Reinforcement Learning - Final Project

Raz Atiya - Nimrod Boazi

Introduction

During the second half of the semester, we explored various Deep RL methods that integrate neural network architectures with traditional reinforcement learning strategies. Our coursework focused on both the development and optimization of these algorithms, as well as a deeper understanding of key principles such as hyperparameter selection, exploration-exploitation balance, and network design for solving the MiniGrid MultiRoom Environment. These methods enable agents to interpret raw pixel observations and make informed decisions despite having only partial visibility of the environment.

Algorithm Selection and Explanation

For this task, we chose to implement the Double DQN, Dueling Double DQN and PPO algorithms.

Dueling Double DQN

Overview:

Deep Q-Networks (DQN) are a class of value based reinforcement learning algorithms that use deep neural networks to approximate the optimal action-value function. However, standard DQNs suffer from instability due to overestimation of Q-values and inefficient learning in states where action selection has little impact. Dueling Double DQN addresses these issues by incorporating two key improvements: dueling architecture and double Q-learning. The dueling architecture separates the Q-value estimation into two components: the state-value function (V), which estimates the value of a given state, and the advantage function (A), which determines the relative benefit of taking a specific action in that state. This helps the network learn which states are important without needing to evaluate every action. Double Q-learning reduces overestimation bias by using one set of Q-values to choose the best action and a separate set to estimate its value. This prevents the algorithm from consistently overestimating action values. The algorithm also employs experience replay, which stores past transitions and samples mini-batches to break temporal correlations, and an epsilon-greedy strategy to balance exploration and exploitation. These improvements make Dueling Double DQN more robust in partially observable environments, particularly those requiring long-term planning, such as MiniGrid, where the agent must navigate complex spaces.

Hyperparameters:

Hyperparameter	Value	Purpose
Learning Rate(lr)	0.0003	Balances stability and faster convergence
Discount Factor(gamma)	0.99	Encourages long-term planning
Epsilon	1.0	Starts with full exploration to gather diverse experiences.
Epsilon Decay	0.0001	Gradually shifts from exploration to exploitation.

Replay Buffer Size(mem_size)	50000	Stores diverse experiences (~500 episodes) to improve sample efficiency and prevent overfitting.
Batch Size	128	Ensures stable and efficient training updates.
Target Network Update Rate(tau)	0.005	Ensures smooth updates and prevents instability (Soft Update)

These hyperparameters promote high initial exploration, gradually transitioning to exploitation while maintaining a strong focus on long-term rewards. The large replay buffer preserves diverse and relevant experiences, ensuring the agent learns effectively without overfitting to outdated strategies. Additionally, the use of a soft update mechanism (via a small target network update rate, tau) helps maintain training stability by gradually blending new and old Q-values, preventing sudden shifts in value estimates. This smooth transition in target updates reduces variance and improves convergence, making learning more stable and efficient in dynamic environments like MiniGrid. **[See epsilon decay visualization - Figure 1], [Agent section in notebook with soft update], [Replay Buffer section in notebook]**

Deep Learning Architecture:

[See Network Architecture in notebook]

Layer	Type	Shape/Filters	Activation	Purpose
Conv1	Conv2D	32 filters, 8x8, stride 4	ReLU	Extract Basic Spatial patterns
Conv2	Conv2D	64 filters, 4x4, stride 2	ReLU	Captures mid level features
Conv3	Conv2D	64 filters, 3x3, stride 1	ReLU	Captures fine-grained spatial details
FC1	Linear	1024 neurons	ReLU	Expands high-level representations
FC2	Linear	512 neurons	ReLU	Reduces dimensionality before output
Value Stream (V)	Linear	1 neuron	-	Predicts the value of a state
Advantage Stream (A)	Linear	*n_actions* = *6* neurons	-	Estimates how each action deviates from the state-value

Our Dueling Deep Q-Network consists of three convolutional layers (Conv2D) that extract spatial features from the grid-based input, followed by fully connected layers that refine these features before splitting into two output streams: state-value (V) and action-advantage (A). The first convolutional layer uses an 8×8 kernel, chosen to align with the 8×8 game tile size of the environment, allowing the network to efficiently capture entire tiles as individual features. The stride of 4 ensures partial overlap, maintaining spatial continuity between tiles. Subsequent layers with 4×4 and 3×3 kernels progressively capture mid-level and fine-grained spatial patterns, improving the network's ability to recognize objects, walls, and paths in the grid-based structure. The dueling structure enables the network to focus on learning about important states, even when individual actions yield similar rewards—an essential feature for environments with partial observability like MiniGrid.

To enhance training efficiency, we use the RMSprop optimizer, which helps stabilize value updates, and Mean Squared Error (MSE) loss, which effectively penalizes inaccurate Q-value predictions. Our model's effectiveness was validated through metrics such as success rate, average reward, and episode length, alongside visualizations like loss curves, Q-value

distributions, and exploration-exploitation trends, **[See Visualization and Graphs in notebook for all training graphs]**. Additionally, by initially training on smaller MiniGrid environments and fine-tuning on larger ones, we ensured its adaptability and transferability. **[See loss function mapping in Figure 2]**

PPO

Overview:

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that improves policy learning through iterative updates while preventing large, destabilizing policy changes. It builds on policy gradient methods, optimizing an actor-critic framework where the actor learns a policy for action selection, and the critic estimates the value function. A key feature of PPO is the clipped surrogate objective, which restricts drastic updates to the policy by penalizing overly large deviations from previous policies, ensuring stable and efficient learning. By learning directly from trajectories, PPO enables agents to adapt to changing layouts, sparse rewards, and long-term dependencies, which are critical challenges in MiniGrid environments. Its ability to refine policies without catastrophic forgetting makes it highly effective for navigation-based tasks that require sequential decision-making over long time horizons.

Hyperparameters:

Hyperparameter	Value	Purpose
Update Frequency = N	800	Sufficient number of steps before updating the policy to allow agents to accumulate sufficient experiences before updates, preventing instability from frequent parameter changes.
Discount Factor = gamma	0.995	Determines the importance of future rewards. This relatively high discount factor encourages long-term planning, which is essential for navigation tasks with sparse rewards.
Learning rate = alpha	0.0001	Controls how quickly the model updates during training. This relatively small learning rate ensures smooth adaptation, preventing drastic policy changes that could disrupt learned behaviors during transfer from a small environment to a larger environment.
Generalized Advantage Estimation= GAE lambda	0.95	Helps balance bias and variance in advantage estimation, helping the agent handle long-term dependencies efficiently.
Policy Clipping Parameter	0.2	Limits policy updates to prevent large changes, ensuring stable training.
Batch Size	512	Number of experiences sampled per update. This large batch size improves stability by incorporating diverse experiences, reducing risk of overfitting.
Training Epochs	5	Number of times the policy updates per batch of experience. Five epochs strikes a balance between learning efficiency and avoidance of overfitting.

Most of our selections were based on suggestions in the study, *"What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study"*
[Link to study- https://arxiv.org/pdf/2006.05990]. We made small adjustments based on our environment and training technique. **[PPO Agent in notebook], [PPO Memory in notebook]**

Architecture:

[See PPO Network in notebook]

Layer	Type	Shape/Filters	Activation	Purpose
Conv1	Conv2D	32 filters, 8x8, stride 4	ReLU	Extract Basic Spatial patterns
Conv2	Conv2D	64 filters, 4x4, stride 2	ReLU	Captures mid level features
Conv3	Conv2D	64 filters, 3x3, stride 1	ReLU	Captures fine-grained spatial details
FC1 - Actor	Linear	256 neurons	ReLU	Processes extracted features for action selection
FC2 - Actor	Linear	256 neurons	ReLU	Further refines actor network representation
Actor Output	Linear	n_actions = 6 neurons	Softmax	Outputs action probabilities for policy selection
FC1 - Critic	Linear	256 neurons	ReLU	Processes extracted features for action selection
FC2 - Critic	Linear	256 neurons	ReLU	Further refines actor network representation
Critic Output	Linear	1 neuron	-	Outputs a single scalar value representing state value

For explanation of CNN layers see **[See Dueling Network Explanation]**

We use a **shared CNN** for the actor and critic networks because both require the same spatial feature representations for decision-making, allowing for efficient parameter sharing and reducing computational cost. To ensure accurate estimations for both policy (actor) and value function (critic), we provide each with two fully connected layers of 256 neurons, a choice informed by the success of our Q-network architecture while balancing performance and GPU efficiency. Since PPO requires separate policy and value function estimates, doubling the network size compared to Dueling DQN, this architecture maintains expressiveness without excessive resource usage. The actor network outputs a probability distribution over the six possible actions via softmax, allowing for stochastic policy sampling, which facilitates exploration and prevents premature convergence. Meanwhile, the critic outputs a single scalar value without activation, enabling unrestricted value estimation for both positive and negative rewards.

Comparison of Algorithms:

Aspect	Double DQN	Dueling Double DQN	PPO
Performance	Reduces Q-value overestimation, leading to more stable learning. Performs well in discrete action spaces but struggles with sparse rewards.	Enhances state representation by estimating state-value and action-advantage separately. More effective than Double DQN in complex environments.	More effective in environments with sparse rewards and complex state dependencies due to its policy gradient approach and direct trajectory learning.
Learning Efficiency	Sample-efficient due to experience replay, but can still be unstable.	More efficient than standard DQN due to improved state-value estimation.	Less sample-efficient but benefits from direct policy optimization, leading to smoother learning.
Convergence	More stable than vanilla DQN due to separate action evaluation, but still prone to overestimation errors without proper tuning.	Can be unstable if hyperparameters are not well-tuned; requires careful balancing of value and advantage estimation.	More stable due to the clipped objective, preventing drastic policy updates that cause performance degradation.
Exploration vs. Exploitation	Same as Dueling DQN: epsilon-greedy exploration with decay to transition from exploration to exploitation.	Uses epsilon-greedy exploration, which starts with high randomness and gradually shifts to greedy action selection.	Uses stochastic policy sampling, ensuring continued exploration without explicit decay.
Handling Partial Observability	does not inherently solve partial observability issues.	Struggles in partially observable environments, but estimating advantage values helps to some extent.	More adaptable to partial observability, as the policy can be trained to account for uncertainty in state information.
Action Selection	Deterministic during exploitation.	Deterministic during exploitation.	We chose to implement a greedy action selection post training.
Suitability for MiniGrid	Struggles with sparse rewards and long-term dependencies, making it less ideal for MiniGrid.	More effective than Double DQN due to better state representation, but still struggles with long-term dependencies and sparse rewards.	More suitable for MiniGrid due to its ability to handle delayed rewards, long-term planning, and partial observability.

Analysis of Approaches

Reward Shaping Approach

When approaching the solution to this environment, we first analyzed what constituted good actions (steps towards door, opening doors, entering new room, steps toward green tile) and bad actions (unnecessary movements, looping, closing doors). This analysis led us to think about the various Reinforcement Learning approaches that could effectively contribute to solving it. As a result of our effective categorization of the goodness of actions, a method that first came to mind was providing Human feedback (HF). Another method that came to mind is using Imitation learning, in this method we guide the agent as part of the training on how to solve the environment. This method fits our environment since we are able to solve this environment efficiently if we were to play ourselves. Ultimately, this analysis of actions led to our implementation of reward shaping for the Dueling Double DQN, where we assigned positive rewards for desirable actions and negative rewards for undesirable ones. This approach proved effective, as DQN relies heavily on TD updates and therefore generally responds well to frequent rewards which allow it to learn accurate Q-values more easily. Additionally, the epsilon-greedy exploration encouraged the agent to discover positively rewarding actions, making it more effective in learning optimal behaviors.

In contrast, PPO struggled with this reward shaping setup, as it converged to a local optimum of avoiding negative actions without actively identifying positive ones. We hypothesize that incorporating entropy loss to encourage exploration could have improved PPO's performance with reward shaping. Another potential improvement could be adjusting the reward values to be less negative or more balanced, which may help PPO avoid over-penalization and learn more effectively. Lowering the learning rate and number of epochs did not have any effect on avoiding this local optimum.

[Reward Shaping code in notebook], [Training Code in notebook]

Reward Structure

Penalizing Inefficiency:

Staying in the Same State (-2):

- If the agent remains in the same position for multiple steps, it receives a penalty of -2.
- This discourages unnecessary waiting, looping, or getting stuck.

Unnecessary Actions (-1):

- If the agent performs actions 3, 4, or 6, which are deemed unnecessary, it receives a penalty of -1.
- This prevents random or inefficient movements.

Encouraging Effective Door Interaction:

Opening/closing a Door:

- If the agent closes an open door, it receives a -7 penalty.
- If it successfully opens a door, it receives a +2 reward.
- We extracted the RGB arrays of opened and closed door tiles for precise state handling. **[Extraction of open and closed door tiles for comparison in notebook] [Functions for opened and closed door identification in notebook]**

Encouraging entering a new room:

Moving Forward After Opening a Door (+1 per step forward):

- After opening a door, moving forward once grants +1 reward.
- Moving forward twice after opening a door earns an additional +1 reward, reinforcing entering a new room.

Reinforcing Success

Completing the Episode (+50 Reward):

- If the agent reaches the green goal tile, the episode ends, and it receives a large reward of +50.
- This ensures that the agent prioritizes solving the environment efficiently.

The balance between the rewards is critical to ensure one does not overpower the other, for example the reward for opening a door can not be greater than the penalty for closing a door, in order to ensure that the agent does not tolerate closing doors for the purpose of later opening them. The large reward for solving the level is critical for incentivizing the agent to finish the level.

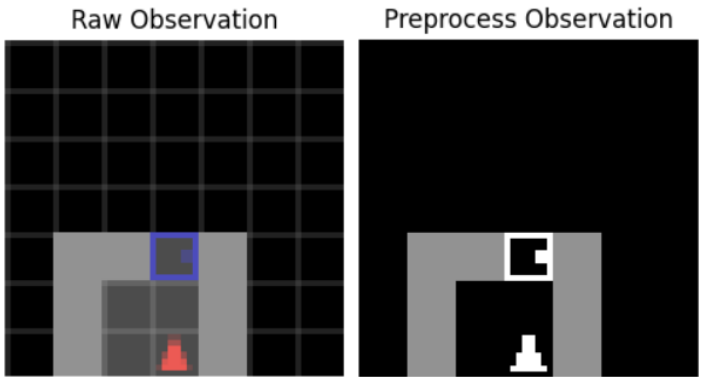
Transfer Learning Approach

While trying to solve the environment, we came up with another approach that could be good in the case of our task - Transfer Learning. At the beginning of the project, we first tried to solve the smallest environment (2 rooms) offered to us by the library (Gymnasium). When our agent learned and understood that he had to find the door in the room, open it and reach the green square, we thought to ourselves that this is exactly the way that he should act when solving the large environment. Therefore, we can start training the agent in the small environment, when he achieves good results, move to the medium environment and finally send him to the large environment. In this process, our agent learns what the correct behavior is that he should do from the small environment, apply it and transfer it to the medium and large environment as well, and in this transition he will solve the large environment. The transfer learning approach worked well with PPO, as it effectively handles sparse rewards (in MiniGrid, rewards are only received at the end of an episode). Because PPO learns directly from trajectories, it can associate early positive steps with the final reward, enabling better long-term credit assignment. **[See Figures 4-6 - PPO Training Graphs], [Training PPO section in notebook], [See Figure 3 - Different Size Environments]**

In contrast, Dueling Double DQN struggled with this approach, likely due to its random batch sampling from the experience replay, which makes it difficult to recognize beneficial actions that were not immediately rewarded. Even with its advantage estimation improvement over standard DQN, it still faced challenges in learning from delayed sparse rewards.

Preprocessing and Exploration-Exploitation

To improve the agent's learning efficiency, we preprocess the raw RGB observations by applying a two-step transformation. First, we replace all pixels except specific predefined colors with white. Then, we refine the image further by mapping all non-permissible colors to black, ensuring a consistent visual representation. The green tile is left unchanged.



This preprocessing step allows the network to extract relevant information from the environment while minimizing unnecessary noise. One significant change was converting all doors to the same color. However, in hindsight, we regret not converting the observations to grayscale, which could have reduced the input dimensions and simplified processing. Another potential improvement could be one-hot encoding colors, as the observation space only contains a limited number of distinct colors. Additionally, distinguishing between the agent's color and the door color might help improve recognition. A challenge we encountered during implementation was that the gray door handle had the same color as the grid lines, causing the gray doors to appear without handles after preprocessing. However, this did not significantly impact performance, as the agent was still able to correctly identify and interact with gray doors.The processed image is then transposed into a channel-first format (3, H, W) for compatibility with deep learning models.**[Preprocess section in notebook]**

Additionally, we implemented a frame-stacking method for our Dueling Double DQN reward shaping approach, where four consecutive preprocessed frames are concatenated along the channel dimension, forming a (12, H, W) tensor. This technique helps the agent capture temporal dependencies, improving decision-making in our partially observable environment. This enables the agent to track its recent movements, avoid unintentionally reversing direction, and retain awareness of door and wall locations. This approach also complemented our **reward shaping** strategy, as rewards were assigned based on the states of multiple sequential frames, reinforcing beneficial behaviors over time. The difference in performance between Dueling Double DQN with stack framing and without can be seen in **Figure 7.**

[Preprocess stacking section in notebook]

Exploration in the Dueling Double DQN reward shaping approach was implemented using an epsilon-greedy policy, as DQN lacks a built-in exploration mechanism. During training, we observed varying performance across runs with identical parameters, which was fixed after adjusting the balance between the learning rate and epsilon decay rate, **[See Figure 1]**

In contrast, PPO naturally handles exploration by learning a probability distribution over actions. Since the algorithm inherently incorporates stochasticity, we did not need to

introduce additional exploration mechanisms for the transfer learning approach and simply allowed the policy to explore as the agent trained. For the reward shaping approach, introducing entropy loss could have improved results as mentioned **here**.

Evaluation and Reporting

We begin by noting that, upon assessing the environment, there are levels where the most efficient solution requires 65–70 steps. Therefore, we define an efficient run as one that completes the level in fewer than 70 steps. A successful run is defined as one where the agent solves the level in fewer than 120 steps. For validation, we tested each of our trained agents on 1,000 randomly generated levels.

Dueling Double DQN and Double DQN with Reward Shaping and Stack Framing

Dueling Successful runs: 94.9%

Dueling Efficient runs: 89.8%

Double Successful runs: 90.5%

Double Efficient runs: 86.1%

See Figures 10 - 12 - Validation Graphs

Dueling Double DQN, when combined with reward shaping, provided a structured way to guide the agent's learning. This approach was particularly useful in environments where defining good and bad behaviors was straightforward, such as MiniGrid. By designing explicit rewards for actions like opening doors and penalizing inefficient movements, we were able to direct the agent toward optimal behavior. However, this approach had significant downsides. Implementing reward shaping was time-consuming, requiring extensive preprocessing, such as extracting and comparing door tile states, identifying center tiles, and parsing observations - **See Helper Functions** .Additionally, the computational cost was higher due to the frequent need for observation processing, frame stacking and updating of the policy in every step. Another drawback of this approach is that it is akin to hardcoding desired behaviors, which limits its scalability. In more complex environments, defining precise reward functions would be increasingly difficult, making this method less viable. This approach is significantly more sample efficient with only 1500 training episodes needed in order to reach convergence, see **Figure 8**. We also note that Double DQN produced performance that was similar, albeit slightly worse—probably due to the high level of guidance in our reward shaping, see comparison graphs - **Figure 9**.

PPO with Transfer Learning

Successful runs: 97.7%

Efficient runs: 92.7%

See Figures 10 - 12 - Validation Graphs

PPO with transfer learning proved to be a more flexible and scalable approach. One of its main strengths was its ability to learn directly from trajectories without requiring manually

designed rewards. This significantly reduced the complexity of implementation and allowed for a more natural learning process. By training the agent on smaller rooms and gradually transitioning to larger ones, we enabled it to generalize across different layouts effectively. PPO required significantly more training episodes to reach good performance (6,000 compared to 1,500 for Dueling Double DQN, as shown in the training graphs), making it less sample-efficient. This did not significantly impact computational efficiency, as PPO was far superior in this regard, (PPO does not require large experience replay, training steps in every episode, frame stacking). It is also important to note that without these provided MiniGrid environments we would have needed to create smaller environments ourselves, which could possibly be even more time consuming than designing a precise reward shaping function.

Conclusion

Both approaches produced similar results with PPO taking the lead slightly **[See Figures 10 - 12]**,Therefore PPO's post training performance together with its ability to adapt to sparse rewards and generalize without reward shaping during training made it the preferred choice for this environment in our eyes.

Challenges

Understanding hyperparameter selection and agent implementation was a major challenge. To address this, we referred to research papers such as *"What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study"* (arXiv:2006.05990), and online forums for Dueling Double DQN hyperparameters which provided valuable theoretical guidance. Additionally, we leveraged online resources, particularly the reinforcement learning tutorials from *Machine Learning with Phil* on YouTube, which offered practical insights and implementation examples that helped guide our implementation of the agents https://www.youtube.com/@MachineLearningwithPhil

Another major challenge was designing an effective reward shaping structure. To tackle this, we first theorized the key behaviors that should be incentivized and then refined our approach through iterative trial and error, adjusting rewards based on observed agent performance.

Additionally, this project marked our first experience working with neural networks, making the learning curve particularly steep. Since neural networks were fundamental to this project, we dedicated time to researching online resources, studying theoretical concepts, and analyzing existing implementations to build a solid understanding. This allowed us to effectively apply deep reinforcement learning techniques to our problem.

Visualization and Graphs

Figure 1 - Dueling Epsilon Decay

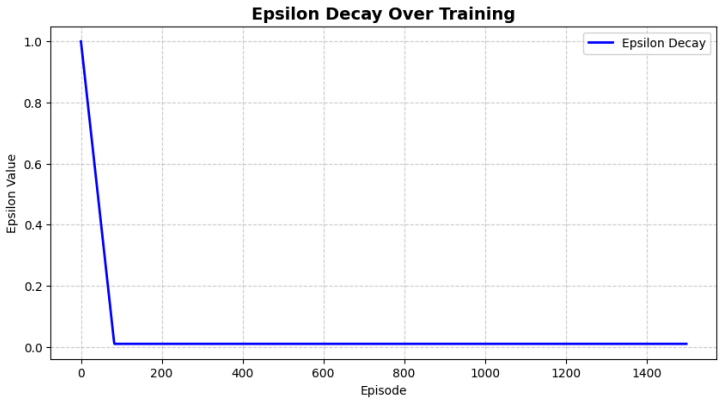


Figure 2 - Dueling Loss Function

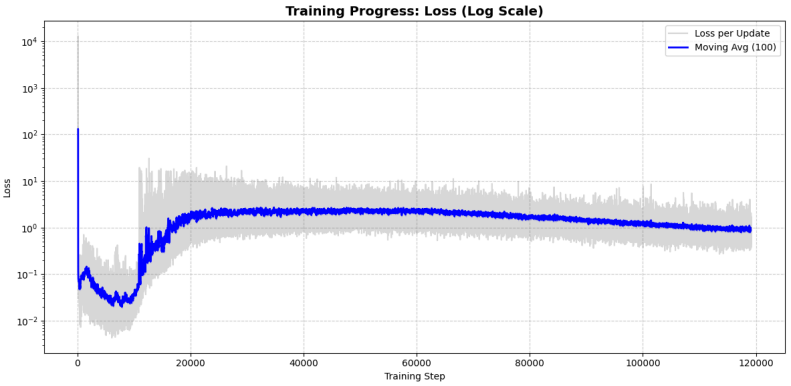


Figure 3 - MiniGrid Environments

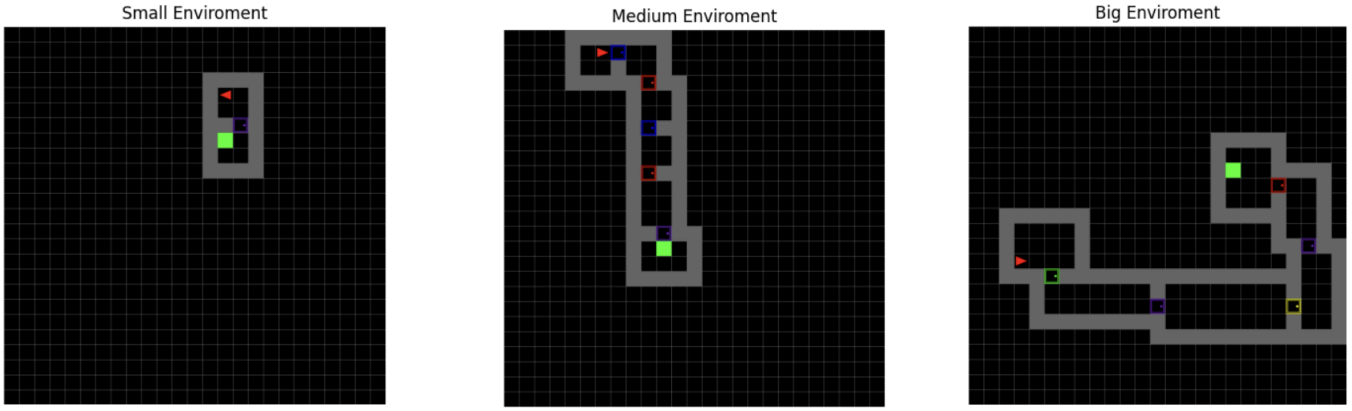


Figure 4 - PPO - Small Environment

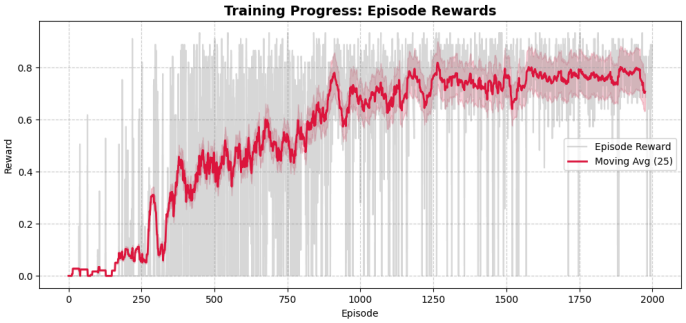


Figure 5 - PPO - Medium Environment

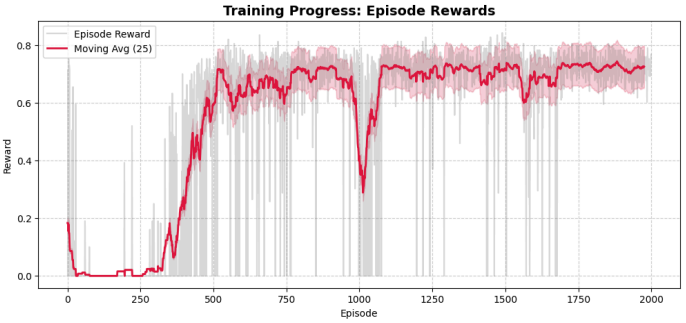


Figure 6 - PPO - Large Environment

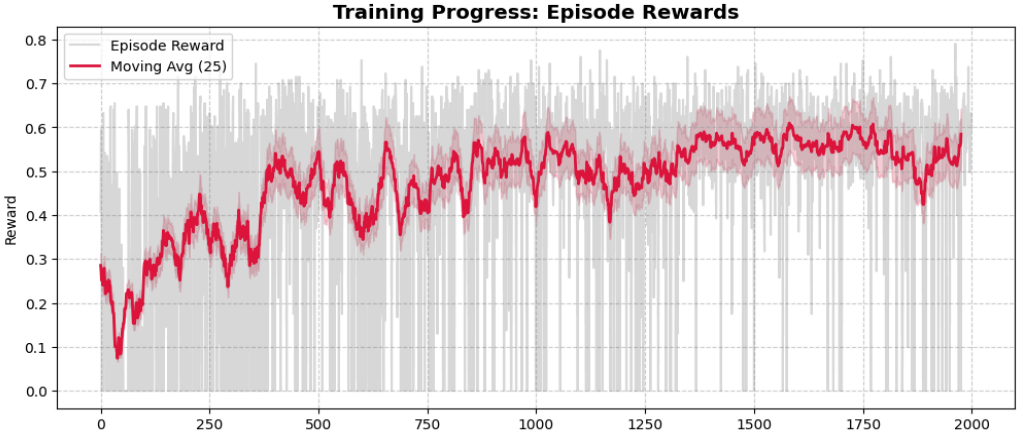


Figure 7 - Dueling Comparison with/without stacking

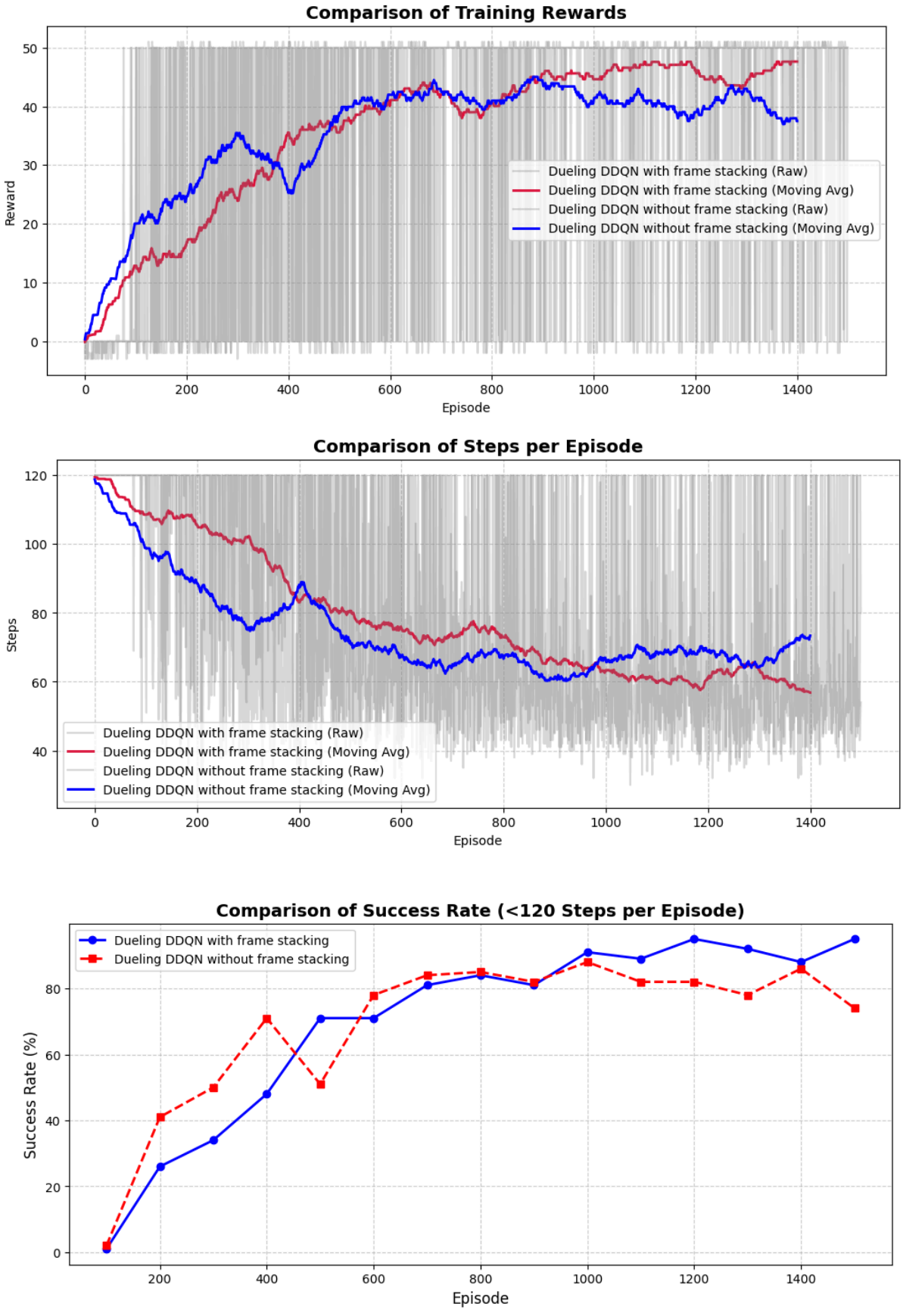


Figure 8 - Dueling DQN Training Graphs

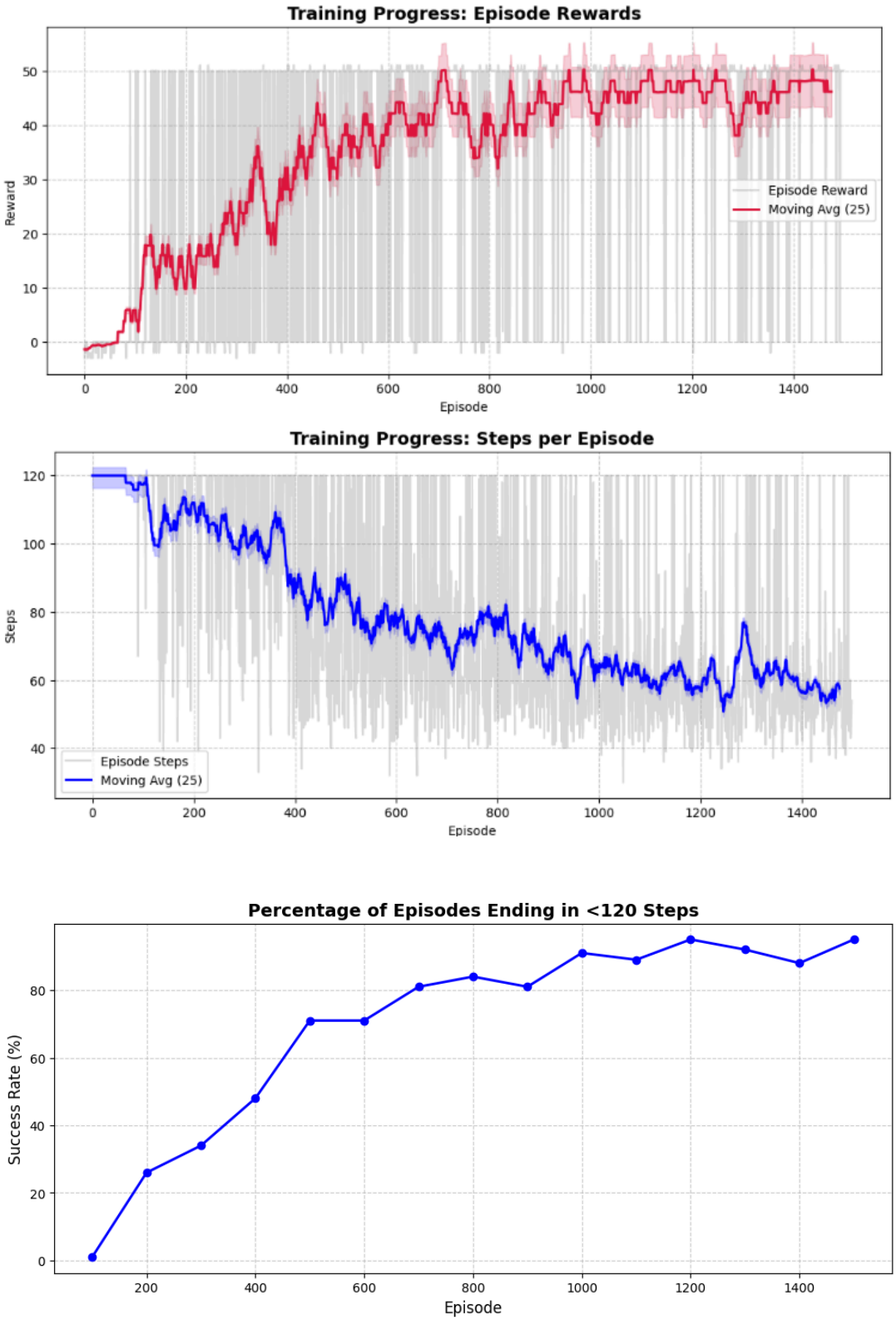
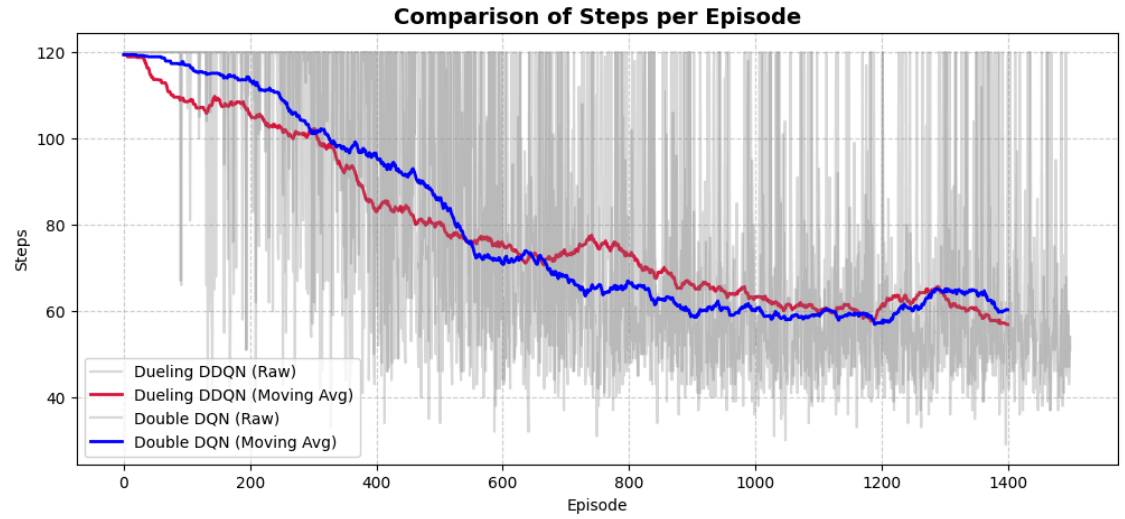
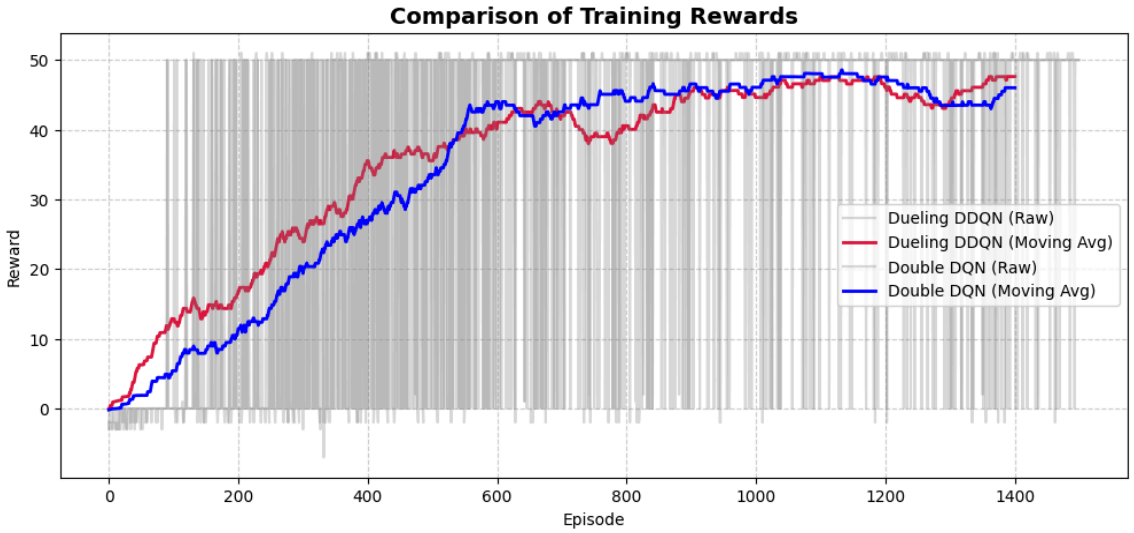


Figure 9 - Comparison Dueling Double





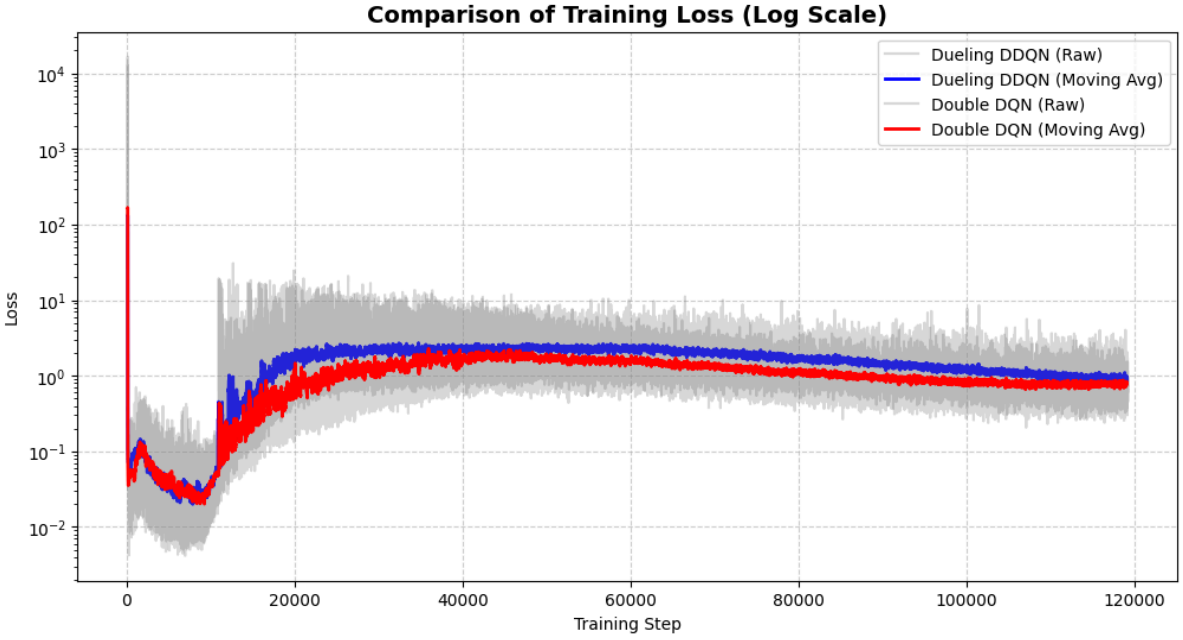


Figure 10 - Double DQN Validation

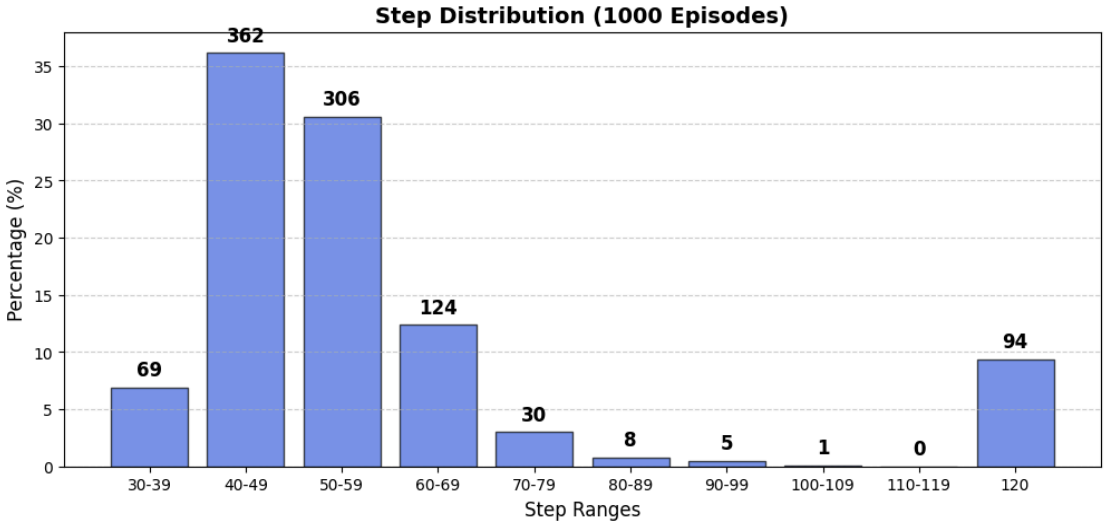


Figure 11 - Dueling DDQN Validation

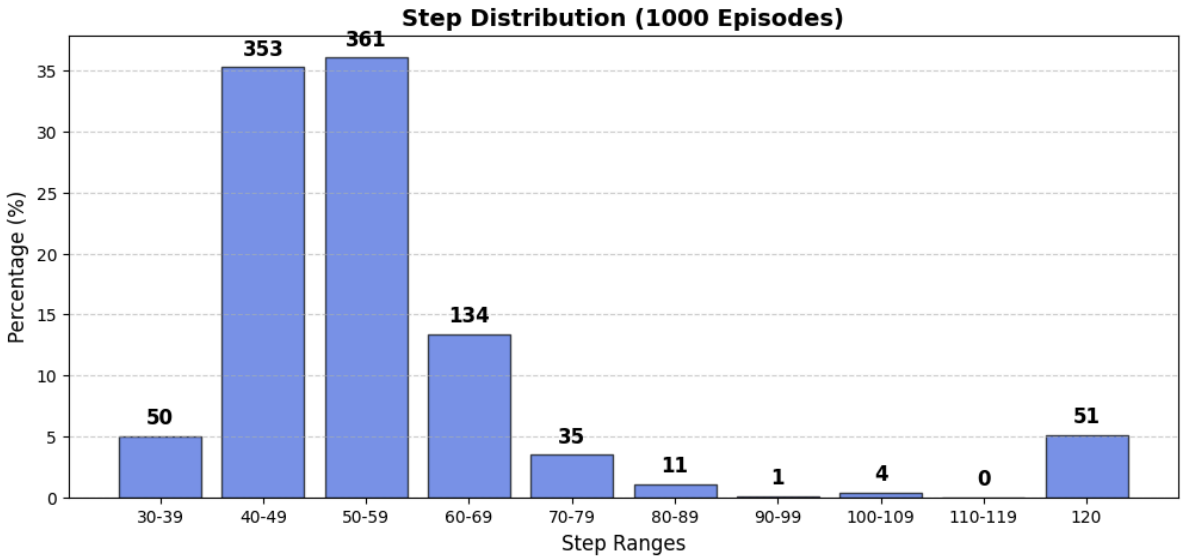


Figure 12 - PPO Validation