Detailed Rainbow DQN Reimplementation with Observations

Abstract

In this paper, we reimplement the Rainbow Deep Q-Network (DQN), a state-of-the-art reinforcement learning algorithm, on a smaller computational scale while maintaining competitive performance. Rainbow DQN integrates several key advancements over the original DQN, including Double DQN, Prioritized Experience Replay, Dueling Networks, Multi-Step Learning, Distributional RL, and Noisy Networks. We demonstrate the successful application of these components on an Apple Silicon M3 Pro machine with limited resources.

Additionally, we explore the effectiveness of transfer learning by initializing agents with pre-trained models from different Atari games. This technique accelerated training and improved performance across different tasks. Our experiments reveal that Rainbow DQN consistently outperforms traditional Double DQN, particularly in complex environments. Furthermore, the sensitivity of the algorithm to hyperparameters, such as Target Update Frequency, highlights the importance of tuning specific settings for optimal performance. Our results underscore the potential for high-performing reinforcement learning agents in resource-constrained environments.

Introduction

Reinforcement Learning (RL) is a branch of machine learning where agents learn to make decisions by interacting with an environment and receiving feedback in the form of rewards. Over time, agents learn to maximize cumulative rewards by identifying optimal policies that map states to actions. Atari 2600 games have become a popular benchmark for testing RL algorithms, as they present diverse and complex environments that challenge agents to develop sophisticated strategies.

Deep Reinforcement Learning (DRL) extends RL by incorporating deep neural networks to handle high-dimensional sensory inputs such as raw pixel data. One of the foundational algorithms in DRL is the Deep Q-Network (DQN), which estimates the value of actions in particular states (Q-values) and updates these estimates based on experiences. However, the original DQN suffers from limita-

tions such as overestimation of Q-values, inefficient exploration, and difficulty in handling complex reward structures.

To address these issues, the Rainbow DQN algorithm was introduced, which combines several key improvements into a single architecture. These include Double DQN, Prioritized Experience Replay, Dueling Networks, Multi-Step Learning, Distributional RL, and Noisy Networks. Each of these components addresses specific weaknesses in the original DQN and collectively enhances the learning efficiency and performance of RL agents, particularly in complex environments like Atari games.

In this work, we focus on reimplementing Rainbow DQN on a smaller computational setup while maintaining the algorithm's performance benefits. Our experiments were conducted on an Apple Silicon M3 Pro machine with 18 GB of unified memory, demonstrating that it is possible to achieve competitive results without the need for large-scale computational resources. Furthermore, we explore the use of transfer learning to enhance agent performance across different games by leveraging pre-trained models.

Our goal is to showcase the feasibility of training effective deep reinforcement learning agents on smaller hardware setups while achieving results comparable to those presented in the original Rainbow DQN paper.

Background

Deep Q-Network (DQN)

The Deep Q-Network (DQN), introduced by Mnih et al. (2015), represents a significant advancement in the application of deep learning to reinforcement learning tasks. By utilizing raw pixel data as input, DQN was able to approximate Q-values for Atari 2600 games, enabling agents to learn effective policies directly from high-dimensional sensory data. The network trained on these inputs to predict the value of various actions in a given state, allowing the agent to select actions that maximize expected future rewards.

Despite its initial success, DQN presents several notable limitations. One prominent issue is the **overestimation of Q-values**, where the network tends to overvalue actions, resulting in suboptimal policies. Additionally, the **-greedy exploration strategy** employed by DQN often proves inefficient, particularly in environments with sparse rewards, as it fails to sufficiently explore the action space. Finally, **instability in training** is another critical challenge. The high correlation between consecutive experiences and the instability of Q-value targets often cause DQN to struggle with convergence.

To address these challenges, several extensions to the original DQN have been proposed, each aimed at improving the stability, efficiency, and performance of the learning process.

Rainbow DQN

Rainbow DQN, introduced by Hessel et al. (2018), integrates several of these key extensions into a unified framework, each addressing specific limitations of the original DQN algorithm. The components incorporated into Rainbow DQN include Double Q-Learning, Prioritized Experience Replay, Dueling Network Architecture, Multi-Step Learning, Distributional Reinforcement Learning, and Noisy Networks.

Double Q-Learning, proposed by Van Hasselt et al. (2016), mitigates the overestimation bias inherent in standard Q-learning. By decoupling action selection from action evaluation, Double Q-Learning ensures more accurate value estimates, leading to more reliable policy evaluation and ultimately more stable learning.

Prioritized Experience Replay (Schaul et al., 2016) improves upon the uniform sampling technique used in traditional experience replay. By prioritizing transitions with higher temporal-difference (TD) errors, this approach enables the agent to focus on experiences that have the greatest impact on learning, accelerating convergence.

The **Dueling Network Architecture** (Wang et al., 2015) separates the estimation of state values from the advantages of actions. By explicitly distinguishing between these two components, the network can more effectively determine which states are inherently valuable, regardless of the action taken, enhancing learning efficiency.

Multi-Step Learning accumulates rewards over multiple steps before updating the Q-values. By considering longer-term consequences of actions, this approach provides a richer learning signal, leading to faster convergence, especially in environments with delayed rewards.

Distributional Reinforcement Learning, as proposed by Bellemare et al. (2017), introduces a fundamental shift in how future rewards are modeled. Rather than estimating a single expected return, the algorithm learns the entire distribution of possible returns. This distributional approach gives the agent a more nuanced understanding of the variability and uncertainty in the rewards associated with different actions, resulting in improved decision-making.

Finally, **Noisy Networks** (Fortunato et al., 2017) replace traditional exploration strategies like -greedy with a learned stochastic exploration mechanism. By introducing noise directly into the network's weights, Noisy Networks enable more effective exploration of the environment without the need for manually tuned exploration parameters.

While Rainbow DQN successfully combines these components to address the limitations of the original DQN, we chose to exclude the **Advantage Actor-Critic (A2C)** algorithm in this work due to its computational complexity and its need for more substantial resources. Our focus is on techniques that can be

efficiently implemented and run on smaller-scale computational setups.

Transfer Learning in Reinforcement Learning

Transfer learning in reinforcement learning seeks to leverage the knowledge gained from one task to accelerate learning in a related task. In the context of Atari 2600 games, this can be particularly useful as many games share common visual and dynamical features, allowing learned representations from one game to be applied to another. Transfer learning has been shown to significantly reduce training time by reusing these shared features, thereby improving the efficiency of deep reinforcement learning agents.

In this work, we apply transfer learning by initializing agents with pre-trained weights from a source game and fine-tuning them on a target game. Specifically, the convolutional layers from the source game are reused to accelerate the learning process in the target game, while the final layers are retrained. This method not only speeds up convergence but also demonstrates the potential for reusing learned representations across tasks in deep reinforcement learning.

Experimental Methods

Atari Environment

We evaluated our Rainbow DQN implementation on two Atari 2600 games: **Breakout** and **Assault**, each providing distinct challenges and action spaces. These games were selected as benchmarks due to their suitability for evaluating the performance of reinforcement learning algorithms. Breakout involves controlling a paddle to deflect a ball and destroy bricks. The agent must learn to strategically aim the ball while avoiding missing it, which would result in the loss of a life. Assault, on the other hand, introduces a more complex environment, where the player pilots a spaceship, shooting enemies while dodging attacks. The dynamic enemy patterns and the required strategic movement make Assault significantly more challenging. Both games involve high-dimensional observation spaces derived from raw pixel data, with each frame consisting of a 210×160 pixel RGB image. These frames were preprocessed before being input into the network to reduce computational overhead while preserving essential information.

Observation and Action Spaces

The observation space in both games consists of sequences of frames representing the visual state of the game. To reduce computational complexity, the frames are resized to 84×84 pixels and converted to grayscale. This reduces the dimensionality of the input while retaining crucial visual details. The pixel values are normalized to a range between [0, 255] and [0, 1], ensuring that the agent can efficiently process the game states without unnecessary memory usage or computational overhead.

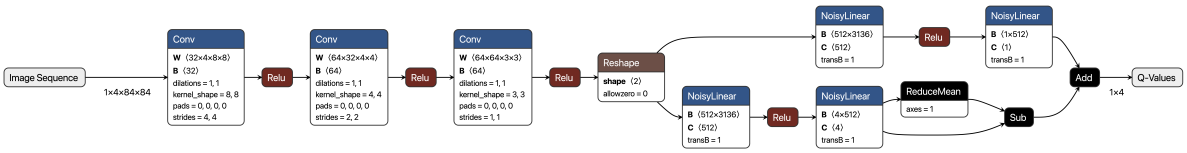

Figure 1: Rainbow DQN Architecture Graph

In Breakout, the action space is comprised of four discrete actions: the agent can either remain stationary, launch the ball into play, move the paddle to the right, or move the paddle to the left. By contrast, Assault presents a more complex action space with seven discrete actions. The agent can choose to remain stationary, fire a projectile, move in any of several directions while firing, or perform a combination of movements and firing actions. This action space reflects the greater complexity and strategic depth required in Assault compared to Breakout.

Network Architecture

Our implementation of Rainbow DQN is based on the original DQN architecture, with several modifications to incorporate the enhanced components of Rainbow DQN. The network processes raw pixel inputs through a series of convolutional layers, followed by fully connected layers that output Q-values for each action in the game's action space.
The architecture is detailed below:

The detailed description of components can be found in Figure 5 in appendix.

The network's convolutional layers are responsible for extracting visual features from the game frames, which are then passed through fully connected layers to produce Q-value estimates for each possible action. The architecture is enhanced with several key components. The dueling network architecture separates the final fully connected layers into two streams, one estimating the state value and the other estimating the advantage of each action. These are combined to produce the final Q-values. Noisy Networks are used to replace standard linear layers, introducing stochastic exploration directly into the network's parameters. Distributional Q-learning modifies the output layer to predict a distribution over discrete atoms representing possible future rewards, rather than a single Q-value estimate.

Hyperparameters

To ensure fair comparisons and optimal performance, we carefully tuned hyperparameters based on those used in the original Rainbow DQN paper.

The key hyperparameters for Double DQN and Rainbow DQN configurations can be found in Figure 2 in appendix.

In Double DQN, exploration is controlled by an ϵ-greedy policy, where ϵ decays from 1.0 to 0.1 over time. In Rainbow DQN, exploration is managed by Noisy Networks, eliminating the need for manually tuned parameters. Rainbow DQN also employs distributional Q-learning, where V_{min} and V_{max} define the range of possible returns, and N_{atoms} is the number of discrete points in the distribution. Prioritized experience replay is controlled by , which adjusts the degree of prioritization, while compensates for importance-sampling bias.

Environment Wrappers

To improve learning efficiency and agent performance, several environment wrappers were implemented using the Gymnasium framework. A No-Op Reset was used at the start of each episode, where a random number (between 1 and 30) of no-operation (NOOP) actions were executed to randomize the initial state. This prevents the agent from overfitting to specific starting conditions. In games like Breakout, where the agent must press FIRE to initiate gameplay, a Fire-On-Reset wrapper was implemented to automatically execute this action at the start of the game or after the agent loses a life. Frame skipping was employed to reduce computational load by repeating the selected action for a fixed number of frames (typically 4). This allows the agent to focus on more strategic actions rather than frame-by-frame decisions. Additionally, frame stacking was used to provide the agent with temporal context. The last four frames were stacked along the channel dimension, creating an input shape of (4, 84, 84), allowing the agent to infer motion and velocity. Lazy frame stacking was implemented to minimize memory usage, as references to the original frames were stored instead of duplicating the frame data.

Training and Evaluation Protocol

The training phase consisted of 5 million steps per game, with the agent performing online updates using experiences sampled from the prioritized replay buffer. Two different target update frequencies (TUF), 32,000 and 10,000, were used to evaluate their impact on performance. The agents were trained with both Double DQN and Rainbow DQN configurations for comparison. After training, agents were evaluated over 500 episodes for each game using the same hyperparameters and environment settings as those used during training. A warm-up phase of 100 episodes was performed prior to formal evaluation to allow the agent to stabilize its performance. Performance was measured in terms of the cumulative rewards achieved across these evaluation episodes.

Results

Training

The performance of our Rainbow DQN implementation was evaluated in terms of both training and testing metrics. Figure 1 shows the training curves for both **Breakout** and **Assault**, comparing the performance of Double DQN and

Rainbow DQN with two different target update frequencies (TUF), alongside random and human performance baselines. Additionally, transfer learning experiments are included, where a Rainbow DQN agent pre-trained on one game is transferred to another.

Full Training Curves containg Mean Q Value and Loss can be found in Figure 4 in appendix.

Testing

See Figure 1 in appendix.

Analysis

Training Behavior and Remarks

Assault In the game of **Assault**, Rainbow DQN consistently outperformed Double DQN across both Target Update Frequency (TUF) settings. The Rainbow agent demonstrated faster learning and achieved higher rewards throughout training, underscoring the effectiveness of the multiple improvements integrated into the Rainbow architecture, such as Noisy Networks and Distributional Reinforcement Learning. These enhancements facilitated more efficient exploration and improved policy learning in the complex dynamics of Assault. In contrast, the Double DQN agent plateaued early in training, irrespective of the TUF used. This stagnation suggests that the simpler Double DQN architecture struggled to fully capture the game's intricate patterns and the broader action space. Consequently, the learning process for Double DQN stabilized at suboptimal performance levels.

The effect of TUF was particularly pronounced in **Assault**, where a higher TUF setting of 32,000 yielded better results for both agents. A higher TUF may help reduce the risk of overfitting by updating the target network less frequently, thereby providing a more stable and consistent learning signal. While Rainbow DQN showed robust performance across both TUF settings, Double DQN did not exhibit significant improvement with a lower TUF (10,000), indicating that its simpler architecture was less responsive to more frequent target network updates.

Breakout In **Breakout**, Double DQN initially outperformed Rainbow DQN during the first 5 million training steps. This suggests that in simpler environments like Breakout, the absence of more sophisticated mechanisms such as Distributional RL may be advantageous for Double DQN, due to the reduced complexity and faster adaptation. However, over time, Rainbow DQN demonstrated the potential to surpass Double DQN with extended training, a trend consistent with the findings from the original Rainbow DQN paper. The long-

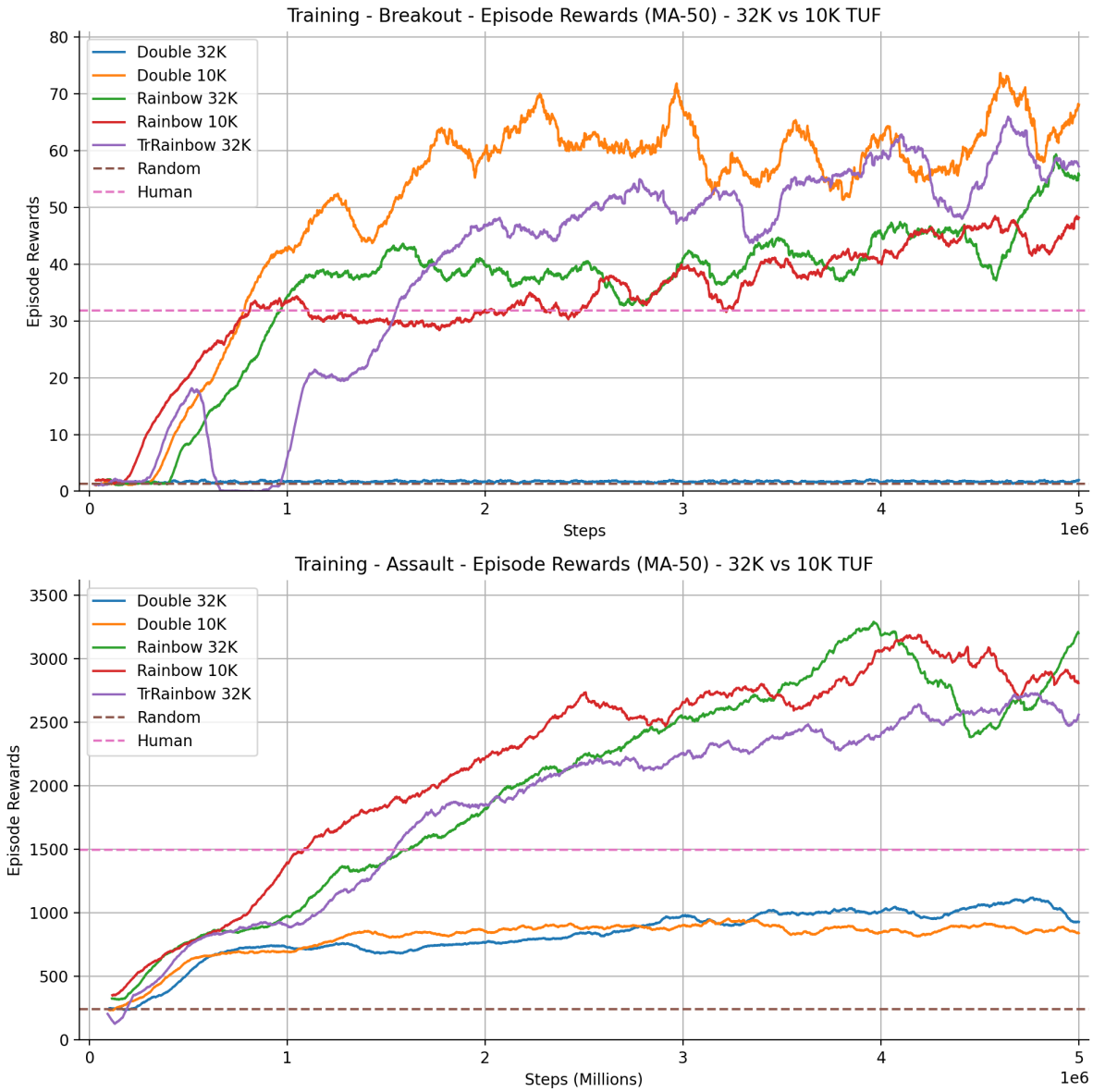

Figure 2: Training Curves

term benefits of Rainbow's advanced components, particularly Distributional RL and Noisy Networks, became more evident as training progressed.

Rainbow DQN exhibited superior performance with a higher TUF (32,000) over the long term. In simpler games like Breakout, less frequent target network updates appeared to prevent overfitting and contributed to improved generalization. The more stable and reliable target network provided by the higher TUF setting played a key role in Rainbow DQN's superior performance during the latter stages of training. In contrast, Double DQN benefitted from a lower TUF (10,000) in the early stages of training, as more frequent updates allowed for quicker adaptation to the game's mechanics. However, over time, this led to underfitting, with the agent over-adapting to recent experiences without adequately generalizing across the broader environment.

General Observations Across both games, the lower TUF facilitated quicker initial learning and convergence. However, the longer-term results indicated that higher TUF settings produced better overall performance, particularly by reducing the risk of overfitting. The benefits of reduced overfitting provided by the less frequent target updates outweighed the early gains achieved from faster learning under a lower TUF.

Loss and Q-value trends revealed interesting insights into the agent's learning process. Notable dips in loss and Q-values corresponded with episodes where the agent achieved high scores, such as reaching 300 points in Breakout. These fluctuations suggest that the network was actively adjusting its estimations in response to high-reward experiences, reflecting the agent's ongoing learning and adaptation in real-time.

Direct comparisons between the loss values for Double DQN and Rainbow DQN were not straightforward due to differences in their loss functions. Double DQN employed the Huber loss, while Rainbow DQN used a cross-entropy loss aligned with its distributional approach. Consequently, the magnitude and trajectory of losses between the two algorithms varied, making direct comparison challenging. Additionally, Rainbow DQN exhibited lower mean Q-values compared to Double DQN, which can be attributed to its distributional nature. By focusing on learning the entire distribution of possible returns, Rainbow DQN tends to offer more conservative value estimates, requiring additional updates to build confidence in the long-term reward structure.

Training Dynamics and Model Adaptation The variation in performance observed across different TUF settings provides valuable insights into overfitting and underfitting. In general, lower TUF settings (10,000) appeared to increase the risk of overfitting, particularly in Double DQN. Frequent target network updates caused the agent to adapt too closely to recent experiences, which impaired its ability to generalize effectively. Conversely, higher TUF settings (32,000) helped mitigate overfitting by maintaining a more stable target for

the network. This allowed the agent to generalize from a broader range of experiences, resulting in more robust learning and better long-term performance.

The effect of TUF also appeared to be game-dependent. In more complex games such as **Assault**, higher TUF settings proved beneficial, as they provided more stable targets that allowed the agent to navigate the intricate game dynamics more effectively. In simpler games like **Breakout**, lower TUF settings promoted more frequent updates, which enabled simpler architectures like Double DQN to adapt quickly to the game's mechanics. However, this also resulted in underfitting over time, limiting the agent's capacity to generalize across a broader set of experiences.

Alignment with Original Rainbow DQN Paper Our results align closely with the findings reported in the original Rainbow DQN paper. In **Breakout**, Rainbow DQN initially underperformed relative to simpler algorithms like Double DQN, but surpassed them with extended training. This outcome suggests that the benefits of Rainbow's advanced components, including Distributional RL and Noisy Networks, become more pronounced over longer training durations. The small performance difference between Double DQN and Rainbow DQN in Breakout mirrors the observations made in the original paper.

In **Assault**, Rainbow DQN exhibited clear advantages over Double DQN, reaffirming the effectiveness of the integrated enhancements in more complex environments. This significant performance gap between Double DQN and Rainbow DQN was also observed in the original paper, further supporting the notion that Rainbow's advanced components are particularly advantageous in environments requiring more sophisticated decision-making strategies.

Conclusion of Training Results

Our experiments highlight the critical role of hyperparameter tuning, particularly Target Update Frequency (TUF), in the training of deep reinforcement learning agents. Rainbow DQN's superior performance in complex environments like Assault, along with its potential to surpass Double DQN in simpler environments like Breakout, underscores the importance of integrating multiple improvements into a single architecture. Additionally, the observed sensitivity of different algorithms to TUF emphasizes the need for tailored approaches based on the complexity of the environment and the architecture used.

By examining these dynamics, we offer insights into how training parameters and architectural choices impact learning efficiency and overall effectiveness, contributing to a deeper understanding of reinforcement learning practices.

Additional Experiments

Hyperparameter Tuning with Optuna

In an effort to refine our model's performance, we employed Optuna, a hyperparameter optimization framework, to explore tuning beyond the Target Update Frequency (TUF). Optuna's flexibility allows for efficient exploration of hyperparameters using techniques such as tree-structured Parzen estimators. However, despite the advantages offered by Optuna, our ability to obtain exhaustive and significant results was constrained by computational limitations. The extensive hyperparameter search space required for fine-tuning reinforcement learning models, combined with limited resources, rendered it impractical to fully explore the potential improvements. Consequently, while some initial insights were gathered, more comprehensive hyperparameter optimization remains an area for future investigation.

Environment Vectorization

To address the time-consuming nature of training and evaluation in reinforcement learning, we explored environment vectorization as a method for accelerating the learning process. By running multiple environments in parallel, vectorization has the potential to substantially reduce training times and allow for more robust testing.

For training, we experimented with parallelizing up to 128 agents. However, synchronizing the replay buffer, which updates every four steps, presented significant challenges. The asynchronous behavior of parallel agents complicates the consistent management of experience replay, particularly in ensuring that the buffer maintains a representative distribution of experiences across agents. While this limitation impeded our ability to fully capitalize on parallel training, it also suggests potential solutions for future work. Implementing algorithms such as Advantage Actor-Critic (A2C) or Distributed Prioritized Experience Replay (Ape-X) could mitigate these issues by enabling more efficient buffer management and parallelism, though these approaches were outside the scope of this study.

For testing purposes, we successfully implemented vectorized environments, allowing us to evaluate agents more efficiently. Specifically, we conducted evaluations over 600 games, with each agent playing 100 warm-up games followed by 500 evaluation games. This approach enabled us to gather robust statistical data on agent performance across a significant number of trials, contributing to a more comprehensive understanding of the model's generalization capabilities.

Limitations and Future Work

Our study opens several avenues for further exploration and development in deep reinforcement learning. One key direction for future research is to extend the training duration beyond the 5 million steps used in this study. Increasing

the number of training steps could offer deeper insights into long-term learning behaviors and further enhance the performance of the agent, particularly in more complex environments.

Another promising area of investigation is expanding the set of Atari games used for testing. By evaluating the generalizability of our implementations across a broader range of games, we can better assess the robustness of the Rainbow DQN architecture and identify game-specific challenges that may arise in different environments. This approach would provide a clearer understanding of the algorithm's adaptability to diverse scenarios.

Additionally, conducting component ablation studies would allow for a more detailed analysis of the contributions made by each individual improvement integrated into the Rainbow DQN. By systematically adding or removing components such as Distributional Q-learning or Prioritized Experience Replay, future work could quantify their individual effects on performance and uncover potential synergies between them.

Efficient hyperparameter optimization also remains a critical area for improvement. The development of faster and more efficient tuning methods, possibly leveraging advanced optimization algorithms or distributed computing resources, could greatly accelerate the search for optimal configurations. This would reduce computational overhead and enhance the scalability of deep reinforcement learning systems.

Parallel training across multiple games represents another exciting direction. By training a single agent on several games simultaneously, researchers could encourage the development of more generalized policies and shared representations, fostering the ability of the agent to generalize across different tasks. This approach could also reduce the overall training time by allowing the agent to transfer knowledge between games with similar features or dynamics.

Incorporating the Advantage Actor-Critic (A2C) algorithm presents an opportunity to further improve learning efficiency, especially in parallelized environments. A2C's ability to balance between value-based and policy-based methods could yield gains in training speed and stability, particularly when combined with other elements of the Rainbow DQN architecture.

Another potential enhancement is investigating the effect of increasing frame stack sizes. Larger frame stacks, such as stacking 32 frames instead of 4, could improve the agent's ability to model temporal dependencies. This could be particularly impactful in games like **Breakout**, where an accurate understanding of the ball's trajectory and speed is critical for successful gameplay.

Lastly, exploring the use of depthwise separable convolutions could help reduce model complexity and computational cost without sacrificing performance. This approach has shown promise in other areas of deep learning and could be an effective means of streamlining the architecture while maintaining high levels of accuracy and efficiency.

References

1. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). *Human-level control through deep reinforcement learning.* Nature, 518, 529–533. Nature
2. Hessel, M., Modayil, J., van Hasselt, H., et al. (2018). *Rainbow: Combining Improvements in Deep Reinforcement Learning.* Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). arXiv
3. Fortunato, M., Azar, M. G., Piot, B., Menick, J., Blundell, C., Legg, S., & Wierstra, D. (2017). *Noisy Networks for Exploration.* arXiv
4. Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). *Prioritized Experience Replay.* arXiv
5. Van Hasselt, H., Guez, A., & Silver, D. (2016). *Deep reinforcement learning with double Q-learning.* In Proceedings of AAAI Conference on Artificial Intelligence (pp. 2094–2100).
6. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). *Dueling Network Architectures for Deep Reinforcement Learning.* arXiv
7. Bellemare, M. G., Dabney, W., & Munos, R. (2017). *A Distributional Perspective on Reinforcement Learning.* arXiv
8. Sutton, R.S. (1988). *Learning to predict by the methods of temporal differences.* Machine Learning, 3(1), 9–44. DOI
9. Chen, Y., Liu, Z., Yan, J., Li, H., Jin, O., & Yang, Q. (2020). *Pre-training Tasks for Embedding-based Large Language Models.* In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 3755-3765).

Appendix

Breakout The results for **Breakout** are summarized below, with performance measured across multiple runs using both a 32K and 10K TUF. These metrics include the mean score, standard deviation, minimum score, and maximum score achieved by the agent during evaluation.
With a **32K Target Update Frequency**, the performance is as follows:

Agent	Mean Score	Standard Deviation	Minimum Score	Maximum Score
Random	1.364	1.394	0.0	7.0
Human	31.8	-	-	-
Double DQN	2.028	2.560	0.0	9.0
Rainbow DQN	60.512	33.293	8.0	353.0
TrRainbow DQN	71.668	37.784	18.0	314.0

When trained with a **10K Target Update Frequency**, the results are as follows:

Agent	Mean Score	Standard Deviation	Minimum Score	Maximum Score
Random	1.364	1.394	0.0	7.0
Human	31.8	-	-	-
Double DQN	163.254	78.810	30.0	385.0
Rainbow DQN	58.594	22.192	23.0	257.0

Assault Similarly, the results for **Assault** were evaluated using the same metrics. The performance with a **32K Target Update Frequency** is as follows:

Agent	Mean Score	Standard Deviation	Minimum Score	Maximum Score
Random	242.382	74.342	63.0	504.0
Human	1496.0	-	-	-
Double DQN	1673.888	544.315	651.0	4158.0
Rainbow DQN	4883.154	2096.328	871.0	9792.0
TrRainbow DQN	3104.678	963.984	849.0	7363.0

Finally, the results with a **10K Target Update Frequency** are summarized below:

Agent	Mean Score	Standard Deviation	Minimum Score	Maximum Score
Random	242.382	74.342	63.0	504.0
Human	1496.0	-	-	-
Double DQN	2144.93	640.198	744.0	5098.0
Rainbow DQN	4750.356	1852.385	776.0	9073.0

Figure 3: Testing Results

Hyperparameter	Double DQN	Rainbow DQN
Learning Rate	0.00025	0.00025
Discount Factor (γ)	0.99	0.99
Replay Memory Size	300,000	300,000
Mini-Batch Size	32	32
Target Update Frequency	32,000 or 10,000	32,000 or 10,000
Frame Skip	4	4
Replay Start Size	80,000	80,000
Save Frequency	500,000	500,000
Vmin	N/A	-10
Vmax	N/A	10
Number of Atoms	N/A	51
N-Step	N/A	3
Alpha (α)	N/A	0.5
Betastart	N/A	0.4
Betaframes	N/A	Max Steps - Replay Start Size

Figure 4: Training Hyperparameters

Component	Layer Type	Input Dimension	Output Dimension
Input	-	(4, 84, 84)	-
Conv Layer 1	Conv2D	(4, 84, 84)	(32, 20, 20)
Conv Layer 2	Conv2D	(32, 20, 20)	(64, 9, 9)
Conv Layer 3	Conv2D	(64, 9, 9)	(64, 7, 7)
Flatten	Flatten	(64, 7, 7)	(7 *7* 64) = 3136
Value Stream Layer 1	NoisyLinear	3136	512
Value Stream Layer 2	NoisyLinear	512	n_atoms
Advantage Stream Layer 1	NoisyLinear	3136	512
Advantage Stream Layer 2	NoisyLinear	512	n_actions * n_atoms
Q-value Combination	-	-	(batch_size, n_actions, n_
Softmax	Softmax	(batch_size, n_actions, n_atoms)	(batch_size, n_actions, n_
Output	-	(batch_size, n_actions, n_atoms)	-

Figure 5: Rainbow DQN Architecture Components

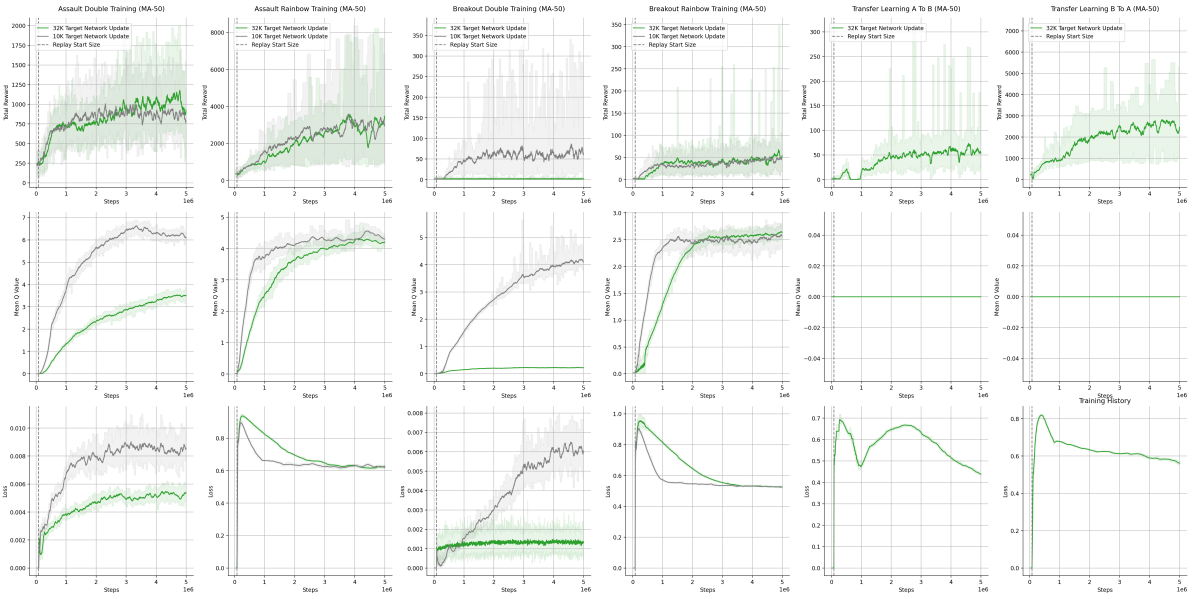

Figure 6: Full Training Curves