Extended Abstract

Motivation Video games provide a useful tool for developing Reinforcement Learning (RL) systems with a variety of complexity levels, from basic policy methods to advanced planning and meta-learning systems, in a safe and controlled environment. The visual aspect of video games is especially important for developing computer vision systems, as in robotics. In this project, we use this visual task of playing Super Mario Bros to push the boundaries of two fundamental RL algorithms and better understand those systems from the ground up.

Methods For DQN, we did custom hyperparameter tuning and experimented with the following adjustments to the algorithm: sampling methods (to select actions during training and testing; either Epsilon-greedy exploration or Boltzmann exploration), warm-starting with behavior cloning from human gameplay data (exploitation-focused playing, exploration-focused playing, or none), and replay buffer pruning (using k-means clustering).

For PPO, in addition to hyperparameter tuning, we experimented with the sampling method and the network architecture. For the sampling methods, we used Boltzmann sampling as well as a pseudo-greedy sampling method, where the model is limited to selecting from the top-k highest-predicted actions. (For $k = 1$, this is equivalent to true greedy sampling.) For the network architecture, we began with the same underlying architecture as in the DQN experiments, and we expanded this network to be both deeper (containing more layers) and wider (containing larger hidden dimensions).

Results For DQN, our best-performing system used Boltzmann exploration and was warm-started from the expert exploration data, with no replay buffer pruning. We found that training with epsilon-greedy exploration produced almost no constructive learning in the model. Similarly, replay buffer pruning produced much worse performance during training and testing. Regarding generalization, we found that the DQN model that beat level 1-1 could not successfully complete level 1-2, even after extensive additional training.

For PPO, we conducted a targeted hyperparameter search to optimize agent performance. Further improvements were achieved using the k-greedy action selection adaptation, which limited exploration during inference to the top $k = 3$ scoring actions. With this setup, the PPO model successfully completed Level 1-1 96.32% of the time, the highest success rate among all tested configurations.

Discussion For DQN, these results highlight the critical role of early exposure to diverse experiences and effective exploration strategies in reinforcement learning. Warm-starting with exploration-heavy expert data provided a broader foundation for learning, enabling the agent to discover robust behaviors early on. Boltzmann sampling encouraged adaptive and value-informed exploration, leading to smoother training, but struggled in sparse-reward scenarios—most notably, near the flag where agents exhibited aimless behavior due to missing terminal rewards. Additionally, the removal of outlier experiences via replay buffer pruning limited the model's ability to learn or perform in novel situations.

In contrast, PPO's deterministic policies yielded more stable performance but limited adaptability; the agent failed to recover from mistakes or generalize to new levels. This was evident in qualitative behaviors such as getting stuck near unfamiliar enemies or failing to adjust to novel layouts. DQN, while more chaotic in training, showed greater flexibility in recovery, aided by occasional stochastic actions. Ultimately, PPO favored consistency over adaptability, while DQN relied heavily on exploration mechanisms and environmental feedback—both highlighting the trade-offs between stability, flexibility, and generalization in RL.

Conclusion In this work, we used reinforcement learning to train neural network systems to play Super Mario Bros. from visual input data. In this context, we explored two reinforcement learning algorithms: DQN and PPO. We observed the best overall performance from PPO, but we generally observed a larger variation from algorithmic and architectural variations within each algorithm than between the two groups. This project highlights the superlative importance of hyperparameter tuning and other design decisions in the evaluation of reinforcement learning methods.

MARIO: Reinforcement Learning on Image Observations

Nika Zahedi
Department of Electrical Engineering
Stanford University
nzahedi@stanford.edu

Nils Kuhn
Department of Electrical Engineering
Stanford University
nfkuhn@stanford.edu

Evelyn Yee
Department of Computer Science
Stanford University
yeevelyn@stanford.edu

Abstract

Video games provide a useful testbed for developing Reinforcement Learning (RL) systems with visual components, due to the interactive nature and level of control researchers can exercise over the environment. In this work, we used Super Mario Bros. to investigate two fundamental RL algorithms: DQN and PPO, performing hyperparameter search and making small algorithmic modifications to understand what components are most essential for performance. We observed the best overall performance from a PPO model, but we generally observed a larger variation from algorithmic and architectural variations within each algorithm than between the two groups. This result highlights the superlative importance of design details in the evaluation of reinforcement learning methods, even more than the general theory of the system. We found that, with RL, the devil truly is in the details.

1 Introduction

This project focuses on Super Mario Bros, a classic video game and popular RL development environment. While previous methods have successfully trained agents to complete the first level, they often reduce the challenge of the task by using specifically defined information about the state, so that the agent doesn't have to extract the information by the image of the current state. Additionally, prior work often reduced the hardness of the game by reducing the number of actions that the agent can choose from.

The objective of this project is to explore and compare different reinforcement learning techniques in playing the game with the full control and images to mimic the experience that a human has. This should also be a baseline for within the bigger challenge of playing from the visual improve generalization across multiple Mario levels from visual-only input, and warm-starting with imitation learning from expert play. By evaluating and contrasting these methods, we aim to identify effective strategies for enabling agents to infer world states, reuse learned behaviors and adapt efficiently, contributing to more robust RL systems suitable for complex, real-world applications.

2 Related Work

Video games are a popular application for new RL algorithms because they offer a safe and controlled environment for training and evaluating. They allow for easy data collection through gameplay and

eliminate real-world risks from poor policy decisions. Additionally, they provide a variety of levels of input complexity, as agents can be trained from the true game state (e.g. the character positions) or from observations, like image and sound. The visual aspect of video games is an especially valuable aspect for training and testing visual RL models, which are widely used in robotics.

One notable video game platform for RL is the Arcade Learning Environment (ALE), created by Marc G. Bellemare (2013). This platform emulates the Atari 2600 and includes implementation for a few classic Atari games, like Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invader. Since ALE's introduction, Atari games have been a staple of reinforcement learning algorithm development, including the original Deep Q-Network (DQN) paper (Mnih et al., 2013) and more recent works on model-based reinforcement learning (Werner Duvaud, 2019; Finn et al., 2017; Hafner et al., 2024; Voelcker et al., 2025) and agentic LLM systems (Xu et al., 2025; Delfosse et al., 2025; Hao et al., 2025)

The Super Mario Bros. environment used in this project (Kauten, 2018) is built on the OpenAI Gym framework (Brockman et al., 2016), which provides a standardized and widely-used platform for building environment models for reinforcement learning (RL) research. Several RL architectures have been applied to the Super Mario Bros. gym environment. Notably, Guzman (2023) and Nguyen (2021) use Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Kundu (2023) and Jung (2016) use DQN to train agents to successfully complete the first level. However, these approaches often simplified the action space by removing sprinting or only enabling simple jumps and forward movement to make the learning problem more tractable.

3 Environment Details

Figure 1: An example frame from the sequence of four state images used as input to the neural network.

We model the problem of learning to play Super Mario Bros as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, with the following definitions:

- **Action Space $\mathcal{A}$:** To give the agent full control over Mario and to closely mimic the experience of a human player, we adopt the "COMPLEX MOVEMENT" action space, which consists of 12 discrete actions (4 movement directions, jumping, running, standing still, and combinations of direction/jump/run). To reduce computational overhead and promote temporal consistency, the agent selects a new action every four frames, in line with the methodology of Mnih et al. (2013). The chosen action is then repeated for the subsequent four frames.
- **State Space $\mathcal{S}$:** In our setup, the agent's observation is a concateation of the pixel data from the four most recent game frames, enabling it to infer temporal information such as its own velocity and the movement direction of nearby enemies. The OpenAI Gym environment outputs RGB images with a resolution of 240×256 pixels, which we convert grayscale and downsample to 84×84 pixels. The resulting four frames are then stacked along the channel dimension to form a state tensor of shape 4×84×84. An example input frame is shown in Figure 1.
- **Rewards $\mathcal{R}$:** We adopt the reward structure defined by the OpenAI Gym environment, where the reward at each time step t is given by:

  $$\text{Reward}_t = v_t + c_t + d_t$$

  where v_t encourages horizontal velocity, $(c_0 - c_1)$ is the difference in clock value before c_0 and after c_1 the action, rewarding faster progression through the level, and $d_t = -15$ if the

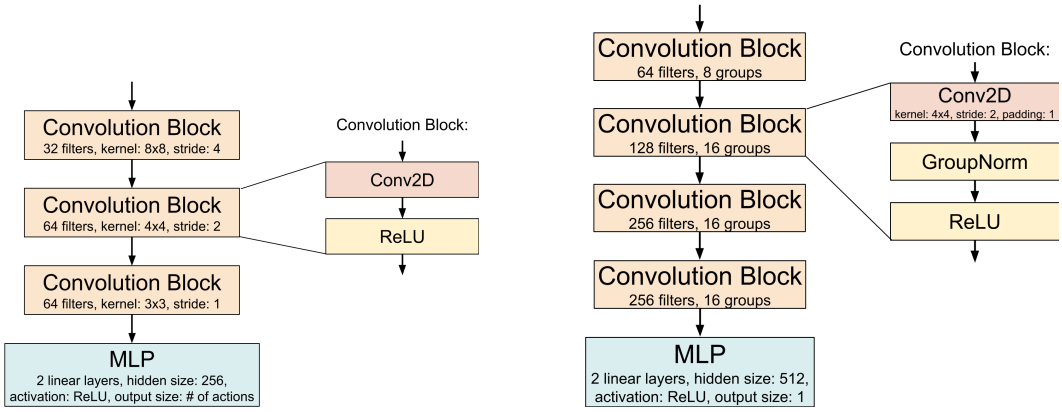
agent dies, otherwise 0. Notably, there is no explicit reward for completing a level or for game-score-related actions such as defeating enemies or collecting coins. Completion is incentivized implicitly by avoiding penalties due to time.

- **Dynamics $\mathcal{P}$:** The environment dynamics are governed by the internal logic of the Super Mario Bros simulation and the actions chosen by the agent. At each time step, the environment transitions deterministically from one state to the next based on the current state and the selected action. These transitions include updates to Mario's position, interactions with terrain elements (e.g., platforms, pipes, and gaps), and responses to collisions with enemies or items. There is minor stochastic variation between each run in the number of enemies, particularly Goombas, in specific regions of the level.

We primarily trained and evaluated with the first level of Super Mario Bros., Level 1-1, but we also performed some generalization experiments on Level 1-2, which had an underground environment and produced very different visual observation distributions.

4 Models

Within our experiments, we used two primary algorithms to train agents in this Super Mario environment: Deep Q-learning (DQN) and Proximal Policy Optimization (PPO).



(a) DQN and PPO policy network.

(b) Modified PPO value network.

Figure 2: The underlying neural network architectures used for our experiments. The convolutional structure for 2a was largely inspired by Mnih et al. (2013).

4.1 Deep Q-learning

The Deep Q-learning (DQN) algorithm we used for this project was originally proposed by Mnih et al. (2013). Q-learning generally trains a neural network to learn a Q-function that estimates the expected trajectory reward from each possible action from a given state. Furthermore, DQN introduces a replay buffer, from which minibatches of training experiences (represented by a 5-tuple of (state, action, reward, next state, next action)) are sampled. Then, after doing a step of Q-learning, the model is rolled-out and the resulting experiences are saved back to the replay buffer for future training. In this algorithm, we experiment with a few design details of this algorithm, particularly: the sampling method (for selecting actions during training and roll-out) and the initialization of the Q-network.

For this project, we used a convolutional neural network architecture to back our Q-network, similar to the architecture designed by Mnih et al. (2013), who also tackled game playing with pixel inputs. Our network architecture can be seen in Figure 2a.

4.1.1 Sampling methods

We experimented with several sampling strategies to improve the agent's ability to explore the environment effectively during training:

1. **Naive Epsilon-Greedy Sampling**
   Our initial approach was the standard epsilon-greedy strategy. At each decision point, the agent chose a random action with probability ε, and otherwise selected the action with the highest predicted Q-value. This approach encourages exploration, particularly in the early stages of training. The value of ε was gradually decayed over time, under the assumption that as the Q-function improves, it becomes more advantageous to exploit the learned policy.
2. **Confidence-Based (Modified) Epsilon-Greedy Sampling**
   To refine the basic epsilon-greedy method, we developed a confidence-aware variant. In this approach, a random action was selected with probability ε *only if* the top two predicted Q-values were close in magnitude. This indicates that the model lacks confidence in its preferred action, and thus exploration is more likely to be beneficial. This strategy allows for more targeted exploration and avoids unnecessary randomness when the model is confident in its prediction.
3. **Boltzmann Sampling**
   Our final sampling method used Boltzmann (or softmax) exploration. At each timestep, the agent selected an action probabilistically, using a softmax over the predicted Q-values:

   $$P(a) = \frac{e^{Q(s,a)}}{\sum_{a'} e^{Q(s,a')}}$$

   Boltzmann sampling maintains a balance between exploration and exploitation by assigning higher probabilities to actions with higher Q-values, while still allowing for stochastic action selection. This makes it well-suited for situations where a purely greedy policy may converge prematurely.

4.1.2 Warm-starting with Behavior Cloning

Another adaptation we made to improve initial training stability and learning efficiency was to warm-start the Q-network using behavior cloning as an agent with expert demonstrations, rather than randomly initializing. For the expert data, we had one of our group members play Level 1-1 of the game in our environment for 12 successful runs, split into 2 data conditions. In the "exploitation" condition, the human expert played with only the goal of successfully completing the level as soon as possible, without dying. In the "exploration" condition, the expert played to explore more of the game states, interacting with enemies, gathering coins, and occasionally getting stuck, while also completing the level successfully.

4.2 Proximal Policy Optimization

In addition to DQN, we also investigated the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017). PPO is a policy-based, actor-critic algorithm that directly learns a stochastic policy $\pi(a|s)$ by maximizing the expected return through gradient ascent. It uses two neural networks: a policy network (actor) that outputs action probabilities, and a value network (critic) that estimates the state-value function $V^{\pi}(s)$. The critic guides the policy updates by estimating the advantage function $A^{\pi}(s, a)$, which reflects the relative value of an action compared to the average performance at that state:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

The actor is trained to maximize the expected advantage, improving actions that yield higher returns.

To ensure stable and conservative updates, PPO optimizes a clipped surrogate objective rather than the raw policy gradient. To enable this, $r_t(\theta)$ is defined as the probability ratio between the updated and old policies for each of the states.

$$r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

The clipped objective prevents the ratio from moving too far from 1 within each policy update by optimizing the following surrogate:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t,\ \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) A_t \right) \right]$$

For our experiments, we used $\varepsilon = 0.1$, which is a typical value for ppo.

In addition to clipping, an early stopping criterion based on the Kullback–Leibler (KL) divergence between the new and old policies was implemented. During training, if the empirical KL divergence exceeds a predefined threshold, the policy update is terminated early:

$$\mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)\right] > \delta$$

For the final trainings of ppo, we used a value of $\delta = 0.15$.

4.2.1 General advantage estimates

To improve the performance and stability of the PPO agent, we implement Generalized Advantage Estimation (GAE) Schulman et al. (2016). GAE allows for a tunable trade-off between bias and variance in the estimation of advantages by applying an exponentially weighted sum of temporal differences, thereby smoothing the advantage signal over time.

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

In our experiments, we choose a value of $\lambda = 0.95$ as well as a value of $\gamma = 0.995$.

4.2.2 K-greedy adaptation of PPO

To improve performance, we introduce a k-greedy variant of PPO for testing. With this method, the agent selects actions only from the top k logits (highest-scoring actions), applying the softmax over this restricted set. This reduces the chance of selecting poorly ranked actions that still retain non-zero probability in standard PPO. While beneficial during testing, this method wasn't used for training, as it limits exploration and may prevent the agent from learning to use less likely but potentially valuable actions.

5 Results and Discussion

5.1 Deep Q-learning (DQN)

Warm Start	Sampling Method	Success Rate (%)	Average Score	Average Coins
None	Boltzmann	7.00	449.00	1.03
Exploitation Expert	Boltzmann	33.00	575.00	1.21
Exploration Expert	Boltzmann	**60.00**	774.42	1.59
Exploration Expert	Greedy ($\varepsilon = 0.005$)	29.00	**936.00**	**1.78**

Table 1: Test performance of trained DQN models on Level 1-1. Success rate indicates the proportion of test trajectories where the model completed the level in the time limit, without dying. All DQN models were trained using Boltzmann sampling, and at test time, we compare the Boltzmann sampling strategy with the epsilon greedy strategy for the DQN models. Each final model was tested for 100 trajectories.

We investigated three main components influencing DQN performance: the choice of sampling strategy, warm-start initialization using behavior cloning, and replay buffer management. The best performing models combined warm-starting from exploration-heavy expert data with Boltzmann sampling, yielding higher rewards during training (as seen in Figure 3), and higher level completion rates and better average scores at testing (as seen in Table 1). Models trained from scratch struggled to learn unless exploration was well-managed, and showed a huge reliance on the hyperparameter ε. Architectural choices in buffer management (such as clustering and pruning) negatively impacted learning by removing valuable outlier experiences.

Sampling Results. Different sampling strategies had a noticeable impact on exploration and policy quality. Boltzmann sampling improved training stability and led to smoother learning curves. In early training phases, we found that Boltzmann sampling allowed the model to meaningfully explore and build a better representation of the true environment Q-function. In contrast, ε-greedy exploration forced occasional random actions, which helped the system escape local optima but also introduced

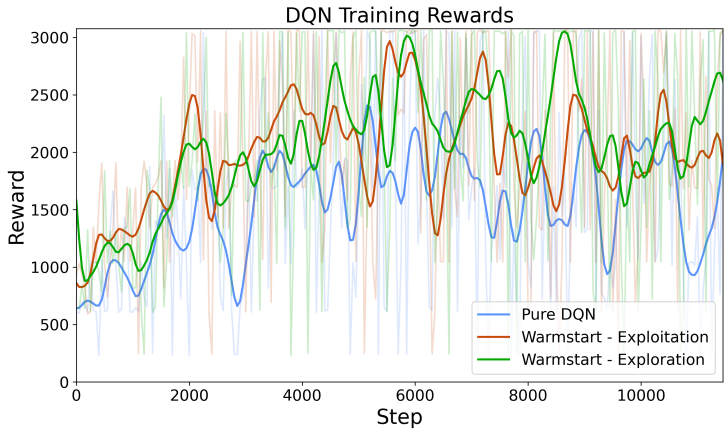


Figure 3: Environment rewards over training steps for the training of DQN models on Level 1-1. For warm-started models, the first 100 steps are BC training on the expert data. The bolded curves were smoothed using a moving average over a window of 50 steps, to indicate general trends.

too much instability, making it unusable for training. However, we found that ε-greedy sampling at test time (after training with Boltzmann sampling) achieved the highest average test score on Level 1-1, despite having poor performance at actually completing the level.

Warmstart Results. Pre-training the agent using behavior cloning on expert demonstrations significantly accelerated learning. Among the warm-starting conditions, the exploration-focused expert data outperformed the exploitation-only data across all metrics. The exploration expert exposed the agent to a wider range of states and behaviors, enabling more robust policy learning. In contrast, the exploitation expert limited the diversity of training data, which resulted in lower success rates and less exploratory behavior. However, it should be noted that the quality of expert data mostly influenced the early stages of training, and it was later diluted by the diversity of states encountered during gameplay.

Qualitative Observations. Under ε-greedy exploration, the agent initially relied on random actions, often resulting in inefficient and erratic behavior before finding a working strategy. However, as ε decreased, the agent reached later stages of the level more reliably, but its ability to adapt to new obstacles diminished. Most commonly, this model got stuck when the greedy action failed to change the state, such as running into a wall, until a randomly sampled action bounced it out. Despite this limitation, we did find that ε-greedy exploration enhanced the agent's ability to recover in unexpected ways from suboptimal actions, learning behaviors such as jumping away from an enemy or redirecting mid-air to land on one. This appears to be a beneficial side effect of the stochastic exploration introduced by random actions.

Boltzmann sampling, by contrast, tended to get stuck in low-risk areas. For example, near the final flag at the end of the level, the agent often jumped around aimlessly. We attributed this to a lack of reward signal; reaching the flag itself provided no explicit reward, and the low velocity penalty did not accumulate over as many actions, as the trajectory stopped when the flag is reached. To test this hypothesis, we trained another agent with Boltzmann sampling and added a reward of 100 for reaching the flag. Using identical hyperparameters and episode count, the modified agent mostly overcame this failure mode. This behavior reveals a limitation of Boltzmann sampling in DQN: when Q-values are similar for many actions, the agent is more likely to fall into suboptimal actions. While reducing temperature can sharpen the action distribution, it may hinder learning, as seen with the ε-greedy models.

5.1.1 Generalization of DQN

We were interested in whether a DQN agent trained exclusively on Level 1 could generalize to Level 2 without additional fine-tuning. As shown in Table 2, models trained solely on Level 1 consistently failed to make meaningful progress on Level 2, typically dying to the first new enemy encountered.

Retraining	Sampling Method	Success Rate (%)	Average Score	Average Coins
No	Boltzmann	0	230.00	0.62
Yes	Boltzmann	0	**819.00**	1.69
Yes	$\varepsilon = 0.01$	0	755.00	**2.03**

Table 2: Generalization test performance. All results from DQN models, trained on Level 1-1 with Boltzmann Sampling. The retrained models took the Level 1-1 checkpoint and trained for an additional 11,000 training episodes. Each final model was tested for 100 trajectories. See Appendix A.1.1 for more results from the DQN generalization experiments.

This suggests limited transferability of the learned policy in the face of even moderate environmental changes.

To investigate further, we resumed training from the final Level 1 checkpoints and continued training the agents on Level 2 for an additional 10,000 steps (see Figure 7). These retrained models showed improved performance over the original baseline, managing to survive longer and progress further through the level. However, even with continued training, none of the models were able to fully complete Level 2, indicating that generalization remained limited and task-specific adaptation was still necessary.

DQN Generalization Qualitative Results. Visually, Level 2 differs significantly from Level 1, featuring a darker color palette and introducing new enemy types. Although the visual domain shift was partially mitigated using a grayscale (black and white) filter, the agent consistently failed to adapt to the behavioral differences posed by the new enemies. In particular, it struggled to reliably bypass or defeat one of the newly introduced enemy types within the training duration. This suggests that the agent's policy may have overfit to familiar patterns from Level 1. To address this limitation, further training with additional expert demonstrations on the new level could provide more targeted guidance. Alternatively, using a larger or more expressive neural network architecture may help capture the broader distribution of state-action pairs necessary for generalization across levels.

5.2 Proximal Policy Optimization (PPO)

To enhance performance and stability of our PPO agent, we conducted a series of architectural and hyperparameter experiments. Figure 4 shows the training behavior of these architectural variations. Initially, we began by using the same DQN convolutional network architecture (Figure 2a) for both the policy and value networks in PPO. For this model, the value loss increased exponentially, and we halted the training run prematurely due to the instability. To improve stability, we introduced a larger and deeper convolutional architecture, as described in Section 4.2. Additionally, we included some input pre-processing, normalizing the input pixel values to a range between $[0, 1]$, and we also increased the training batch size from 64 to 256.

As shown in Figure 4, using the larger network for both the value and policy networks produced a more stable value loss. However, we observed a strong correlation between rising value loss and reward improvements, suggesting that value estimation errors could destabilize the policy in long training runs. To mitigate this, we tested a hybrid setup, using the smaller, DQN-style network for the policy and the larger architecture for the value. This configuration led to more stable value loss, which was decorrelated from the performance of the policy net. This improved stability indicates that when the policy is less complex, the expected outcome is easier to model, leading to improved value estimates.

The performance of the policies (as seen in the top graph of Figure 4) suggests that the value estimates had limited immediate effect on the policy. Long-term training revealed that stable value estimation was critical for the policy to converge. Most configurations performed comparably within the first 1600 episodes (with reward accumulated over 1536 environment steps, which may span multiple episodes). 1536 steps were chosen to fix the amount of memory needed within the GPU. For comparison, a policy with chances of successfully ending the level often showed accumulated rewards of over 15000. We trained the final version of ppo on 13700 games with network updates every 8 games and 10 epochs of updates. The final success rate of our agent is shown in table 3 under

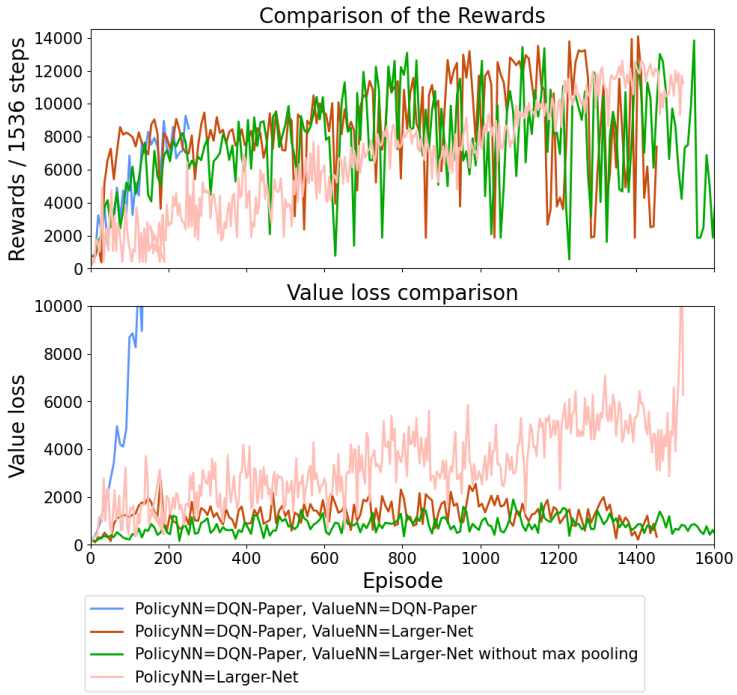


Figure 4: Learning curves from PPO training, over 1600 epochs. The top figure shows environment rewards earned at each step of training, and the bottom figure shows the loss of the Critic Value network at each step. "DQN-Paper" indicates the original convolutional network architecture as in DQN (Figure 2a), and "Larger-Net" indicates the expanded architecture (Figure 2b).

the stochastic sampling method. We also tested that ppo agent with the restricted Pseudo-Greedy method. The agent performed best with $k = 3$, consistently finishing the game most of the tests.

Sampling method	Success Rate (%)	Average Score	Average Coins
Stochastic	85.49	285.00	0
Pseudo-Greedy ($k = 3$)	**96.32**	278.00	0

Table 3: Test performance of the PPO models with the final architecture (large value network, smaller policy network). Each model was trained with the specified sampling method and tested for 100 trajectories.

5.2.1 Qualitative Analysis of Proximal Policy Optimization

Unlike DQN, the PPO agent begins policy updates only after collecting a batch of interactions, resulting in initially random behavior and slower learning in the early stage. Additionally, PPO exhibits a fundamentally different form of exploration due to its probabilistic action selection, sampled from a softmax distribution. Although this ensures non-zero probability for all actions in theory, the policy often becomes nearly deterministic in practice, strongly favoring certain actions. This deterministic behavior promotes stable and consistent trajectories once the policy has converged. However, it also limits the agent's capacity for learning recovery. Unlike DQN, which takes random actions with a certain probability and is then forced to learn how to recover, PPO tends to avoid such situations altogether.

The lack of recovery strategies in PPO was quantitatively reflected in the improved performance of a k-greedy variant of the trained PPO agent (with $k = 3$). Restricting the agent to sample only from its top 3 most probable actions reduced trajectory variance and helped avoid states where the original agent was more likely to get stuck or die.

5.2.2 Final Comparison: DQN vs. PPO

Motivated by the insights gained from hyperparameter tuning in PPO, we applied several analogous modifications to the DQN setup to evaluate whether it could also achieve a high success rate on the first Super Mario level. Specifically, we replaced the original DQN network with the value network architecture used in PPO, adjusting the output size to match the number of discrete actions. We also increased the learning rate to 1.5×10^{-4}, raised the batch size to 32, and normalized input pixel values to the $[0, 1]$ range, consistent with the PPO preprocessing pipeline. These adjustments led to a significant performance improvement, with the DQN agent achieving a success rate of up to 90.1%, as shown in Table 4.

RL Algorithm	Success Rate (%)	Average Score	Average Coins
DQN	91	194	0.02
PPO	**96.32**	278.00	0

Table 4: Test performance of final trained models on Level 1-1, after making modifications to the DQN system inspired by our PPO investigation. Each final model was tested for 100 trajectories.

6 Conclusion

In this work, we used reinforcement learning to train neural network systems to play Super Mario Bros. from visual input data. In this context, we studied two reinforcement learning algorithms: Deep Q-learning (DQN) and Proximal Policy Optimization (PPO), exploring the impact of hyperparameters and other design details for each algorithm. We observed the best overall performance from PPO, using a larger network for the critic and sampling actions stochastically from a smaller network for the actor. However, for both algorithms, we observed a larger variation due to design variations within each algorithm than between the two groups. This finding highlights the superlative importance of hyperparameter tuning and other design decisions in the evaluation and deployment of reinforcement learning methods.

7 Team Contributions

- **Nika:** DQN implementation, hyperparameter tuning, and experimentation.
- **Nils:** Gather human expert data for BC training. PPO implementation, hyperparameter tuning, and experimentation.
- **Evelyn:** PPO initial implementation, ~~MuZero implementation,~~ data visualization, poster drafting, report drafting + editing.

Changes from Proposal We removed the MuZero and MAML algorithms from our project and instead focused more closely on the effects of hyperparameters, architecture details, and other design decisions for only DQN and PPO.

References

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

Quentin Delfosse, Jannis Blüml, Fabian Tatai, Théo Vincent, Bjarne Gregori, Elisabeth Dillies, Jan Peters, Constantin Rothkopf, and Kristian Kersting. 2025. Deep Reinforcement Learning Agents are not even close to Human Intelligence. arXiv:2505.21731 [cs.LG] `https://arxiv.org/abs/2505.21731`

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. arXiv:1703.03400 `https://arxiv.org/abs/1703.03400`

James Guzman. 2023. Super-Mario-Reinforcement-Learning. `https://github.com/james94/Super-Mario-Reinforcement-Learning`

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2024. Mastering Diverse Domains through World Models. arXiv:2301.04104 [cs.AI] https://arxiv.org/abs/2301.04104

Qianyue Hao, Yiwen Song, Qingmin Liao, Jian Yuan, and Yong Li. 2025. LLM-Explorer: A Plug-in Reinforcement Learning Policy Exploration Enhancement Driven by Large Language Models. arXiv:2505.15293 [cs.LG] https://arxiv.org/abs/2505.15293

Alexander Jung. 2016. Playing Mario with Deep Reinforcement Learning. https://github.com/aleju/mario-ai

Christian Kauten. 2018. Super Mario Bros for OpenAI Gym. GitHub. https://github.com/Kautenja/gym-super-mario-bros

Sourish Kundu. 2023. Using Reinforcement Learning algorithms to teach the computer to beat Super Mario Bros. https://github.com/Sourish07/Super-Mario-Bros-RL.git

Joel Veness Michael Bowling Marc G. Bellemare, Yavar Naddaf. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47 (2013).

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602* (2013). https://arxiv.org/abs/1312.5602

Viet Nguyen. 2021. Proximal Policy Optimization (PPO) algorithm for Super Mario Bros. https://github.com/vietnh1009/Super-mario-bros-PPO-pytorch

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1506.02438

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

Claas Voelcker, Anastasiia Pedan, Arash Ahmadian, Romina Abachi, Igor Gilitschenski, and Amir massoud Farahmand. 2025. Calibrated Value-Aware Model Learning with Stochastic Environment Models. arXiv:2505.22772 [cs.LG] https://arxiv.org/abs/2505.22772

Aurèle Hainaut Werner Duvaud. 2019. MuZero General: Open Reimplementation of MuZero. https://github.com/werner-duvaud/muzero-general.

Zelai Xu, Zhexuan Xu, Xiangmin Yi, Huining Yuan, Xinlei Chen, Yi Wu, Chao Yu, and Yu Wang. 2025. VS-Bench: Evaluating VLMs for Strategic Reasoning and Decision-Making in Multi-Agent Environments. arXiv:2506.02387 [cs.AI] https://arxiv.org/abs/2506.02387

A Additional Results

A.1 DQN

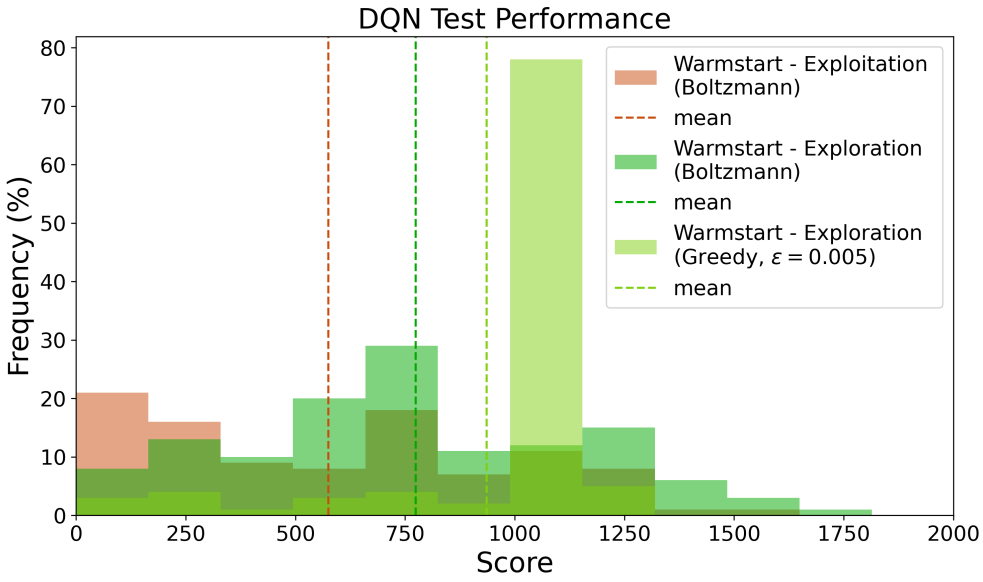


Figure 5: Distribution of final game scores from testing the DQN models on Level 1-1. Each model was tested for 100 trajectories.

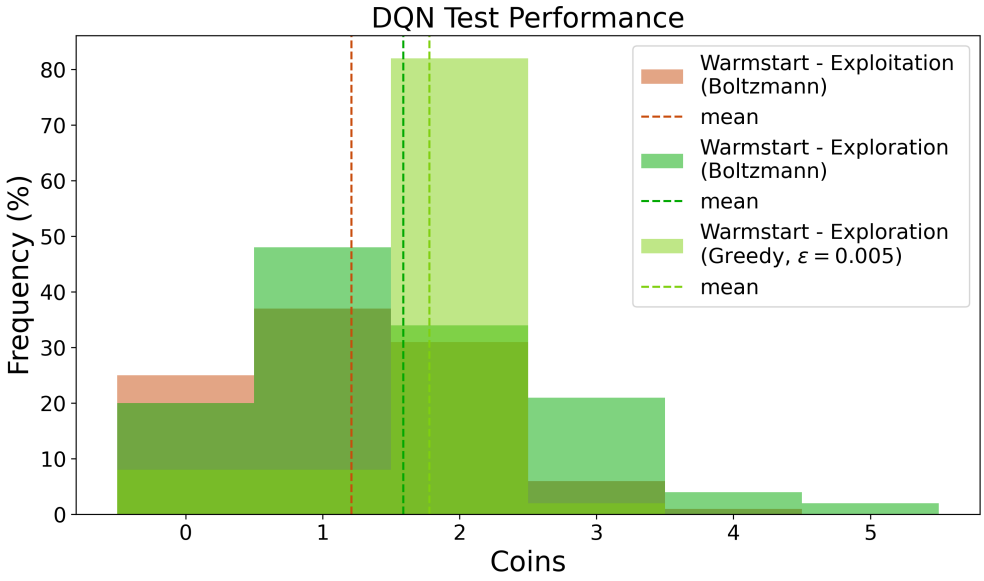


Figure 6: Distribution of coin collection from testing the DQN models on Level 1-1. Each model was tested for 100 trajectories.

A.1.1 DQN Generalization

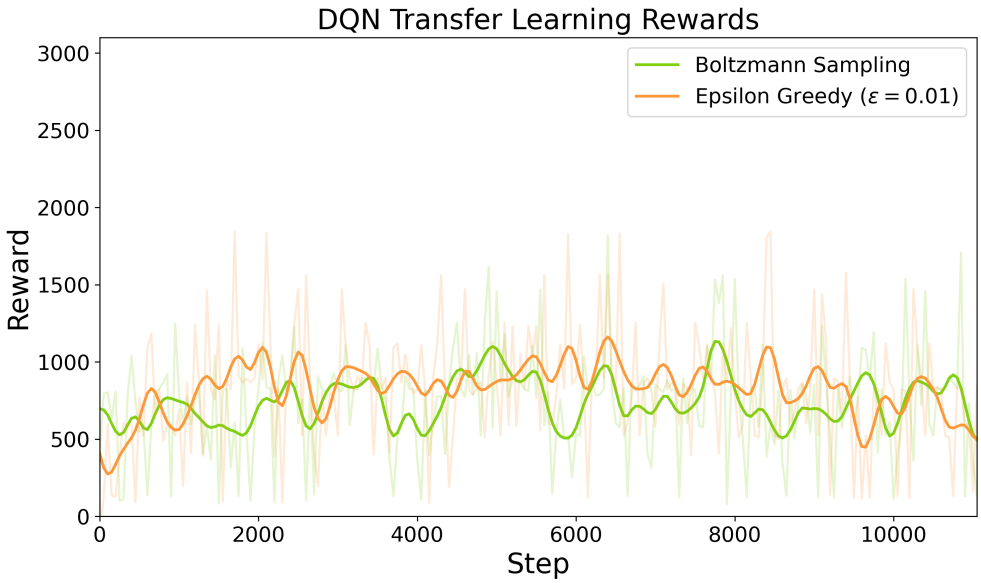


Figure 7: Environment rewards over training steps for the re-training of the Level 1-1 DQN network on Level 1-2 experiences.

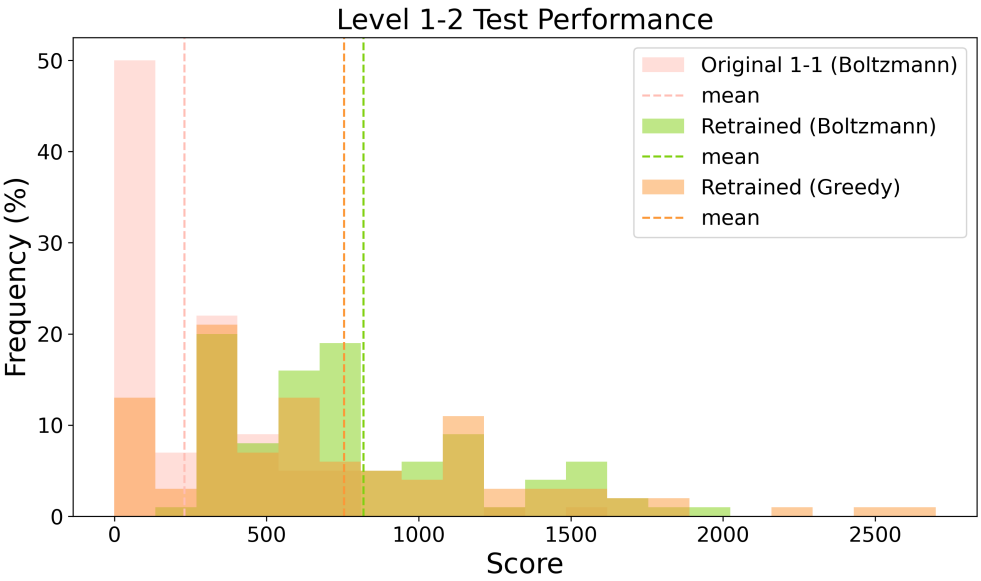


Figure 8: Distribution of final game scores from testing the DQN models on Level 1-2. Each model was tested for 100 trajectories.

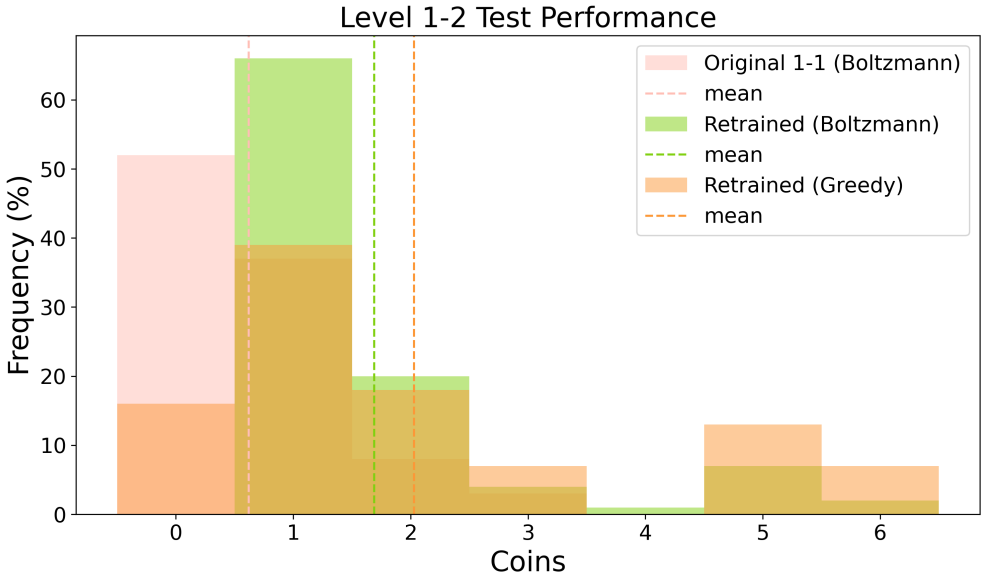


Figure 9: Distribution of coin collection from testing the DQN models on Level 1-2. Each model was tested for 100 trajectories.