) or sparse and binary (**Acrobot**). In such low-dimensional control tasks, the interaction between the additional novelty signal provided by RND and standard reward shaping can be analyzed to determine whether they exhibit complementary or interfering effects.

DeepSea provides a highly interpretable benchmark specifically crafted to test an agent’s ability to perform long-horizon exploration under deceptive rewards. Success in DeepSea requires committing to extended action sequences without immediate reward feedback, making it a valuable setting to assess whether RND helps mitigate the short-horizon bias of standard DQN.

MNISTBandit introduces a single-step decision-making task with high-dimensional observations and no temporal dependencies. As a contextual bandit environment, this setup

²Available at https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/dqn_jax.py

enables analysis of whether RND encourages meaningful exploratory behaviour in non-sequential settings, where novelty must be discovered within the context space itself.

Together, these environments provide a structured testbed to probe the capabilities and limitations of RND in value-based reinforcement learning. By evaluating performance across varying reward sparsity, temporal complexity, and input dimensionality, this study aims to offer a deeper understanding of when and how intrinsic motivation enhances exploration in DQN agents.

4.3 Hyperparameter Sweeps in Simple Environments

To systematically investigate how key design choices influence the performance of RND-augmented DQN agents, hyperparameter sweeps were conducted on three different environments: `CartPole`, `Acrobot`, and `DeepSea`. These environments offer a well-understood and computationally efficient setting for evaluating algorithmic behaviour across a range of configurations.

Each agent is evaluated across five random seeds per sweep configuration to assess the stability and sensitivity of the results. The hyperparameters varied are:

- **Boltzmann temperature (τ):** controls the sharpness of the action selection distribution, impacting the exploration-exploitation trade-off.
- **Predictor and target hidden dimension size:** controls the capacity of the predictor and target networks.
- **Intrinsic/extrinsic reward coefficient ratio:** the intrinsic reward coefficient is fixed to 1, and the extrinsic reward coefficient is varied to explore different balances between task-driven and novelty-driven learning.

Controlled sweeps across multiple environments and seeds provide empirical insights into how these parameters interact with RND and influence agent behaviour. This type of analysis is essential for understanding when RND is likely to be effective, what tuning is necessary for it to succeed, and whether robust configurations emerge across tasks with different reward structures³.

4.4 Contextual Bandit Evaluation

In addition to standard control environments, contextual bandit settings are included to evaluate the behaviour of RND in highly simplified, non-sequential decision-making tasks. Unlike full reinforcement learning environments, contextual bandits present a new context at each timestep without state transitions, isolating the agent’s ability to explore under uncertainty without the confounding effects of temporal credit assignment [24]. This setting provides a controlled testbed to analyze how the intrinsic reward signal influences action selection when each decision is independent and immediate feedback is available. Performance in these environments is examined to assess whether RND retains its exploratory benefits when the problem structure is minimal, and to evaluate how it compares to simpler strategies like ϵ -greedy, as employed by the baseline agent.

³See Appendix A.2 for the full list of hyperparameter values used in each sweep.

5 Results and Discussion

This section presents an empirical analysis of how RND influences learning in DQN agents. The analysis begins with an examination of the method’s robustness across environments, followed by an investigation of the balancing between intrinsic and extrinsic rewards. The relation between temperature and convergence is then studied, along with an evaluation of performance in the MNIST contextual bandit. The section concludes with an analysis of the influence of predictor and target hidden dimension size.

5.1 Environment-Specific Hyperparameter Sensitivity

One of the core questions in this study is whether hyperparameter configurations that perform well in one environment generalize effectively to others. This is particularly relevant for RND-augmented DQN agents, where the interaction between intrinsic rewards and value-based learning dynamics introduces additional sensitivity to design choices.

The experimental results indicate that there is no universally robust configuration across environments. Instead, strong performance requires environment-specific tuning of key hyperparameters. This sensitivity highlights an important limitation of the approach: although RND provides a generic mechanism for intrinsic motivation, its effectiveness is highly dependent on the balance of extrinsic and intrinsic rewards, and exploration strategy parameters.

To illustrate this point, Table 1 shows how **CartPole** and **Acrobot** exhibit substantially different optimal settings in the hyperparameter sweeps conducted, underscoring the need for careful tuning when applying RND across tasks.

Hyperparameter	CartPole	Acrobot
Predictor and Target Hidden Dimension	16	64
Temperature (τ)	1.0	0.2
Extrinsic/Intrinsic Coefficient	0.2	20

Table 1: **Highest return RND agent hyperparameters for CartPole and Acrobot.** The configurations shown achieved an average episodic return of 500 (**CartPole**) and -68 (**Acrobot**), each averaged over 5 different seeds. See detailed hyperparameter sweep settings used for this table in Appendix A.2.

This trend is not limited to these two environments. Across all tasks studied, including **DeepSea** and **MNISTBandit**, optimal hyperparameter configurations differed substantially Appendix 4. These findings suggest that while RND is a flexible mechanism for driving exploration, effective use of the method requires careful tuning for each target environment. As such, general-purpose deployment of RND-augmented DQN agents without environment-specific tuning may lead to suboptimal performance.

5.2 Balancing Intrinsic and Extrinsic Rewards

In RND-augmented DQN agents, the magnitude of the intrinsic reward can vary substantially across environments. Although the observations passed to the predictor and target networks are normalized, the outputs of these networks can differ in scale, and the intrinsic reward is computed as the mean squared error (MSE) between them. As a result, the intrinsic reward signal can take different magnitudes depending on the environment observation space and training hyperparameters.

To address this, separate coefficients to scale the intrinsic and extrinsic rewards are introduced. In practice, the intrinsic coefficient is kept fixed at 1, while the extrinsic coefficient is varied to control the ratio between exploration and exploitation signals. This ratio effectively determines whether the agent prioritizes novelty-seeking behaviour or task-driven learning.

Figure 1 illustrates the impact of this balancing in *Acrobot*, showing that only the runs with an extrinsic coefficient of 20 achieve strong returns, while the others fail to converge to optimal behaviour. The intrinsic reward in this environment oscillates around 12, whereas the extrinsic reward is typically negative, bounded between -1 and 0 . Critically, episodes in *Acrobot* terminate only upon successful completion of the task (swinging the end effector above the target line) or after 500 time steps. In this setting, low extrinsic coefficients allow the intrinsic reward to dominate, promoting continued exploration without sufficient incentive to complete episodes early. However, *Acrobot* requires the agent to exploit known strategies to reliably achieve the goal and terminate episodes successfully. Scaling up the extrinsic reward with a higher coefficient shifts the balance toward exploitation, providing the necessary incentive for task completion and enabling the agent to learn an optimal policy.

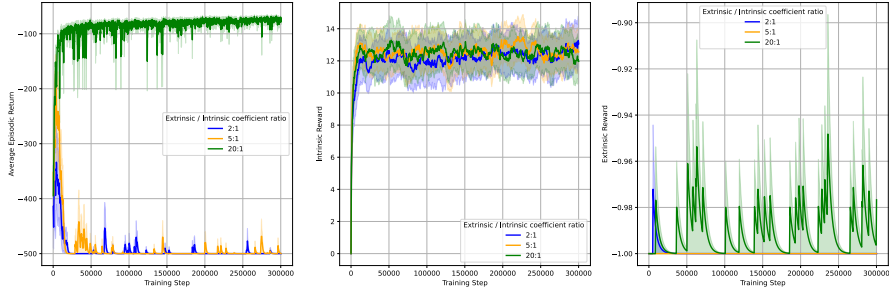


Figure 1: **Effect of extrinsic reward scaling on exploration-exploitation balance in *Acrobot*.** Runs were conducted with different hyperparameter settings, running each configuration across 5 different seeds. The shaded regions represent the standard error across seeds. The left plot shows the average episodic return per group of runs. Middle and right plots show the evolution of intrinsic and extrinsic rewards, respectively. Higher extrinsic coefficients enable the agent to exploit and achieve strong performance, while lower coefficients lead to intrinsic signal dominance and suboptimal behaviour. Smoothed using exponential moving average with $\alpha = 0.1$ for both mean and standard error. See hyperparameter settings used in this figure in Appendix A.3.

These results demonstrate the critical need for tuning RND’s reward scaling ratio. In environments requiring exploitation over pure exploration, appropriate extrinsic signal scaling is necessary for learning effective policies. The observed sensitivity to fixed scaling ratios suggests promise for dynamic adjustment mechanisms that could automatically balance intrinsic and extrinsic rewards throughout training, mitigating the risk of outdated exploration incentives.

5.3 Impact of Temperature on Learning Stability and Exploration Behaviour

While Boltzmann exploration offers a flexible mechanism for trading off exploration and exploitation, it is also sensitive to the choice of temperature parameter τ . Experimental results indicate that using an improperly tuned temperature can result in highly unstable training dynamics. Even after an agent appears to achieve strong performance during training, a high temperature can cause abrupt drops in episodic return, disrupting convergence.

This behaviour is illustrated in Figure 2, where lower temperatures demonstrate more stable learning dynamics and appear to converge toward an optimal return. In contrast, higher temperature settings display instability, with sudden drops in average return even after periods of optimal return. This suggests that while higher temperatures promote exploration, they can also interfere with learning stability if not properly balanced.

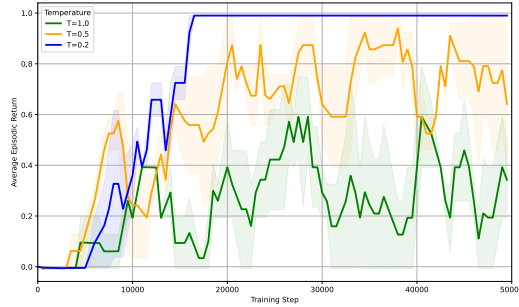


Figure 2: **Comparison of average episodic return between different temperature values** in DeepSea Runs were conducted with environment size 10, across 5 seeds, using three different temperature settings. The shaded regions represent the standard error across seeds. The blue line ($\tau = 0.2$) shows stable learning and convergence, while higher temperatures lead to unstable behaviour and sudden drops in performance. Smoothed using a moving average with window size 5 for both mean and standard error. See hyperparameter settings used in this figure in Appendix A.3.

Temperature significantly influences exploration behavior in RND-augmented DQN agents, as measured through policy entropy. Higher temperatures ($\tau = 1.0$) promote exploratory behavior with reduced variance, while lower temperatures ($\tau = 0.2$) yield more deterministic policies with increased variance. As theoretically expected, this highlights τ 's critical trade-off: consistent exploration versus stable value learning. Appendix B provides complete entropy calculation methodology and detailed visual analysis.

Overall, these results show that temperature tuning plays a crucial role in balancing exploration and learning stability in RND-augmented DQN agents using Boltzmann exploration. Careful calibration is required to achieve both effective exploration and stable convergence. Future work should investigate temperature annealing schedules to dynamically adjust exploration pressure, potentially improving performance in environments requiring precise policy refinement during later training stages.

5.4 Evaluating Novelty-Driven Exploration in Contextual Bandits

Contextual bandit environments offer a simplified setting to analyze exploration strategies without the confounding effects of temporal credit assignment. In these environments, each timestep presents an independent context, and the agent must select an action based solely on that observation. This makes contextual bandits an ideal testbed for isolating whether RND promotes meaningful exploration in the input space.

The `MNISTBandit` environment provides a high-dimensional, structured input domain where each context is a digit image, and the agent is rewarded for correct classification. This setup enables assessment of whether RND-augmented DQN agents learn faster or more effectively than baseline DQN agents relying on ϵ -greedy exploration.

To evaluate learning dynamics, the average classification accuracy over the last 1000 steps is tracked at each timestep. As shown in Figure 4, RND consistently outperforms the baseline in terms of classification accuracy, indicating faster and more efficient learning. The enhanced performance suggests that intrinsic rewards from RND, coupled with Boltzmann exploration, help the agent focus on novel or ambiguous inputs, even in the absence of temporal exploration challenges.

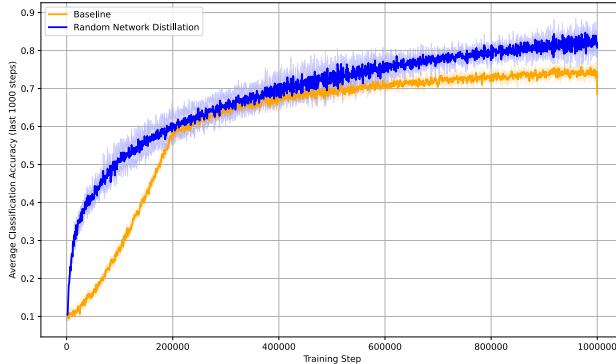


Figure 3: **Average classification accuracy comparison in MNISTBandit contextual bandit environment.** Runs conducted across 5 different seeds, comparing RND-augmented DQN and baseline DQN agents. Average classification accuracy is computed by mapping environment rewards in the range $[-1, 1]$ to accuracy values in $[0, 1]$, averaged over a 1000-step sliding window. The shaded regions represent the standard deviation across seeds. The classification accuracy of the RND agent is consistently higher, indicating faster learning and more effective use of novel information in the context space. No exponential smoothing was applied; accuracy values are based on a rolling average over raw per-step rewards. See hyperparameter settings used in this figure in Appendix A.3.

To further support this conclusion, the number of unique contexts visited throughout training was tracked. This metric, shown in Appendix A.4, shows that RND agents achieve higher classification accuracy despite visiting a similar number of unique contexts, suggesting that the intrinsic reward, coupled with Boltzmann exploration, helps focus learning on more informative or uncertain inputs rather than merely broadening coverage.

5.5 Impact of Predictor and Target Network Capacity on Intrinsic Reward Dynamics

The size of the predictor and target hidden dimension used in the predictor and target networks was included as a hyperparameter in the experimental sweeps. While prior RND literature typically focuses on tuning reward scaling or exploration strategies, the role of the predictor and target hidden dimension in shaping intrinsic reward dynamics has received comparatively little attention.

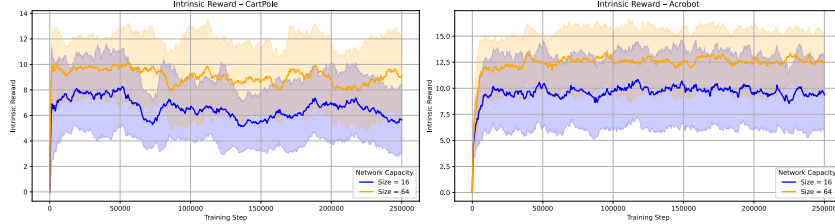


Figure 4: **Effect of predictor and target hidden dimension size on intrinsic reward in CartPole and Acrobot.** Runs were conducted across 5 seeds. The shaded regions represent the standard error across seeds. On the left (**CartPole**), the orange line (network capacity = 64) exhibits higher intrinsic reward values and greater variance ($std_dev = 3.1$), while the blue line (network capacity = 16) results in lower intrinsic rewards with reduced variance ($std_dev = 2.9$). On the right (**Acrobot**), the orange line (network capacity = 64) exhibits higher average intrinsic reward and reduced variance ($std_dev = 3.3$), while the blue line (network capacity = 16) results in lower intrinsic reward with greater variance ($std_dev = 3.8$). Intrinsic reward curves are smoothed using exponential moving averages ($\alpha = 0.1$), with interpolation and forward/backward filling applied prior to aggregation. See hyperparameter settings used in this figure in Appendix A.3.

Contrary to initial expectations that larger predictor and target hidden dimensions would inherently produce greater variance in intrinsic rewards due to richer representational capacity, experimentation revealed that the relationship between network capacity and intrinsic reward dynamics is more complex than previously understood. The updated analysis compares intrinsic reward patterns between **CartPole** and **Acrobot** environments, revealing environment-dependent effects that challenge simple theoretical predictions about network capacity. Specifically, while larger dimensions consistently produced higher magnitude intrinsic rewards across both environments, the variance characteristics showed non-monotonic patterns that defied initial theoretical models.

The dimensionality and structure of the observation space fundamentally affects how RND networks process state information. **CartPole** features a 4-dimensional observation space representing position, velocity, angle, and angular velocity, while **Acrobot** operates in a 6-dimensional space with trigonometric encodings of joint angles and angular velocities. This difference in observation space complexity likely contributes to the varying sensitivity of intrinsic rewards to network capacity changes. Additionally, the relationship between environment complexity and policy entropy may influence how network capacity affects intrinsic reward generation. Environments with different inherent complexity levels, such as the relatively simple **CartPole** versus the more complex **Acrobot**, may exhibit different sensitivity to network architectural choices.

The relationship between predictor and target network capacity and intrinsic reward dynamics appears more nuanced than initially anticipated, with environment-specific factors playing a significant role in determining optimal architectural choices. While larger hidden dimensions consistently produce higher intrinsic reward signals in these experiments, the underlying mechanisms and their implications for exploration effectiveness require further investigation. Future research should expand this analysis to diverse environment types and investigate the mechanistic foundations of these capacity-reward relationships to develop more principled guidelines for RND network design.

6 Conclusions and Future Work

This work explored the integration of Random Network Distillation (RND) into Deep Q-Learning (DQN), providing an empirical analysis of its effects on exploration behaviour and learning performance across a diverse set of environments. The results demonstrate that while RND can substantially improve the agent’s capacity to explore novel or uncertain states, its success is highly sensitive to environment-specific design choices. Hyperparameters such as temperature and reward coefficients have a strong impact on both the stability of learning and the agent’s ability to balance exploration and exploitation. No single set of robust hyperparameters generalized effectively across environments, underlining the need for careful tuning when applying RND in practice. Additionally, the relationship between predictor and target hidden dimension size was analyzed, highlighting some experimental limitations and future lines of research.

Furthermore, the application of RND in the `MNISTBandit` contextual bandit setting demonstrates that its benefits extend beyond sequential decision-making tasks, enabling agents to leverage novelty signals even in single-step environments. Overall, this work provides new insights into the practical challenges and opportunities of combining RND with DQN, and motivates further research into more adaptive and robust integration strategies.

Several open issues remain following this study, both due to practical constraints and as avenues for further research. First, limited compute resources restricted experimentation on complex Atari environments such as `MontezumaRevenge`. Due to the extreme exploration requirements of this environment, achieving meaningful results requires large-scale experiments with significantly longer training horizons. As such, a thorough evaluation of RND’s behaviour in DQN agents coupled with Boltzmann exploration in these settings remains an important area for future work.

Second, while Boltzmann exploration was shown to provide advantages over ϵ -greedy in several environments, its use of a fixed temperature may limit performance in tasks where greedy behaviour becomes more desirable over time. Incorporating a temperature annealing schedule could help balance exploration and exploitation more effectively as learning progresses, particularly in environments that require precise policies for optimal performance. Similarly, the dynamic adjustment of the extrinsic / intrinsic coefficient ratio could be explored as well. This ratio was statically defined in the current experiments, but adaptive schemes could better align the agent’s learning objectives with the demands of different phases of training.

Further investigation into the impact of predictor and target hidden dimension size on intrinsic reward dynamics is warranted, as this relationship appears heavily influenced by environmental characteristics. Additionally, more sophisticated replay buffer strategies should be evaluated to mitigate the potential for outdated exploration incentives, particularly regarding the storage and updating of intrinsic rewards.

7 Responsible Research

This research adheres to principles of transparency, reproducibility, and ethical conduct. All experiments are implemented using openly available tools and environments, and the codebase is based on the CleanRL framework, ensuring minimalism and reproducibility by design. Hyperparameter settings, evaluation metrics, and environment configurations are clearly documented to allow replication of the results. The complete codebase and experiment artifacts are available at <https://github.com/albertomoreno04/EpistemicUncertaintyRL>.

No human data or personal information is used in this project. All experiments are conducted using simulated environments from Gymnasium and Gymnax, with no ethical risks related to data collection or usage. The inclusion of both successful and inconclusive results aligns with good scientific practice, avoiding bias toward positive outcomes and supporting a fair evaluation of the method’s limitations.

In addition, large language models (LLMs) were used during the writing and debugging process. On the coding side, LLMs were used to troubleshoot errors and generate plotting scripts. Whereas on the writing side, LLMs were used for spellchecking, word clarity, and document formatting. These uses were strictly supportive and did not influence the scientific interpretation or originality of the research design. The example prompts can be found in Appendix 5

References

- [1] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
- [2] Aleksa Sukovic and Goran Radanovic. Reward design for justifiable sequential decision-making. In *International Conference on Learning Representations (ICLR)*, 2024.
- [3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [5] Kendrick Tan. Exploration of obstacle tower using random network distillation. 2019. Preprint.
- [6] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 663–670, 2000.
- [7] Simon Schmitt, John Shawe-Taylor, and Hado van Hasselt. Exploration via epistemic value estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

- [8] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.
- [9] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. *Wiley Series in Probability and Mathematical Statistics*, 1994.
- [10] Cart pole - gymnasium documentation. https://gymnasium.farama.org/environments/classic_control/cart_pole/, 2024. Accessed: 2025-06-16.
- [11] Acrobot - gymnasium documentation. https://gymnasium.farama.org/environments/classic_control/acrobot/, 2024. Accessed: 2025-06-16.
- [12] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado Van Hasselt. Behaviour suite for reinforcement learning, 2020.
- [13] Heming Yuan, Tianhe Yu, Jie Cang, Jiacheng Yang, Shanghang Zhang, Guoqing Zhang, and Dale Schuurmans. Automatic intrinsic reward shaping for exploration in deep reinforcement learning. *Proceedings of Machine Learning Research*, 2023.
- [14] Yuhang Wang, Xiangyang Li, Ziyang Chen, and Yifan Zhang. An overview of the action space for deep reinforcement learning. *ACM Transactions on Intelligent Systems and Technology*, 13(4):1–20, 2022.
- [15] Christoph Dann, Yishay Mansour, Mehryar Mohri, Ayush Sekhari, and Karthik Sridharan. Guarantees for epsilon-greedy reinforcement learning with function approximation. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 4694–4720. PMLR, 2022.
- [16] Ling Pan, Qingpeng Cai, Qi Meng, Wei Chen, and Longbo Huang. Reinforcement learning with dynamic boltzmann softmax updates. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2785–2791, 2020.
- [17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2 edition, 2018.
- [18] Marco Wiering and Hado van Hasselt. Comparing exploration strategies for q-learning in random environments. In *Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms*, pages 223–234, 2010.
- [19] Milos Tokic and Günther Palm. Adaptive ϵ -greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence*, pages 203–210. Springer, 2010.
- [20] Baeldung. Epsilon-greedy q-learning. <https://www.baeldung.com/cs/epsilon-greedy-q-learning>, 2025. Accessed: 2025-06-16.
- [21] Farama Foundation. Montezumarevenge - gymnasium documentation. https://gymnasium.farama.org/environments/atari/montezuma_revenge/, 2024. Accessed: 2025-06-21.

- [22] Roberta Raileanu and Tim Rocktäschel. Improving intrinsic exploration by creating stationary objectives. *arXiv preprint arXiv:2310.18144*, 2024.
- [23] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.
- [24] Amr Sharaf and Hal Daumé III. Meta-learning effective exploration strategies for contextual bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9133–9141, 2021.

A Appendix

A.1 RND Pseudo-code

Algorithm 1 describes the training process of the RND-augmented DQN agent using Boltzmann action sampling. The agent collects transitions, computes intrinsic rewards using the predictor and target networks, and updates Q-values and predictor weights through sampled mini-batches. Observation normalization is performed prior to training using a randomly acting agent.

Algorithm 1 RND-DQN with Boltzmann Exploration

- 1: **Input:** Total timesteps T , batch size B , learning rate α , temperature τ , buffer size $|\mathcal{D}|$
 - 2: **Initialize:** Q-network Q_θ , target Q-network $Q_{\bar{\theta}}$, predictor network f_ψ , fixed target network $f_{\bar{\psi}}$
 - 3: **Initialize:** Replay buffer $\mathcal{D}$, observation normalization stats using M random steps
 - 4: **for** $t = 1$ **to** T **do**
 - 5: Observe state s_t
 - 6: **Compute:** Softmax over Q-values $\pi(a|s_t) = \frac{\exp(Q_\theta(s_t, a)/\tau)}{\sum_{a'} \exp(Q_\theta(s_t, a')/\tau)}$
 - 7: Sample action $a_t \sim \pi(\cdot|s_t)$
 - 8: Execute a_t , observe reward r_t^{ext} , next state s_{t+1}
 - 9: **Intrinsic reward:** $r_t^{\text{int}} = \|f_{\bar{\psi}}(\hat{s}_{t+1}) - f_\psi(\hat{s}_{t+1})\|^2$
 - 10: Store transition $(s_t, a_t, r_t^{\text{ext}}, r_t^{\text{int}}, s_{t+1})$ in buffer $\mathcal{D}$
 - 11: **if** $t \geq$ learning start threshold and $t \bmod \text{update frequency} = 0$ **then**
 - 12: Sample mini-batch $\mathcal{B} \subset \mathcal{D}$
 - 13: Compute total reward: $r_t = \beta_{\text{ext}} r_t^{\text{ext}} + \beta_{\text{int}} r_t^{\text{int}}$
 - 14: Compute target: $y_t = r_t + \gamma \max_{a'} Q_{\bar{\theta}}(s_{t+1}, a')$
 - 15: Update Q_θ using TD loss: $\mathcal{L}_{\text{TD}} = (Q_\theta(s_t, a_t) - y_t)^2$
 - 16: Update f_ψ using MSE loss: $\mathcal{L}_{\text{RND}} = \|f_{\bar{\psi}}(\hat{s}_{t+1}) - f_\psi(\hat{s}_{t+1})\|^2$
 - 17: **end if**
 - 18: **if** $t \bmod \text{target update frequency} = 0$ **then**
 - 19: Update target Q-network: $\bar{\theta} \leftarrow \theta$
 - 20: **end if**
 - 21: **end for**
-

A.2 Hyperparameter Sweep Details

CartPole and Acrobot Robustness Comparison

Hyperparameter	Sweep Values
Predictor and Target Hidden Dimension	{16, 64}
Extrinsic / Intrinsic Coefficient Ratio	{0.2, 20.0}
Temperature (τ)	{0.2, 0.5, 1.0}

Table 2: **Hyperparameter sweep values for RND agent in CartPole and Acrobot.**

Hyperparameter	Value
Batch Size	16
Buffer Size	50,000
Learning Rate	0.001
Gamma (γ)	0.999
Total Timesteps	250,000
Predictor Update Frequency	5
Target Network Frequency	200
Seeds	{7, 42, 99, 1337, 2025}

Table 3: **Fixed hyperparameters in CartPole and Acrobot.**

CartPole

Hyperparameter	Sweep Values
Predictor and Target Hidden Dimension	{16, 32}
Extrinsic / Intrinsic Coefficient Ratio	{0.1, 0.2, 0.3}
Temperature (τ)	{0.3, 0.5, 0.7}

Table 4: **Hyperparameter sweep values for RND agent in CartPole.**

Hyperparameter	Sweep Values
Epsilon Start	{0.5, 1.0}
Epsilon Final	{0.01, 0.1}
Exploration Fraction	{0.3, 0.9}

Table 5: **Hyperparameter sweep values for baseline DQN agent in CartPole.**

Hyperparameter	Value
Batch Size	16
Buffer Size	50,000
Learning Rate	0.001
Gamma (γ)	0.999
Total Timesteps	250,000
Predictor Update Frequency	5
Target Network Frequency	200
Seeds	{7, 42, 99, 1337, 2025}

Table 6: **Fixed hyperparameters for all agents in CartPole.**

Acrobot (1)

Hyperparameter	Sweep Values
Predictor and Target Hidden Dimension	{32, 64}
Extrinsic / Intrinsic Coefficient Ratio	{15.0, 20.0, 30.0}
Temperature (τ)	{0.2, 0.3, 0.4}

Table 7: **Hyperparameter sweep values for RND agent in Acrobot (1).**

Hyperparameter	Sweep Values
Epsilon Start	{0.5, 1.0}
Epsilon Final	{0.001, 0.01}
Exploration Fraction	{0.01, 0.03, 0.1}

Table 8: **Hyperparameter sweep values for baseline DQN agent in Acrobot (1).**

Hyperparameter	Value
Batch Size	64
Buffer Size	50,000
Learning Rate	0.001
Gamma (γ)	0.999
Total Timesteps	300,000
Predictor Update Frequency	4
Target Network Frequency	250
Seeds	{7, 42, 99, 1337, 2025}

Table 9: **Fixed hyperparameters for all agents in Acrobot (1).**

Acrobot (2)

Hyperparameter	Sweep Values
Extrinsic / Intrinsic Coefficient Ratio	{2.0, 5.0, 20.0}

Table 10: **Hyperparameter sweep values for RND sweep Acrobot (2).**

Hyperparameter	Value
Batch Size	64
Buffer Size	50,000
Temperature (τ)	0.2
Predictor and Target Hidden Dimension	64
Learning Rate	0.001
Gamma (γ)	0.999
Total Timesteps	300,000
Predictor Update Frequency	4
Target Network Frequency	250
Seeds	{7, 42, 99, 1337, 2025}

Table 11: **Fixed hyperparameters for RND sweep Acrobot (2).**

DeepSea

Hyperparameter	Sweep Values
Environment Size	{ 8, 10, 12 }
Predictor and Target Hidden Dimension	{16, 32}
Extrinsic / Intrinsic Coefficient Ratio	{0.5, 1.0, 2.0}
Temperature (τ)	{0.2, 0.5, 1.0}

Table 12: **Hyperparameter sweep values for RND agent in DeepSea (1).**

Hyperparameter	Sweep Values
Environment Size	{ 8, 10, 12 }
Epsilon Start	{0.7, 1.0}
Epsilon Final	{0.001, 0.01, 0.1}
Exploration Fraction	{0.1, 0.2, 0.3}

Table 13: **Hyperparameter sweep values for baseline DQN agent in DeepSea.**

Hyperparameter	Value
Batch Size	64
Buffer Size	50,000
Learning Rate	0.001
Gamma (γ)	0.999
Total Timesteps	50,000
Predictor Update Frequency	20
Target Network Frequency	200
Seeds	{7, 42, 99, 1337, 2025}

Table 14: **Fixed hyperparameters for all agents in DeepSea.**

A.3 Figure Details

Environment and Training Settings	
Environment Name	CartPole
Batch Size	64
Buffer Size	50,000
Total Timesteps	250,000
RND and Exploration Parameters	
Extrinsic / Intrinsic Coefficient Ratio	0.2
Temperature (τ)	0.7
Predictor and Target Hidden Dimension	16
Predictor Update Frequency	5
Gamma (γ)	0.999
Learning Rate	0.001
Target Network Frequency	200
Logging and Reproducibility	
Seeds	{7, 42, 99, 1337, 2025}
Environment and Training Settings	
Environment Name	Acrobot
Batch Size	64
Buffer Size	50,000
Total Timesteps	300,000
RND and Exploration Parameters	
Extrinsic / Intrinsic Coefficient Ratio	20.0
Temperature (τ)	0.4
Predictor and Target Hidden Dimension	64
Predictor Update Frequency	4
Gamma (γ)	0.999
Learning Rate	0.001
Target Network Frequency	250
Logging and Reproducibility	
Seeds	{7, 42, 99, 1337, 2025}

Table 15: **Hyperparameters used for Table 1**
Hyperparameters are grouped by training settings, RND/exploration parameters, and logging/reproducibility settings.

Environment and Training Settings	
Environment Name	Acrobot
Batch Size	64
Buffer Size	50,000
Total Timesteps	300,000
RND and Exploration Parameters	
Extrinsic / Intrinsic Coefficient Ratio	(15.0, 20.0, 30.0)
Temperature (τ)	0.2
Predictor and Target Hidden Dimension	64
Predictor Update Frequency	4
Gamma (γ)	0.999
Learning Rate	0.001
Target Network Frequency	250
Logging and Reproducibility	
Seeds	{7, 42, 99, 1337, 2025}

Table 16: **Hyperparameters used for Figure 1.**
Hyperparameters are grouped by training settings, RND/exploration parameters, and logging/reproducibility settings.

Environment and Training Settings	
Environment Name	DeepSea
Batch Size	64
Buffer Size	50,000
Total Timesteps	50,000
RND and Exploration Parameters	
Extrinsic / Intrinsic Coefficient Ratio	2.0
Temperature (τ)	(0.2, 0.5, 1.0)
Predictor and Target Hidden Dimension	(16, 32)
Predictor Update Frequency	20
Gamma (γ)	0.999
Learning Rate	0.001
Target Network Frequency	200
Logging and Reproducibility	
Seeds	{7, 42, 99, 1337, 2025}

Table 17: **Hyperparameters used for Figure 2.**
Hyperparameters are grouped by training settings, RND/exploration parameters, and logging/reproducibility settings.

Environment and Training Settings	
Environment Name	MNISTBandit
Batch Size	64
Buffer Size	50,000
Total Timesteps	1,000,000
RND Parameters	
Extrinsic / Intrinsic Coefficient Ratio	4.0
Temperature (τ)	0.2
Predictor and Target Hidden Dimension	512
Predictor Update Frequency	45
Target Network Frequency	100
Baseline Parameters	
Epsilon Start	1
Epsilon Final	0.1
Exploration Fraction	0.2
Target Network Frequency	50
Exploration Parameters	
Gamma (γ)	0.999
Learning Rate	0.0001
Logging and Reproducibility	
Seeds	{7, 42, 99, 1337, 2025}

Table 18: **Hyperparameters used for Figure 3.**

Hyperparameters are grouped by training settings, RND parameters for the RND runs, baseline parameters for the baseline runs, exploration parameters, and logging/reproducibility settings.

Environment and Training Settings		
Environment Name	CartPole	Acrobot
Batch Size	16	16
Buffer Size	50,000	50,000
Total Timesteps	250,000	250,000
RND and Exploration Parameters		
Extrinsic / Intrinsic Coefficient Ratio	0.2	20.0
Temperature (τ)	1.0	0.2
Predictor and Target Hidden Dimension	(16, 64)	(16, 64)
Predictor Update Frequency	5	5
Gamma (γ)	0.999	0.999
Learning Rate	0.001	0.001
Target Network Frequency	200	200
Logging and Reproducibility		
Seeds	{7, 42, 99, 1337, 2025}	

Table 19: **Hyperparameters used for Figure 4.**

Hyperparameters are grouped by training settings, RND/exploration parameters, and logging/reproducibility settings.

A.4 Additional Experimentation Results

Policy entropy is computed during training using:

$$H = - \sum_{a=1}^N p(a) \log(p(a) + \epsilon) \quad (7)$$

where $\epsilon = 10^{-12}$ prevents numerical instability, N is the action space size, and $p(a)$ represents the mean empirical action probability across training batches.

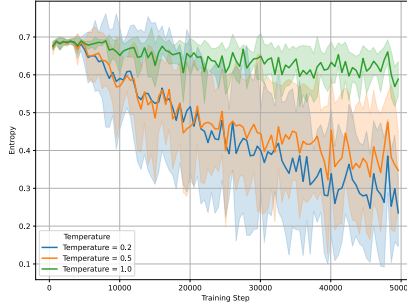


Figure 5: **Comparison of entropy of the agent’s policy across training between different values of temperature** Runs were conducted across 5 seeds in DeepSea for environment size 10. The shaded regions represent the standard error across seeds. Higher temperatures (green line, $\tau = 1.0$) lead to higher entropy with lower variance. Lower temperatures (blue line, $\tau = 0.2$) result in lower entropy but larger variance, as indicated by the wider shaded region. Each line is smoothed using a moving average with interpolation and forward/backward fill applied at missing steps for alignment. See hyperparameter settings used in this figure in Appendix A.3.

Figure 5 illustrates how temperature affects the entropy of the agent’s policy. In the results, higher temperatures (green line, $\tau = 1.0$) result in higher entropy with lower variance across training. Conversely, lower temperatures (blue line, $\tau = 0.2$) lead to lower average entropy but greater variance, as reflected by the wider shaded region. As theoretically expected, this result highlights the trade-off introduced by the temperature parameter τ : while higher temperatures promote consistent exploratory behaviour, they can also destabilize learning if they prevent the agent from sufficiently exploiting learned value estimates.

Figure 6 shows the number of unique contexts visited over time for both RND and baseline agents in the MNISTBandit environment. The RND agent encounters a more diverse set of inputs, supporting the idea that intrinsic rewards and boltzmann exploration encourage broader exploration.

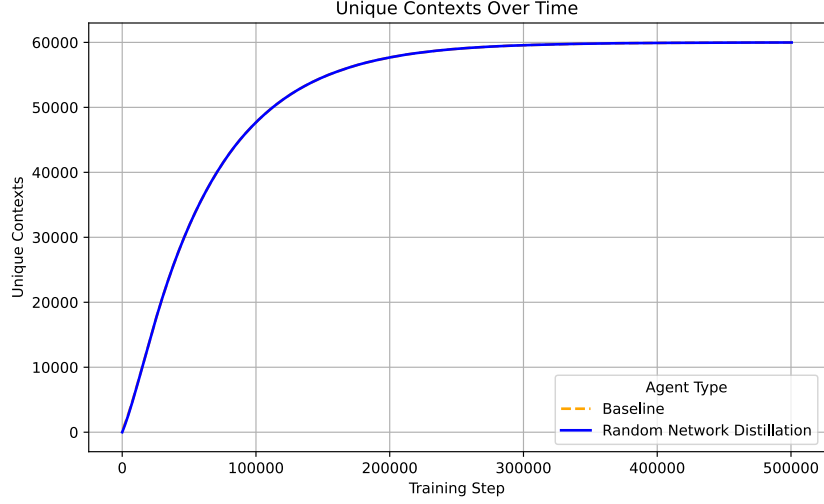


Figure 6: **Unique context comparison in MNISTBandit contextual bandit environment.** Runs conducted across 5 different seeds, comparing RND-augmented DQN and baseline DQN agents. The overlapping lines indicate that both agents encounter a similar number of unique contexts over time. This does not imply they visit the same specific contexts, but rather reflects the finite size of the MNIST dataset and the shared data sampling process.

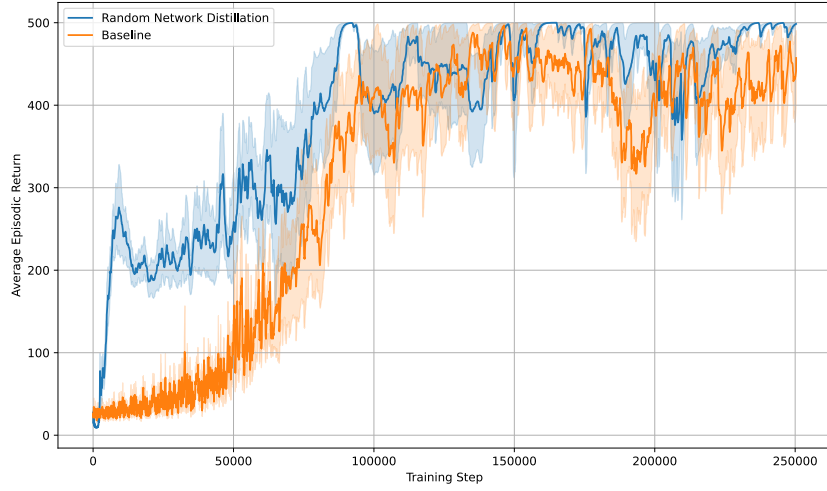


Figure 7: **Comparison of average episodic return between RND and baseline agents in CartPole.** Runs were conducted across 5 seeds. The shaded regions represent the standard error across seeds. The blue line (RND agent) shows faster learning, while the orange line (baseline agent) takes longer to learn and shows more instability. Smoothed using exponential moving average ($\alpha = 0.1$) for both mean and standard error of episodic returns.

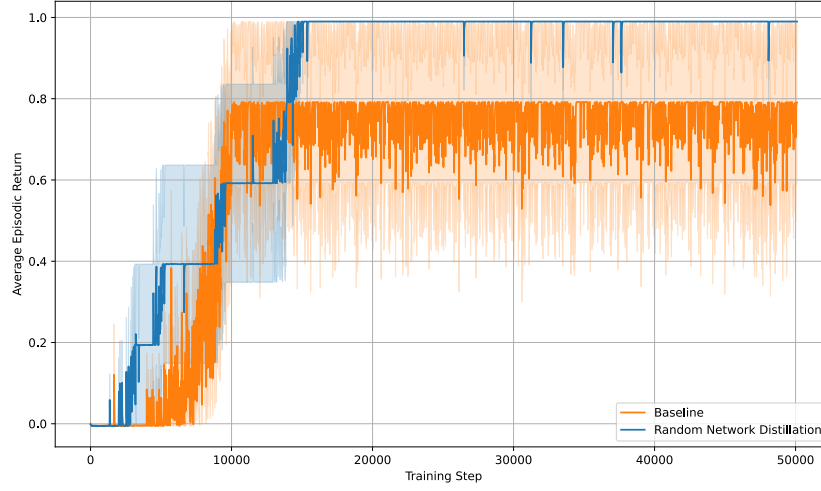


Figure 8: **Comparison of average episodic return between RND and baseline agents in DeepSea.** Runs were conducted across 5 seeds. The shaded regions represent the standard error across seeds. The blue line (RND agent) shows more stability and convergence, while the orange line (baseline agent) shows more instability and, on average does achieve optimal behaviour. Smoothed using exponential moving average ($\alpha = 0.1$) for both mean and standard error of episodic returns.

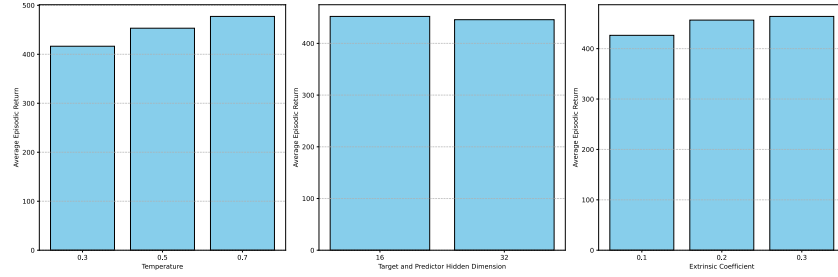


Figure 9: **Average episodic return for given values of hyperparameter found in CartPole hyperparameter sweep.** Every hyperparameter setting was ran across 5 seeds. On the left, temperatures are compared, the highest average episodic return was achieved by $\tau = 0.7$. There is a small difference on the average episodic return between different hidden dimension sizes, where $size = 16$ slightly outperforms $size = 32$, as seen in the plot on the middle. Lastly, the plot on the right compares extrinsic coefficients, where the value of 0.3 outperformed the rest.

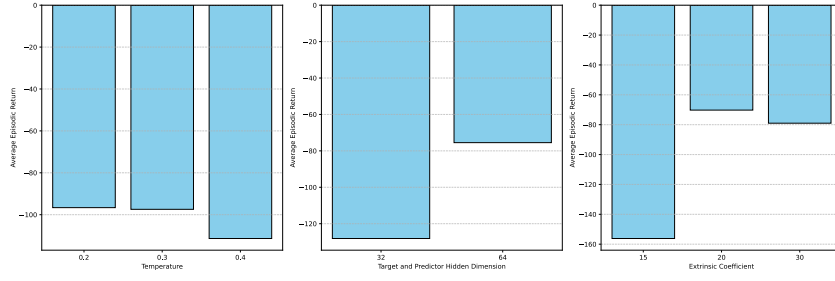


Figure 10: **Average episodic return for given values of hyperparameter found in Acrobot hyperparameter sweep.** Every hyperparameter setting was ran across 5 seeds. On the left, temperatures are compared, the highest average episodic return was achieved by $\tau = 0.4$. There is a difference on the achieved average episodic return between different hidden dimension sizes, where $size = 32$ outperformed $size = 16$, as seen in the plot on the middle. Lastly, the plot on the right compares extrinsic coefficients, where the value of 15.0 outperformed the rest.

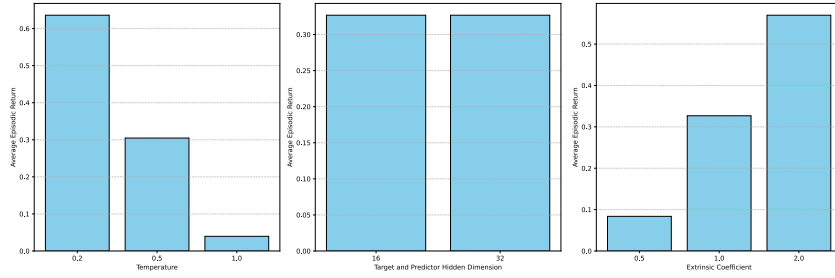


Figure 11: **Average episodic return for given values of hyperparameter found in DeepSea hyperparameter sweep.** Every hyperparameter setting was ran across 5 seeds. On the left, temperatures are compared, the highest average episodic return was achieved by $\tau = 0.2$. There is no difference on the achieved average episodic return between different hidden dimension sizes, where $size = 16$ performs as well $size = 32$, as seen in the plot on the middle. Lastly, the plot on the right compares extrinsic coefficients, where the value of 2.0 outperformed the rest.

A.5 Example Prompts and Responses for LLM Usage

Prompt (1)

I have two CSV files. The first one (meta.csv) contains columns Name and agent_type. The second one (metrics.csv) has columns Step and episodic_return, with column names formatted like run_name1 - episodic_return, run_name1 - episodic_return_MIN, run_name1 - episodic_return_MAX, run_name2 - episodic_return, etc. Each agent belongs to either the baseline or rnd group, as defined in meta.csv. Can you help me write a script to group agents by agent_type and plot their average episodic return over time, with separate lines for each group and a shaded area for standard deviation?

Response (1)

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict

# Load metadata
meta_df = pd.read_csv("meta.csv")
name_to_type = dict(zip(meta_df["Name"].str.strip().str.lower(), meta_df["agent_type"]))

# Load episodic returns
metrics_df = pd.read_csv("metrics.csv")
step_col = "Step"
steps = metrics_df[step_col].to_numpy()
agent_data = defaultdict(list)

# Process each return column
for col in metrics_df.columns:
    if col == step_col or "__" in col or not col.endswith("episodic_return"):
        continue
    agent_name = col.split(" - ")[0].strip().lower()
    agent_type = name_to_type.get(agent_name)
    if agent_type is None:
        continue
    values = metrics_df[col].dropna().to_numpy()
    agent_data[agent_type].append(values)

# Compute mean and std per agent_type
plot_data = {}
for agent_type, series in agent_data.items():
    min_len = min(len(s) for s in series)
    trimmed = np.array([s[:min_len] for s in series])
    mean = trimmed.mean(axis=0)
    std = trimmed.std(axis=0)
    plot_data[agent_type] = (steps[:min_len], mean, std)

# Plot
plt.figure(figsize=(10, 6))
colors = {"rnd": "blue", "baseline": "orange"}
for agent_type, (x, y, s) in plot_data.items():
    plt.plot(x, y, label=agent_type.capitalize(), color=colors.get(agent_type, None))
    plt.fill_between(x, y - s, y + s, alpha=0.2, color=colors.get(agent_type, None))

plt.xlabel("Training Step")
plt.ylabel("Episodic Return")
plt.title("Average Episodic Return by Agent Type")
plt.legend()
plt.grid(True)
plt.tight_layout()

```

```
plt.savefig("episodic_return_plot.svg")
```

Prompt (2)

```
\bibliographystyle{plain}  
\bibliography{references}
```

Why is my bibliography not shown in order of appearance in the document?

Response (2)

The reason your bibliography is not shown in order of appearance is because you're using the `plain` bibliography style, which sorts entries alphabetically by author name rather than in the order they are cited.

To display the bibliography in citation order, change your style to `unsrt`:

```
\bibliographystyle{unsrt}  
\bibliography{references}
```

The `unsrt` style (short for “unsorted”) ensures that references appear in the same order as they are cited in the text, which is the standard behavior in many scientific publications.