Smarter Moves: Enhancing the Exploration Method of MuZero

Action Selection in Model-Based Reinforcement Learning

Francisco Ruas Vaz[1]
Supervisor(s): Dr. Frans A. Oliehoek[1], **Dr. Jinke He**[1]
[1]EEMCS, 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: Francisco Ruas Vaz
Final project course: CSE3000 Research Project
Thesis committee: Dr. Frans A. Oliehoek, Dr. Jinke He, Dr. Michael Weinmann

Abstract

MuZero is a state-of-the-art reinforcement learning algorithm developed by DeepMind. This artificial intelligence program achieves superhuman performance in complex domains, the most noteworthy being popular board games and Atari games. Reinforcement learning agents, like MuZero, depend critically on effective exploration to discover optimal decision-making strategies. In MuZero, this is done with a standard exploration mechanism based on Monte Carlo Tree Search visit counts and a temperature-controlled softmax action selection. This thesis investigates the potential for enhancing MuZero's performance, sample efficiency, and learning trajectory by systematically evaluating alternative exploration strategies. Our research explores several modifications to the action selection process. This involves varying the scheduling of temperature for softmax selection, an epsilon-greedy action selection and using Thompson Sampling with an ensemble of models for the exploration step. These methods provide a robust approach to balancing exploration and exploitation, and have already been applied to other programs similarly. This paper contributes a comprehensive empirical comparison of these strategies, providing insights into their practical implications for optimising MuZero and advancing the understanding of exploration in complex model-based RL agents.

1 Introduction

Reinforcement learning (RL) algorithms face the fundamental challenge of exploration, deciding which actions to take in order to discover informative or rewarding parts of the environment. Striking the right balance between exploration and exploitation is fundamental for efficient learning, particularly in environments where rewards are sparse, deceptive, or delayed.

MuZero is a powerful RL algorithm that combines model learning with lookahead planning. This learning process often involves extensive trial and error. A significant subfield of RL, Model-Based Reinforcement Learning (MBRL), endeavours to enhance the sample efficiency of this process by enabling the agent to learn a model of the environment's dynamics. Such a model allows the agent to simulate experiences and plan, thereby reducing the need for costly real-world interactions. Rather than relying on an explicit model of the environment's transitions or rewards, MuZero learns a latent dynamics model from experience and uses Monte Carlo Tree Search (MCTS) to simulate future outcomes and guide decision-making. This architecture has led to impressive results in games such as Go, Chess, and Atari, where deep planning and long-term reasoning are essential.

While much of MuZero's effectiveness comes from its use of MCTS to evaluate and plan over action sequences, how the agent selects which action to take in the actual environment, is equally important. After each planning step, the agent must choose an action based on the search results, typically using the visit counts of each action explored in MCTS. The standard practice is to either select the action with the highest visit count or to sample actions proportionally to their visit frequency.

This project explores alternative strategies for this decision step, investigating how the method of selecting actions for real environment interaction impacts learning performance. We compare different action selection strategies: annealing temperature, epsilon-greedy exploration and Thompson Sampling. These approaches will vary in how they encourage trying less-certain or less-explored actions. Therefore, offering an informative sample of techniques to experiment with.

By systematically evaluating these strategies, this work aims to explore their effect on learning dynamics in MuZero and contribute practical insights into how simple changes in action selection policy can influence long-term performance and data efficiency.

The paper is structured in the following manner. Section 2 provides a review of the exploration in the MuZero algorithm, a survey of relevant exploration strategies, and a discussion of related work in modifying MuZero. Section 3

details the methodology employed, including the baseline MuZero implementation and the specific modifications made to its exploration strategy. Section 4 outlines the experimental setup, including the environments, hyperparameters and evaluation metrics. Section 5 presents the empirical results of the experiments as well as a thorough analysis of these findings. Section 6 concludes the thesis by summarising the key findings and suggesting avenues for future research. Finally, Section 7 addresses the ethical and societal implications of the research.

2 Background and Related Work

2.1 Background

Exploration remains a critical problem in model-based RL, especially in environments with sparse or deceptive reward signals, which drastically increase errors in uncertainty estimation. An agent must efficiently seek out informative experiences while avoiding premature convergence to suboptimal behaviours. In MBRL, where agents learn a model of the environment to plan actions, effective exploration is even more challenging, since planning through a learned model introduces risks, compounding model errors. This can sway the agent toward familiar states, limiting generalisation through over-fitting and causing a self-fulfilling cycle of exploiting suboptimal actions.

MuZero is a leading MBRL algorithm that achieves high performance in complex domains by combining model learning with planning. First, it learns a dynamics model in a latent space, which predicts the next latent state and immediate reward using only the current state and action. Then, MCTS is used to simulate possible action sequences and estimate their expected outcomes, thereby forming a 'plan' of which action the model should take. This groundbreaking implementation allows MuZero to learn and play various games at superhuman levels [10]. Before MuZero came several iterations of this concept, beginning with AlphaGo, followed by AlphaGo Zero and AlphaZero. With each iteration, the emphasis was to remove a significant aspect of its knowledge domain and prompt it to naturally re-learn this information. Within all of these models, the output of MCTS typically embodies a distribution of visit counts over actions, from which the agent selects an action to execute in the environment. Traditionally, MuZero samples actions proportionally to their visit counts, which can implicitly balance exploration and exploitation through the Upper Confidence Bound (UCB) selection. At a high level, UCB sums the average mean reward at a given state with an exploration bonus, a term which increases in size when a certain action has not been explored much. More specifically, MuZero leverages the estimated action-value at the current state (the exploitative function), with the prior predicted probability of an action based on the policy network (the exploration bonus).

Once the sample of actions is obtained from the MCTS, MuZero must then deploy a function to finalise the action-selection step. Notably, this choice of function for selecting actions has not been thoroughly studied in MuZero. Since this selection mechanism directly affects the data the agent collects and learns from, it plays a vital role in shaping its policy. Alternative selection strategies could improve learning speed, policy quality, or robustness, especially in environments where MuZero's default behaviour is either too conservative or overconfident.

Thus, this project investigates and compares several simple but effective action selection strategies applied during the exploration stage. Specifically, we implement temperature annealing strategies, epsilon-greedy selection and Thompson Sampling for exploration. These methods are lightweight to implement and allow a focused study on how action selection during environment interaction influences overall agent performance.

2.1.1 MuZero

MuZero represents the culmination of a series of breakthroughs in deep reinforcement learning, evolving from AlphaGo, to AlphaGo Zero , and AlphaZero [12, 14, 13]. As explained before, each iteration progressively removed reliance on human knowledge. Starting with AlphaGo, which used a complete dataset of human expert games, AlphaGo Zero built

on this by learning entirely from self-play, but was provided the rules of the game. Lastly, AlphaZero generalised this approach to multiple games. MuZero marked the final, significant step in this progression by removing knowledge of the environment's dynamics entirely. The algorithm learns its latent model of the world and uses it to plan, achieving a new level of generalisation [10].

MuZero's architecture is built around three tightly coupled networks: representation, dynamics and prediction. Together, they enable both learning and planning without explicit environment models. The representation network encodes observations into a compact latent state, capturing all information required for decision making. Given this state, as well as a candidate action, the dynamics network predicts the subsequent latent state and an immediate reward, effectively simulating one step of the environment. Finally, the prediction network maps any latent state to a policy and a value estimate. During training, these networks are updated jointly: real interactions provide targets for representation and prediction, while imagined trajectories, generated by iteratively applying dynamics and prediction, are used to sharpen planning. In practice, MuZero begins by encoding the current observation, then alternates between dynamics and prediction to explore future possibilities, and uses the resulting policy and value assessments to select actions. At each timestep, the MCTS algorithm iteratively simulates future trajectories by repeatedly selecting actions and using the dynamics function to transition to subsequent latent states. The policy and value predictions from the prediction function are used to guide this search. After a fixed number of simulations, the final policy for the original state is calculated from the visit counts of the actions in the search tree. This process of self-play generates data, which is then used to train the three neural networks. A key innovation, known as Reanalyse, allows MuZero to efficiently learn from past data by re-running MCTS with a more mature network, improving sample efficiency [11]. Evidently, action selection with MCTS is a core component of MuZero, which can yield significant performance benefits if improved.

2.1.2 Temperature Annealing

A sub-element of MuZero's algorithm is the use of temperature to guide agent exploration. In general, RL algorithms apply the use of temperature in their training to better adjust the model to its natural learning process. The agent is initially quite uncertain of its actions and their results. therefore it is encouraged, via this temperature, to be more risky and try options that it is unsure about. As the model improves, it becomes more aware of the dynamics of the environment. As such, it is expected that over time, the algorithm should be more risk-averse and pick the actions it believes will be the most optimal.

Temperature annealing is a strategy which leverages this assumption, where the temperature is systematically decreased over the course of training. Initially, the agent's knowledge is limited, so a high temperature promotes broad exploration of the state-action space. As the agent gains experience and its policy improves, the temperature is lowered, causing the agent to increasingly exploit its learned knowledge and select what it believes to be the optimal actions. This mimics a natural learning process, moving from experimentation to confident execution. This technique is explicitly mentioned as part of the MuZero self-play process [10].

When selecting an action during self-play, the visit counts from the MCTS, denoted by $N(\text{root}, a)$, are used to form a probability distribution, via a softmax function, shown in Equation 1.

$$P(a|\text{root}) = \frac{N(\text{root}, a)^{1/T}}{\sum_b N(\text{root}, b)^{1/T}} \tag{1}$$

Clearly, a higher temperature ($T > 1$) flattens this distribution, making the selection of actions more uniform and thus encouraging exploration. A lower temperature ($T < 1$) sharpens the distribution, making the agent more likely to select the action with the highest visit count, thus favouring exploitation.

2.1.3 Thompson Sampling

Thompson Sampling, first proposed by William Thompson [16], is a Bayesian approach to balancing exploration and exploitation. Instead of maintaining a single estimate for the value of an action, it establishes a posterior probability distribution over that value. To select an action, it draws a single sample from each action's posterior distribution and chooses the action corresponding to the highest sampled value. This method provides an elegant and efficient form of exploration. Actions with high uncertainty, and in turn have a wide posterior distribution, have a chance of producing a high-value sample, encouraging their selection. As an action is selected more frequently, its posterior distribution converges to its true mean value. This property of shifting probabilities naturally reduces exploration for actions that are well understood.

In the context of RL, this idea has been formalised in algorithms like Posterior Sampling for Reinforcement Learning (PSRL), where an agent samples a complete Markov Decision Process (MDP) from a posterior distribution at the start of each episode and then follows the optimal policy for that sampled MDP [7]. This concept has also been integrated directly into MCTS, where Thompson Sampling can be used to select actions within the search tree instead of the standard UCB1 formula [1]. This project will investigate applying Thompson Sampling at the root of the MCTS, using the visit counts to inform a posterior distribution over the action values.

2.2 Related Work

Prior research has explored various enhancements to MuZero's planning and representation learning components. for instance, approximate Thompson Sampling with neural networks has been studied as a scalable way to encode uncertainty [8]. Optimistic exploration via bootstrapped ensembles has also shown promise. However, little work has explicitly compared action selection mechanisms at the root of the MCTS tree.

Other research has sought to understand the nature of the model that MuZero learns. For instance, He et al. empirically demonstrated that MuZero learns a value-equivalent latent model, which is sufficient for planning but may not accurately simulate the true environment dynamics over long horizons [5]. This finding highlights the importance of robust exploration, as the agent's planning is constrained by an imperfect, albeit highly effective, internal model.

The application of Thompson Sampling in complex, high-dimensional settings has also been an active area of research. Since maintaining exact posterior distributions is often intractable with deep neural networks, various approximation methods have been developed. Bootstrapped Thompson Sampling uses an ensemble of models trained on bootstrapped data to approximate a posterior distribution, a technique that has been successfully applied to deep RL [9]. More recent work, like Ensemble++ (Li et al., 2024), proposes more scalable ensemble methods to make Thompson Sampling practical for large-scale problems.

Despite this extensive body of work, there has been limited research that directly and systematically compares different action selection mechanisms at the root of the MCTS tree in MuZero. While many papers acknowledge the use of temperature-based sampling, the performance implications of this choice compared to other well-established exploration strategies like epsilon-greedy or Thompson Sampling have not been thoroughly benchmarked. This thesis addresses this specific gap by providing a direct, empirical comparison of these methods, aiming to isolate the impact of the final action selection strategy on the overall performance of the MuZero agent.

3 Methodology

This section expands on the details of the experimental methods used to investigate the impact of different exploration strategies on the performance of the MuZero algorithm. We begin by outlining the baseline MuZero configuration in

Section 3.1, which serves as a control for our experiments. Subsequently, we describe the specific modifications made to the action selection mechanism to implement three distinct classes of exploration strategies. Firstly, Section 3.2 describes the use of epsilon-greedy, a simple method commonly used in RL. Section 3.3 discusses experimenting with different temperature step schedules and Section 3.4 temperature scheduling functions. Finally, Section 3.5 provides a description for implementing Thompson Sampling for action selection on the MuZero architecture.

In Section 1 we introduced MuZero and suggested the research question explored in this paper. The core objective is to isolate the effect of the action selection strategy. Therefore, all other components of the MuZero algorithm, including the neural network architecture, the MCTS simulation, and the training procedure, were held constant across all experiments. This controlled approach ensures that any observed differences in performance can be confidently attributed to the exploration strategy under investigation. In Section 4, we detail how experiments were set up and note the specific environments and parameters used to conduct these comparisons. These experiments involve comparing the performance of different exploration strategies across multiple Atari environments, chosen for their varying complexity and reward sparsity. This allows for a robust evaluation of how each method performs when faced with distinct exploration challenges.

3.1 Baseline MuZero

The baseline for this study is the standard MuZero algorithm as described by the team from DeepMind [10]. The implementation is based on the popular mctx library [3] and the flashbax library [17], which faithfully reproduces the core components of the original work. In this configuration, action selection at the root of the search tree is performed after running MCTS for a fixed number of simulations. The visit counts $N(\text{root}, a)$ for each available action a from the root state are recorded. As shown previously in the equation (1), an action is then sampled from the probability distribution generated by applying a temperature-controlled softmax function to these visit counts, where T is the temperature parameter. In the implementation we used, the default temperature annealing schedule is a discrete 3-step process. For the first 50% of training steps $T = 1$, then $T = 0.5$ for the next 25% of training steps, and finally $T = 0.25$ for the remainder of training.

The goal is to isolate the impact of the exploration strategy itself. Therefore, a correct implementation and validation of the baseline's performance are important prerequisites for drawing reliable conclusions about the strategies under investigation.

3.2 Epsilon Greedy Action Selection

As a well-established and simple alternative to softmax exploration, we implemented an epsilon-greedy action selection strategy. This method, foundational in reinforcement learning [15], provides a clear and direct way to balance exploration and exploitation. After the MCTS process concludes, the greedy action is identified as the one with the most visits, $a^* = \text{argmax}_a N(\text{root}, a)$. The action selection policy is then defined as:

$$a_t = \begin{cases} \text{a random legal action} & \text{with probability } \epsilon \\ a^* & \text{with probability } 1 - \epsilon \end{cases} \tag{2}$$

A key distinction of this method is its interaction with MCTS. While the MCTS search identifies a good action based on its deep planning, the epsilon-greedy mechanism may disregard this recommendation entirely in favour of a random action. This contrasts with softmax sampling, where every action has a non-zero selection probability weighted by its MCTS visit count. We experimented with a constant epsilon value throughout training to establish a straightforward

alternative for exploration. While seemingly less sophisticated than annealed softmax, evaluating this strategy allows us to explore whether a simple, untuned exploration mechanism can be competitive in certain environments.

3.3 Temperature Scheduling Step Variations

To investigate the sensitivity of MuZero to the specific annealing schedule, we designed and evaluated several alternative discrete step-based schedules for the softmax temperature T. These variations alter the timing and magnitude of the temperature changes, allowing for different exploration dynamics.

3.3.1 Alternative 3-Step Temperature Schedule

One of the alternative schedules was designed to test a different exploration profile, starting with a higher initial temperature and ending with a similar final temperature after a sharp drop. The schedule is as follows:

- Steps 0-60%: $T = 1.5$
- Steps 60-80%: $T = 0.5$
- Steps 80-100%: $T = 0.25$

3.3.2 Gentler Alternative 3-Step Temperature Schedule

This schedule provides a gentler drop in temperature, but still maintains a higher level of exploration for longer before committing to exploitation. The schedule is as follows:

- Steps 0-50%: $T = 1.25$
- Steps 50-75%: $T = 0.5$
- Steps 75-100%: $T = 0.25$

3.3.3 5-Step Temperature Schedule

A 5-step temperature schedule was designed to provide finer-grained control over the annealing process, compared to the default 3-step schedule. This schedule involves more intermediate temperature values, potentially allowing for a smoother transition from exploration to exploitation. The schedule is as follows:

- Steps 0-20%: $T = 1.5$
- Steps 20-40%: $T = 1.0$
- Steps 40-60%: $T = 0.75$
- Steps 60-80%: $T = 0.5$
- Steps 80-100%: $T = 0.25$

3.4 Temperature Annealing Strategies

To move beyond discrete steps, three continuous annealing strategies were implemented, inspired by common practices in optimisation such as simulated annealing. For each strategy, $T_{initial}$ is the starting temperature, $T_{initial}$ is the target minimum temperature, and N_{decay_steps} is the total number of training steps over which the temperature anneals from $T_{initial}$ to T_{final} .For the rest of the run, $T_t = T_{final}$.

- Linear Decay: The temperature decreases at a constant rate.

$$T_t = T_{initial} - (T_{initial} - T_{final}) \times \frac{t}{N}$$

- Exponential Decay: The temperature decreases multiplicatively, leading to a faster reduction initially, which then slows down.

$$T_t = T_{final} + (T_{initial} - T_{final}) \times e^{-\gamma \times t}$$

- Cosine Decay: The temperature follows the shape of a cosine curve, providing a smooth annealing profile.

$$T_t = T_{final} + \frac{1}{2}(T_{initial} - T_{final})(1 + \cos\left(\frac{\pi \times t}{N}\right))$$

These continuous strategies offer a different paradigm compared to the abrupt changes of the discrete step-decay approach, potentially yielding more stable learning dynamics. The tuning parameter of each schedule is also crucial. While all schedules decay over a similar portion of the total training budget and between the same $T_{initial}$ and T_{final} values, we further explore this in Section 4.2. For a reliable comparison, these parameters were chosen consistently between the multiple environments.

3.5 Thompson Sampling

As a more sophisticated, principled exploration strategy, we implemented Thompson Sampling (TS), a Bayesian approach for balancing exploration and exploitation. In our context of deep reinforcement learning, where maintaining an exact posterior distribution over model parameters is intractable, we use an ensemble-based approximation [6]. Instead of relying on the uncertainty estimates of a single model, this method leverages the disagreement among the ensemble of learned models.

Our implementation modifies the MuZero architecture by having 5 separate models used in combination. The ensemble of models is initialised with the same architecture but with different random weight initialisations. These models are trained in parallel on data collected into a shared replay buffer. While the models learn from the same pool of experience, their differing initialisations cause their learned parameters to diverge, leading each to develop a unique understanding of the environment's dynamics.

To integrate this TS-based approach with MuZero, we randomly choose one model for each action selection, and the agent explores based on the distribution of policies learned by the different models in the ensemble. When MuZero has to select an action, we sample one of the models, and the parameters of only this selected model are then used for the MCTS search. During the learning phase of the agent, it also makes use of a replay buffer. For the TS implementation, each model is trained on the same batch sampled from the replay buffer and are updated separately according to their own understanding. Then priorities for the replay buffer are updated based on the calculations from all models. During evaluation, the agent combines the different models and uses their averaged predictions for its action selection policy. By averaging the predictions of all models, we get a more stable estimate of the best action, reducing the risk of a single model making a poor decision.

4 Experimental Setup

This chapter details the configuration of our experiments, including the environments in Section 4.1, the specific parameters chosen for each exploration strategy in Section 4.2, and the tools and metrics used for training and evaluation in Section 4.3. All experiments were built upon a JAX-based MuZero implementation, making use of the mctx library for MCTS and the flashbax library for the replay buffer. All modifications were confined to the action selection modules described in Chapter 3.

4.1 Environment

To test the exploration strategies under diverse conditions, we selected three classic Atari 2600 games from the Gymnasium library. These environments were chosen for their varying complexity and reward structures, presenting distinct exploration challenges.

We first ran initial experiments with Pong, the easiest of the three environments, for hyperparameter tuning and testing the correctness of our implementations. Pong is a simple two-player game with a small action space and dense rewards. A reward of $+1$ is given for winning a point and -1 for losing one. Success depends more on immediate, reactive policy learning than on long-term planning. Then we tried the Breakout environment, which requires more strategic play. The agent receives frequent rewards for destroying bricks, but efficient play involves learning to tunnel through the brick wall to hit bricks at the top for higher scores, which requires short-term planning. Our final environment is Ms. Pac-Man. This game is a significantly more complex environment featuring a larger state space, multiple non-stationary adversaries (the ghosts), and sparse rewards obtained from eating pellets and power-ups. Success demands effective long-term planning and a robust exploration strategy to navigate the maze efficiently without getting trapped.

Across all environments, general MuZero training hyperparameters were kept constant to ensure a fair comparison. The most significant of these include the use of the Adam optimiser, a total of $100,000$ environment steps, 50 MCTS simulations per turn, and a replay buffer capacity of $100,000$ transitions. A complete list of the hyperparameters can be found in the Appendix.

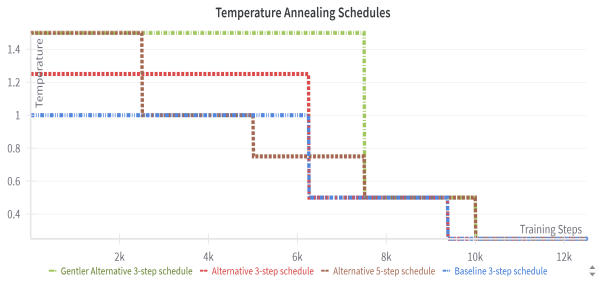
4.2 Exploration Strategy Parameters

Proper hyperparameter selection for each exploration strategy is crucial. However, given that training a single MuZero agent is computationally intensive, an exhaustive grid search was infeasible due to time constraints and the queue-based nature of the available computing clusters.

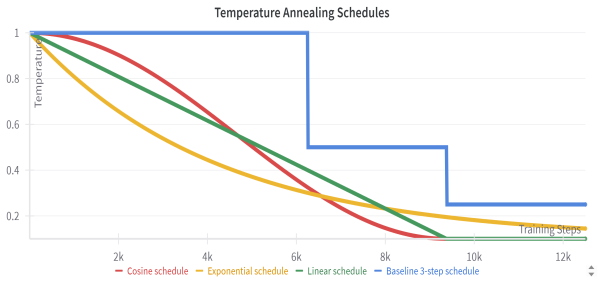
Therefore, a more pragmatic approach was used. Initial, single-run experiments were conducted on the simplest environment, Pong, to identify promising parameters for each strategy. The best-performing parameters from this initial phase were then selected for the final, more rigorous experiments, where each strategy was run for 3 different seeds on all three environments. The specific seed values used are 0, 42 and 2025. While this means parameters were not individually optimised for Breakout and Ms. Pac-Man, it ensures a consistent and fair comparison across environments. The primary metric for parameter tuning was the final mean episodic return after a run.

For the epsilon greedy strategy, we tested two different constant values for ϵ, 0.2 and 0.5. As stated previously, three alternative step-decay temperature schedules were used, described in Section 3.3. The temperature annealing functions, which are named in Section 3.4, were initially tuned by comparing a single constant seed run of Pong, using different parameters. For these continuous functions, we observed that varying the starting and final temperatures slightly had little effect on the outcome, and more importantly, it made direct comparisons between the functions less reliable. Therefore, to ensure a fair comparison, all continuous schedules were tested using the same starting temperature $T_{initial} = 1.0$ and final temperature $T_{final} = 0.1$ and over the same decay periods (or decay rate, used in

the exponential function). In the figures below you can better visualise the different temperature step schedules and temperature decay functions.



(a) Temperature Step Schedules



(b) Temperature Annealing Schedules

4.3 Training and Evaluation

Experiments were conducted using high-performance computing resources, primarily the Delft High Performance Computing Centre (DHPC) [4] and an instance from Vast.ai [18], to manage the significant computational load. The specific hardware configurations of DHPC include a *AMD EPYC 7402 24C 2.80 GHz* CPU with *NVIDIA Tesla V100S 32GB* GPU and *Intel XEON E5-6448Y 32C 2.1GHz* CPU with *NVIDIA Tesla A100 80GB* GPU. The Vast.ai instance uses a *AMD EPYC 7643 48-Core Processor* CPU with a *A100 SXM4* GPU.

All training metrics and results were logged and visualised using the Weights & Biases platform [2]. This tool was convenient for tracking agent performance in real-time and comparing metrics such as MCTS search values, policy entropy, loss components, episode returns and other values across different runs.

The computational cost varied significantly between methods. A typical run for the baseline, and most other experiments, took approximately 4-5 hours to complete. As expected, the Thompson Sampling agent, which trains a 5-model ensemble, took nearly five times as long, requiring 18-20 hours per run.

To ensure the reliability of our results, each final experiment was repeated across 3 different random seeds. The primary metric for evaluation is the mean total reward per episode, averaged over the 3 seeds and plotted against the number of environment steps. We also analyse sample efficiency and the stability of learning, as indicated by the variance in performance across the different random seeds for each strategy.

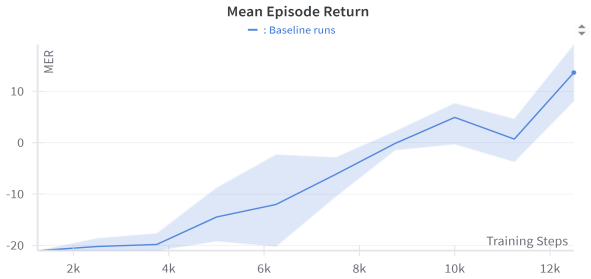
5 Results and Discussion

This chapter presents the empirical results from the experiments detailed in Section 3 and Section 4. The objective is to evaluate how different exploration strategies impact the performance, sample efficiency, and learning stability of the MuZero agent. We analyse the performance of each strategy across the three distinct Atari environments.

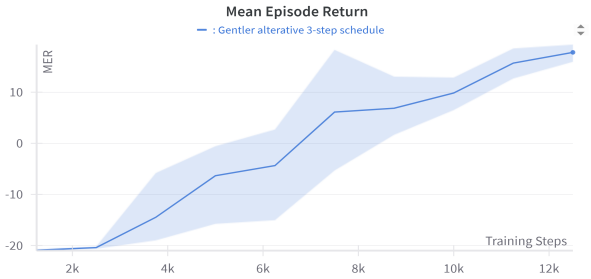
The results are grouped by environment. For each game, we present the primary learning curves, which plot the mean episodic return against the number of training steps. These plots serve as the main criteria for comparing the overall performance and sample efficiency of the baseline MuZero, epsilon-greedy variants, different temperature annealing schedules, and Thompson Sampling. The shaded region around each curve represents the standard mean error across three independent runs with different random seeds, providing insight into the stability of each method. We then discuss the key observations from these plots. Interestingly, performance for some methods varied significantly

between runs, simply because of the different initial seed provided.

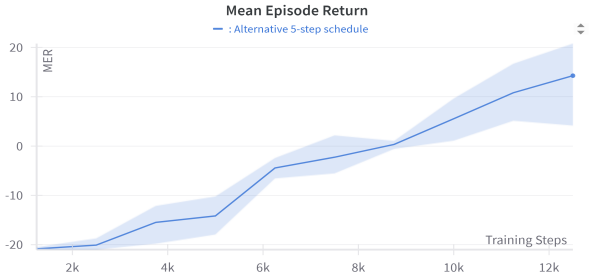
For Pong, most of the results followed a similar pattern and all tested strategies were sufficient to achieve similar performance to the baseline, indicating that the choice of exploration method is not a critical factor. However, still notably, for the run-time we set, the epsilon greedy methods struggled to reach the same return as the other methods. Out of all methods, only the epsilon greedy experiments and the linear temperature decay were unable to reach the baseline value of 11 at the end of their runs. Even though the agent was still able to perform. Surprisingly, the gentler alternative 3-step schedule, which includes a high initial temperature and a drop halfway through, managed to match the performance of the cosine decay method. The epsilon-greedy experiments also produced results with high mean error. This was not only the case for Pong, but also for the other environments.



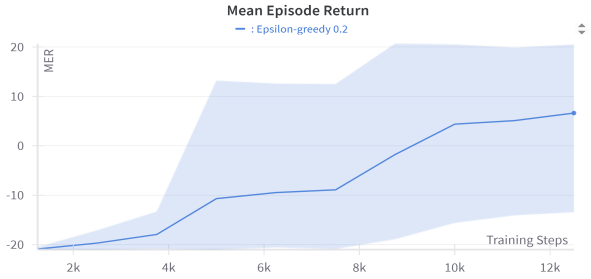
(a) Baseline



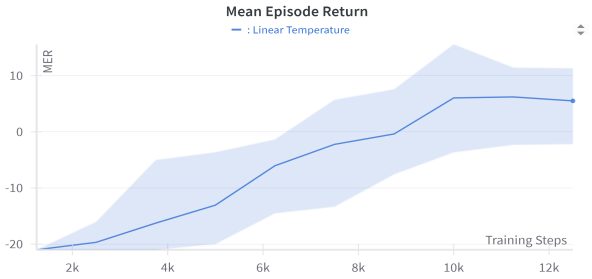
(b) Gentler Alternative Step Schedule



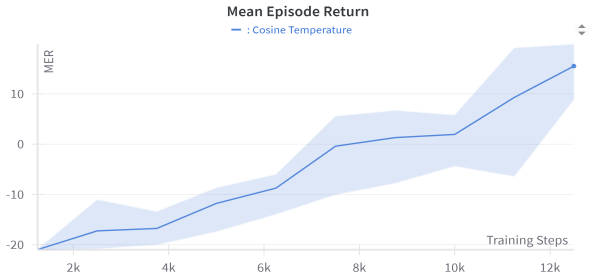
(c) Alternative 5 Step Schedule



(d) Epsilon Greedy 0.2

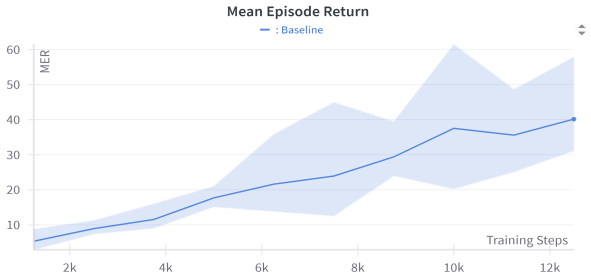


(e) Linear Temperature Decay

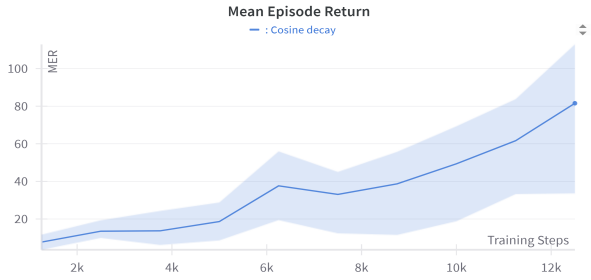


(f) Cosine Temperature Decay

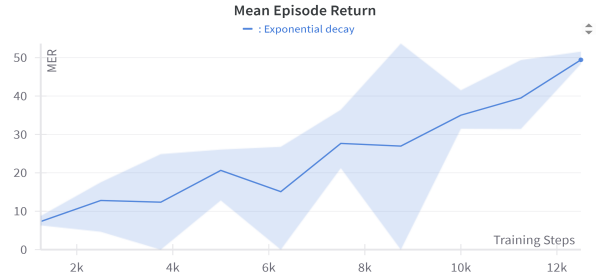
In Breakout, which requires a specific, multi-step strategy, we began to see a divergence. Strategies that encourage more structured or sustained exploration, such as continuous annealing, showed an advantage over purely random methods like epsilon-greedy. The choice of strategy started to matter.



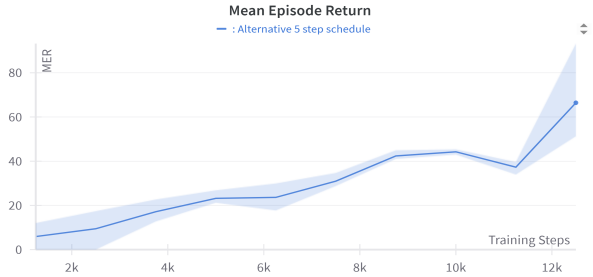
(a) Baseline



(b) Cosine Temperature Decay

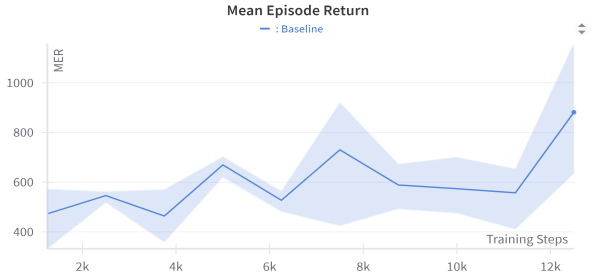


(c) Exponent Temperature Decay

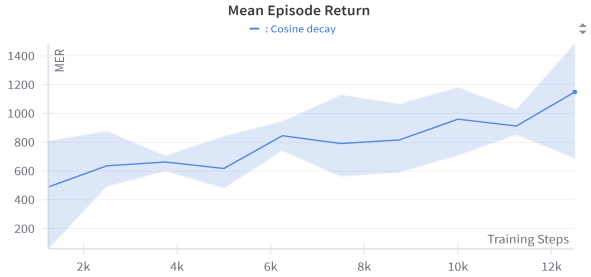


(d) Alternative 5 Step Schedule

This trend was clear in Ms. Pac-Man, where the significant exploration challenge magnified the differences between methods. Here, sophisticated, temporally consistent exploration, as embodied by the annealing functions, proved to be the most effective approach for navigating the complex state space and sparse rewards to achieve high performance.



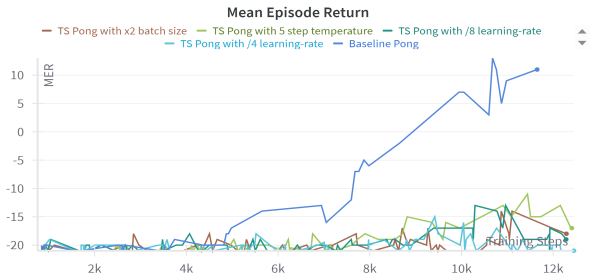
(a) Baseline



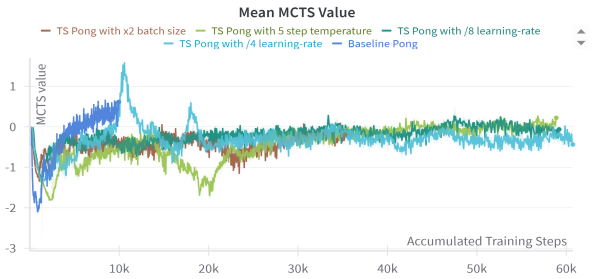
(b) Cosine Temperature Decay

Unfortunately, due to various limitations, the Thompson Sampling method was only run in the Pong environment. This method took significantly longer to train than the others, and although supposedly a more robust method of action selection, it failed to match the other methods in performance, even when tuned. The low performance of the implementation suggests that it was incorrectly applied to MuZero, on the other hand, several other training metrics indicate that the TS experiments were closely following the other methods. As we can see in the figures below, the mean episode of the Thompson Sampling experiments stays flat throughout, but strangely, the mean MCTS value matches that of the baseline run. The same can be said for the target values, predicted values, target rewards and predicted rewards. If the implementation is not the cause of this, potential reasons for these unexpected results could be that when the agent applies the use of TS, it requires longer training time to reach equivalent levels of performance. In

our experiments, we attempted to address the poor performance by running TS with different initial hyperparameters, such as reducing the learning rate and increasing the training batch size. However, these did not change the end result meaningfully.



(a) Thompson Sampling Mean Episode Return



(b) Thompson Sampling Mean MCTS Value

6 Conclusions and Future Work

This thesis investigated how different action selection strategies impact the performance of the MuZero algorithm. Our findings show that no single strategy is universally optimal; instead, effectiveness is highly dependent on the environment's complexity. In simple games like Pong, the choice of exploration method had little effect on the final performance. However, as the task complexity increased in games like Breakout and Ms. Pac-Man, more structured exploration methods, particularly continuous temperature annealing, demonstrated a clear advantage over simpler approaches.

The most perplexing result was the failure of our Thompson Sampling implementation to achieve competitive performance, despite its theoretical advantages. In essence, this research confirms that action selection is a crucial component of MuZero and that performance on challenging tasks can be significantly improved by tailoring the exploration strategy to the problem at hand.

Based on our findings, several avenues for future research are apparent. A primary focus should be a thorough investigation and debugging of the Thompson Sampling implementation to understand its unexpected performance. Additionally, testing these strategies on a broader suite of hard-exploration games, such as Montezuma's Revenge, would provide a more strenuous and revealing benchmark.

Further work could also explore adaptive exploration mechanisms that adjust parameters dynamically based on agent uncertainty or performance. Finally, a more extensive project with greater computational resources could undertake a full hyperparameter optimisation for each strategy on each environment to determine the true performance ceiling of these methods.

7 Responsible Research

In this section, we highlight our approach to ensure responsible research practices. This project does not involve any use of personal data or potentially sensitive information and has no direct societal interaction, therefore posing minimal ethical risk. Nonetheless, it adheres to general ethical guidelines for artificial intelligence research. The goal of this study is to contribute to the broader scientific understanding of reinforcement learning, specifically by analysing how different exploration strategies affect the performance of MuZero in various game environments.

To promote reproducibility and transparency, the full codebase repository can be found in the Appendix. The implementation used follows the standard MuZero architecture and exploration methods, using well-established Python libraries, predominantly Jax.

As previously stated in Section 4.3, experiments were conducted using the computational resources of the Delft-Blue supercomputer, provided by Delft High Performance Computing Centre [4], and resources from Vast.ai [18]. All training runs and evaluations were tracked and logged using Weights & Biases (wandb.ai)[2], ensuring comprehensive logging of metrics and training behaviour.

To reinforce the reliability of the findings, all experiments were repeated using multiple seeds. All reported metrics include standard error where applicable to account for variance across runs. While random initialisation and stochasticity in training may still lead to some run-to-run variation, care has been taken to minimise these effects through consistent methodology and robust evaluation practices. Details of the environments and exploration strategy parameters used are specified in Section 4.1 and Section 4.2, respectively.

Large language models (LLMs) were used to assist in the writing process of this investigation. Specifically, they supported the rewording of sentences for improved clarity and consistency as well as light proofreading to enhance readability and structure.

By making all experimental details available and by using principled evaluation strategies, this research aims to contribute to a responsible and reproducible reinforcement learning literature.

References

[1] Aijun Bai, Feng Wu, and Xiaoping Chen. Posterior sampling for monte carlo planning under uncertainty. *Applied Intelligence*, 48(12):4998â5018, 2018.

[2] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[3] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

[4] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2`, 2024.

[5] Jinke He, Thomas M. Moerland, Joery A. de Vries, and Frans A. Oliehoek. What model does muzero learn?, 2024.

[6] Xiuyuan Lu and Benjamin Van Roy. Ensemble sampling. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[7] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling, 2013.

[8] Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Approximate thompson sampling via epistemic neural networks, 2023.

[9] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, and Ian Osband. A tutorial on thompson sampling. *CoRR*, abs/1707.02038, 2017.

[10] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604â609, December 2020.

[11] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model, 2021.

[12] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484â489, 2016.

[13] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354â359, 2017.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[16] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[17] Edan Toledo, Laurence Midgley, Donal Byrne, Callum Rhys Tilbury, Matthew Macfarlane, Cyprien Courtot, and Alexandre Laterre. Flashbax: Streamlining experience replay buffers for reinforcement learning with jax, 2023.

[18] Vast.ai. Vast.ai - gpu rental marketplace. `https://www.vast.ai`, 2023. Accessed: 2025-06-20.

A Experimental Setup

A.1 Codebase

The following GitLab repository holds the baseline MuZero implementation as well as the various modifications explored in this paper: `https://gitlab.tudelft.nl/jinkehe/bachelor-research-project`

A.2 Default Experiment Configuration

```
# ==============================================================================
# training
# ==============================================================================
training:
  seed: 0
  env_steps: 100_000
  offline_update_steps: 0
  learn_per_update_step: 1
  eval_per_update_step: 1250
  log_per_update_step: 125
  save_model_per_update_step: 12500

# ==============================================================================
# logging
# ==============================================================================
logging:
  use_wandb: True
  use_tensorboard: False
  wandb:
    project: muzerorp # your project name
    entity: amiguitozinho-tu-delft # your wandb username
    name: muzero_run # run name
  profiling: True

# ==============================================================================
# environment
# ==============================================================================
env:
  common:
    game_name: 'Pong'
    env_kwargs: {}
    n_skip: 4
    n_stack: 4
    screen_size: 96
  train:
    num_envs: 8
    vectorization_mode: async
    max_episode_steps: 3000
    noop_max: 30
    clip_reward: True
    terminal_on_life_loss: True
    record_episode_statistics: True
  eval:
    num_envs: 50 # equivalent to number of episodes for evaluation
    vectorization_mode: async
    max_episode_steps: 6000 # 27000
    noop_max: 30
```

```
    clip_reward: False
    terminal_on_life_loss: False
    record_episode_statistics: True

# ============================================================================
# agent
# ============================================================================
agent:
  type: MuZero
  value_transformation:
    num_bins: 601
    support: [-300, 300]
    use_transformation: True
  discount_factor: 0.997
  neural_network:
    norm: LN
    prediction_net_norm: LNv2
    use_v2: False
    activation: leaky_relu
    representation_net_num_channels: 64
    prediction_net_num_channels: 128
    prediction_net_mlp_num_features: 64
    normalize_state: True
    prediction_net_mlp_zero_initialize_last_layer: True
    initializer: variance_scaling
  act:
    train:
      num_simulations: 50
      pb_c_init: 1.25
      pb_c_base: 19652
      dirichlet_alpha: 0.3
      dirichlet_fraction: 0.25
      max_depth: null
      qtransform:
        by: updatable_min_max
        epsilon: 0.01
        completed_by: node
      temperature:
        - [0.5, 1.0]
        - [0.75, 0.5]
        - [1.0, 0.25]
    eval:
      num_simulations: ${agent.act.train.num_simulations}
      pb_c_init: ${agent.act.train.pb_c_init}
      pb_c_base: ${agent.act.train.pb_c_base}
      dirichlet_alpha: ${agent.act.train.dirichlet_alpha}
      dirichlet_fraction: 0.0
      max_depth: ${agent.act.train.max_depth}
```

```
    qtransform:
      by: ${agent.act.train.qtransform.by}
      epsilon: ${agent.act.train.qtransform.epsilon}
      completed_by: ${agent.act.train.qtransform.completed_by}
    temperature:
      - [1.0, 0.0]
  reanalyze:
    num_simulations: ${agent.act.train.num_simulations}
    pb_c_init: ${agent.act.train.pb_c_init}
    pb_c_base: ${agent.act.train.pb_c_base}
    dirichlet_alpha: ${agent.act.train.dirichlet_alpha}
    dirichlet_fraction: ${agent.act.train.dirichlet_fraction}
    max_depth: ${agent.act.train.max_depth}
    qtransform:
      by: ${agent.act.train.qtransform.by}
      epsilon: ${agent.act.train.qtransform.epsilon}
      completed_by: ${agent.act.train.qtransform.completed_by}
learn:
  reanalyze:
    per_learn_step: 25
    batch_size: null
  replay_buffer:
    min_size: 2_000
    max_size: 100_000
    PER:
      alpha: 0.6
      beta_start: 0.4
      beta_end: 1.0
      priority: predicted_to_target
      correction_per_learn_step: 25
    log:
      per_learn_step: 10
      sample_size: 100
    save: False
  optimizer:
    type: adamw
    batch_size: 256
    lr: 0.0008
    weight_decay: 1e-4
    max_grad_norm: 5.0
    steps: 8
    warmup_ratio: 0.0
  loss:
    model:
      unroll_steps: 5
    reward:
      coef: 1.0
    policy:
```

```
  coef: 1.0
  entropy_coef: 0.0
value:
  coef: 0.25
  td_steps: 5
```