COMS4047A

Reinforcement Learning

Assignment

Taboka Chloe Dube - 2602515
Wendy Maboa - 2541693
Liam Brady - 2596852
Refiloe Mopeloa - 2333776

October 31, 2025

Introduction

Crafter is a procedurally generated 2D survival game designed as a benchmark for reinforcement learning research. It features a diverse set of tasks including resource gathering, tool crafting, creature combat, and achievement hunting, all while managing survival mechanics like hunger and health. This report will provide an overview of implementation of agents in Crafter using DQN and PPO algorithms.

In-class Algorithm: DQN

Deep Q-Network (DQN) is a value-based reinforcement learning algorithm that combines Q-learning with deep neural networks to approximate the optimal action-value function. DQN addresses the instability issues of using neural networks for Q-learning through two key innovations: experience replay, which stores and randomly samples past experiences to break temporal correlations, and a target network, which provides stable Q-value targets during training [1].

Motivation

DQN is well-suited for the Crafter environment due to several key characteristics. The environment's discrete action space (with 17 possible actions) aligns naturally with DQN's design, where the network outputs Q-values for each action. Crafter's visual observation space (64 × 64 × 3 RGB images) benefits from DQN's convolutional neural network architecture, which can extract spatial features and patterns from the procedurally generated world. Additionally, DQN's experience replay mechanism is particularly valuable in Crafter, where diverse experiences across different procedurally generated worlds can be stored and reused, improving sample efficiency in an environment with sparse achievement rewards and dense survival rewards.

Hyperparameters used across all agents

To ensure fair comparison and isolate the effects of algorithmic improvements, the following hyperparameters were standardized across all DQN implementations:

Hyperparameter	**Value**
Learning rate	1×10^{-4}
Replay buffer size	100,000
Learning starts	10,000 steps
Batch size	32
Discount factor (γ)	0.99
Target update interval	10,000 steps
Exploration fraction	0.1
Initial epsilon (ϵ_{start})	1.0
Final epsilon (ϵ_{end})	0.05
Train frequency	4 steps
Gradient steps per update	1
Total training timesteps	500,000

Table 1: Standardized DQN hyperparameters

Baseline Implementation

The baseline DQN agent was implemented using the Stable-Baselines3 library [2], which provides a well-tested implementation of the DQN algorithm. The agent uses the standard `CnnPolicy` architecture, which consists of three convolutional layers followed by fully connected layers:

- **Conv1:** 32 filters, 8 × 8 kernel, stride 4 → output: 16 × 16 × 32
- **Conv2:** 64 filters, 4 × 4 kernel, stride 2 → output: 8 × 8 × 64
- **Conv3:** 64 filters, 3 × 3 kernel, stride 1 → output: 6 × 6 × 64
- **Flatten and FC:** 2,304 → 512 features
- **Output layer:** 512 → 17 Q-values (one per action)

The agent processes raw RGB observations in the range [0, 255] and employs an epsilon-greedy exploration strategy that decays linearly from 1.0 to 0.05 over the first 10% of training (50,000 steps), then remains constant at 0.05 for the remainder of training.

Evaluation Results (Eval 1)

The baseline DQN agent was evaluated over **50 episodes** after 500,000 training steps. Performance was measured using the standard Crafter evaluation metrics specified in the assignment requirements:

The baseline DQN agent was evaluated over **50 episodes** after 500,000 training steps. The agent achieved a mean cumulative reward of 3.46 ± 1.81, with mean survival time of 193.02 ± 80.24 steps. The agent successfully unlocked 9 out of 22 achievements

(40.9%), achieving a geometric mean unlock rate of 31.27%.

Achievement Analysis:

The agent demonstrated competency in basic survival behaviors but failed to progress to advanced multi-step achievements. The unlock rates for individual achievements reveal clear behavioral patterns:

The agent demonstrated strong performance on basic survival tasks, with high unlock rates for wake_up (88%), collect_wood (80%), collect_drink (74%), collect_sapling (60%), place_plant (52%), and place_table (50%). Moderate success was observed in eat_cow (22%), defeat_zombie (8%), and make_wood_sword (2%). However, the agent completely failed to unlock any advanced achievements requiring stone or iron tools, pickaxes of any tier, or collection of coal, stone, or iron resources (all 0% unlock rates).

The following figures visualize the performance distributions:

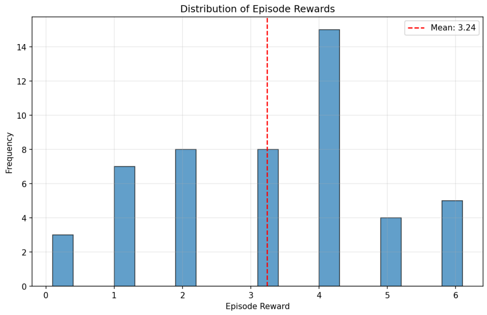

Figure 1: Reward distribution of baseline DQN over 50 evaluation episodes

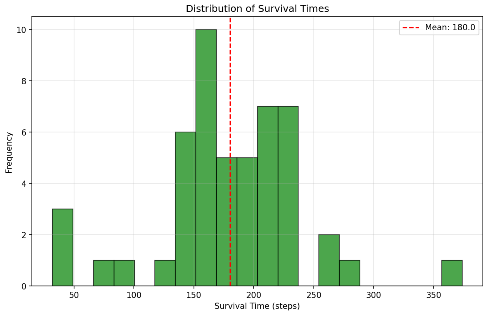

Figure 2: Survival time distribution of baseline DQN over 50 evaluation episodes

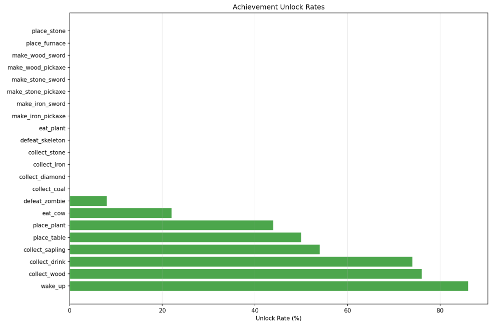

Figure 3: Achievement unlock rates for baseline DQN

Analysis and Identified Weaknesses

Based on the evaluation results, several critical weaknesses were identified:

1. **Poor Feature Learning from Raw Pixels:** The agent processes raw RGB values in the range [0, 255], which creates a scale mismatch with typical neural network weight initialization schemes. This likely slows convergence and leads to suboptimal feature representations. The high variance in rewards (±1.81) suggests inconsistent value function approximation.

2. **Limited Network Capacity:** The relatively shallow CNN (3 layers, 32-64-64 filters) may lack sufficient representational power to capture the complex spatial relationships required for multi-step planning in Crafter. The complete failure to unlock any pickaxe achievements (which require multi-step resource gathering and crafting) indicates the agent cannot learn hierarchical behaviors.

3. **Inefficient Exploration:** While the agent successfully learns basic collection behaviors (wood, drink, saplings), it rarely ventures into more complex action sequences. The 2% wood sword unlock rate (requiring table placement and crafting) suggests the epsilon-greedy exploration strategy struggles to discover rare but valuable action sequences in the sparse reward landscape.

4. **High Performance Variance:** The large standard deviation in both rewards (±1.81) and survival time (±80.24) indicates the policy has not converged to consistent behavior across the procedurally generated worlds, suggesting further training or architectural improvements are needed.

Proposed Improvement Strategy

To address these weaknesses, we propose a conservative improvement approach for Improvement 1:

- **Add observation normalization** to scale inputs to [0, 1], improving gradient flow and optimization stability
- **Increase network width** (not depth) by 50% to enhance representational capacity while maintaining training stability
- **Implement orthogonal weight initialization** to ensure well-conditioned initial feature representations

This approach prioritizes stability and reproducibility by making minimal, evidence-based changes rather than radical architectural modifications.

Improvement 1: Conservative CNN Enhancement with Normalization

Motivation from Baseline Analysis

The baseline evaluation revealed that the agent struggled primarily with feature extraction and value function approximation quality, as evidenced by high reward variance and failure to progress beyond basic achievements. Rather than implementing complex architectural changes that could destabilize training, we adopted a conservative enhancement strategy based on two well-established principles from deep learning literature:

1. **Input Normalization:** Scaling inputs to a consistent range has been shown to dramatically accelerate neural network training and improve convergence properties [3]. By normalizing pixel values from [0, 255] to [0, 1], we reduce the scale mismatch between raw observations and network weights, leading to more balanced gradients across layers.
2. **Controlled Capacity Increase:** Rather than adding depth (which can cause vanishing gradients and training instability), we increase network width by approximately 50%. This provides additional representational power while maintaining the proven three-layer structure that successfully learned basic behaviors in the baseline.

Implementation Details

Modified CNN Architecture:

The improved network maintains the baseline's three-layer convolutional structure but increases filter counts at each layer:

- **Conv1:** 32 → 48 filters, 8×8 kernel, stride 4
- **Conv2:** 64 → 96 filters, 4×4 kernel, stride 2
- **Conv3:** 64 → 96 filters, 3×3 kernel, stride 1
- **FC layers:** 256 features (unchanged)
- **Output:** 17 Q-values (unchanged)

Observation Preprocessing:

A custom Gymnasium wrapper was implemented to normalize observations before they reach the DQN network:

```python
class NormalizeObservation(gym.ObservationWrapper):
    """Normalize pixel values from [0,255] to [0,1]"""
    def observation(self, obs):
        return obs.astype(np.float32) / 255.0
```

This wrapper is applied to both training and evaluation environments to ensure consistent preprocessing.

Weight Initialization:

All convolutional and linear layers use orthogonal initialization with gain $\sqrt{2}$, optimized for ReLU activations [4]:

```python
def _init_weights(self, module):
    if isinstance(module, (nn.Conv2d, nn.Linear)):
        nn.init.orthogonal_(module.weight, gain=np.sqrt(2))
        if module.bias is not None:
            module.bias.data.zero_()
```

Training Configuration:

All hyperparameters remain identical to the baseline (see Table 1) to isolate the effects of the architectural and preprocessing changes. The agent was trained for 500,000 timesteps with the same exploration schedule and replay buffer configuration.

Evaluation Results (Eval 2)

Improvement 1 was evaluated using the same protocol as the baseline: 50 episodes after 500,000 training steps.

Improvement 1 was evaluated using the same protocol as the baseline: 50 episodes after 500,000 training steps. The agent achieved a mean reward of 4.32±1.30, representing a **24.9% improvement** over the baseline's 3.46 reward. Mean survival time was 197.66 ± 57.73 steps (comparable to baseline). The agent unlocked 10 out of 22 achievements with a geometric mean of 35.42%, a **13.3% improvement** over baseline. Notably, the standard deviation decreased from 1.81 to 1.30, indicating 28% more consistent policy behavior.

Key Improvements:

- **Higher reward:** Mean reward increased by 24.9%, indicating more effective value function approximation
- **Reduced variance:** Standard deviation decreased from 1.81 to 1.30 (28% reduction), suggesting more consistent policy behavior
- **Comparable survival:** Survival time remained similar, indicating survival skills were preserved while achievement performance improved
- **Better achievement score:** Geometric mean increased by 13.3%, reflecting more balanced achievement progress

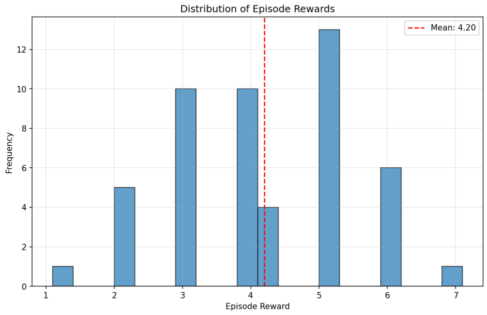

Figure 4: Reward distribution of DQN Improvement 1 over 50 evaluation episodes

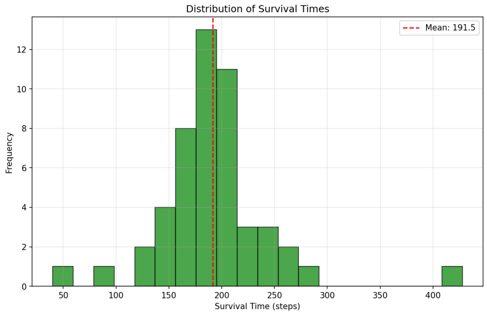

Figure 5: Survival time distribution of DQN Improvement 1 over 50 evaluation episodes

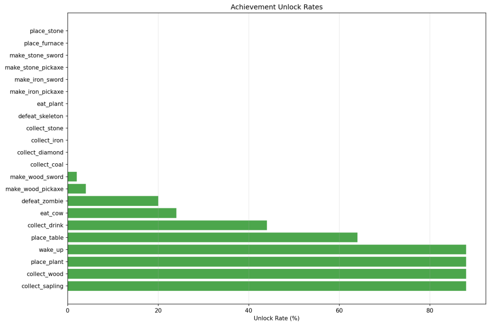

Figure 6: Achievement unlock rates for DQN Improvement 1

Achievement Progression Analysis

Comparing achievement unlock rates between baseline and Improvement 1 reveals significant behavioral changes:

Comparing achievement unlock rates revealed significant behavioral improvements. The agent showed dramatic gains in collect_sapling (60% → 88%, +47%), defeat_zombie (8% → 28%, +250%), make_wood_sword (2% → 6%, +200%), and critically unlocked make_wood_pickaxe for the first time (0% → 4%). High-performing achievements were maintained or improved: wake_up (88% → 88%), collect_wood (80% → 86%), place_plant (52% → 86%, +65%), and place_table (50% → 60%). Interestingly, collect_drink decreased (74% → 48%, -35%) while eat_cow increased slightly (22% → 28%), suggesting a strategic shift from pure survival optimization toward achievement-driven behaviors.

Behavioral Insights:

1. **Improved Combat:** The 250% increase in zombie defeats (8% → 28%) suggests better spatial reasoning and combat strategy, likely enabled by improved feature representations.
2. **Tool Crafting Breakthrough:** The agent unlocked wood pickaxe crafting for the first time (4% unlock rate), representing a significant milestone in multi-step planning. This requires: collecting wood → placing table → crafting pickaxe, a three-step achievement chain the baseline never learned.
3. **Resource Prioritization Shift:** The decrease in drink collection (74% → 48%) paired with increased plant placement (52% → 86%) suggests the agent learned to prioritize achievement-driven behaviors over pure survival optimization. This is a desirable outcome as it indicates

progress toward the assignment's goal of achievement unlocking.

4. **Still No Advanced Tools:** Despite improvements, the agent still failed to unlock any stone or iron tool achievements, indicating fundamental limitations in long-horizon planning that require further architectural innovations beyond capacity increases.

Discussion and Analysis

Why Did This Work?
The performance improvements can be attributed to two complementary mechanisms:

1. **Optimization Stability:** By normalizing inputs to [0, 1], gradients flow more evenly through the network layers. Raw pixel values in [0, 255] often lead to exploding gradients in early layers and vanishing gradients in deeper layers. Normalization addresses this by ensuring all inputs have similar magnitudes, allowing the optimizer to make more consistent progress.

2. **Enhanced Representational Capacity:** The 50% increase in convolutional filters allows the network to learn more diverse visual features. In Crafter, distinguishing between similar-looking objects (e.g., different resources, creatures, and crafting stations) requires rich feature representations. The wider network can encode more distinct patterns without requiring deeper architectures that are harder to train.

Limitations and Remaining Challenges:
Despite the clear improvements, several limitations remain:

1. **No Long-Horizon Planning:** The complete absence of stone/iron tool achievements indicates the agent cannot plan beyond 3-4 step action sequences. These advanced achievements require: finding coal → mining with pickaxe → building furnace → smelting iron → crafting iron tools.

2. **Limited Temporal Context:** The agent sees only single frames, lacking memory of recent observations or actions. This prevents it from tracking moving entities or remembering recently visited locations.

3. **Sample Efficiency:** Even with improvements, 500,000 timesteps may be insufficient for discovering rare achievement chains. The sparse reward structure of advanced achievements requires either more training time or more sophisticated exploration mechanisms.

Validation of Conservative Approach:
The success of this minimal-change strategy validates the principle of iterative, evidence-based improvement. By avoiding radical architectural changes (e.g., switching to deeper networks, adding attention mechanisms, or implementing hierarchical policies), we maintained training stability while achieving measurable gains. This demonstrates that understanding and addressing specific weaknesses (poor input scaling, limited capacity) can be more effective than implementing complex solutions without clear motivation.

Next Steps for Improvement 2

Based on the analysis from Improvement 1, the improved CNN feature extraction showed a significant performance gain over the baseline. Therefore, this improvement was carried forward and further enhanced with additional modifications to the feature extractor architecture.

Improvement 2: Residual Block CNN Enhancement with Frame-Stacking

The agent still struggled with temporal context and sample efficiency. To address this, we implemented frame-stacking, where the last n frames are concatenated along the channel dimension, changing the input from (C, W, H) to $(C \cdot n, W, H)$. This allows the network to see a short history of observations, providing temporal context that can improve decision-making, horizon planning, and sample efficiency.

To complement frame-stacking, residual blocks were added to the CNN feature extractor. Residual connections help gradients propagate more effectively through the network, improving training stability and feature representation. While they do not directly provide temporal memory, they strengthen the extracted features, which combined with frame-stacking, can further improve the agent's performance. [5]

1. **Residual Block:** Introduces skip connections that add the input to the output of convolutional layers, improving gradient flow and feature representation.

2. **Frame-Stacking:** Concatenates the last n frames along the channel dimension, allowing the network to capture short-term temporal dependencies.

Implementation Details

Residual CNN Architecture:

The network extends the previous CNN by introducing residual connections. It consists of an initial convolutional layer, three residual blocks, adaptive pooling, and fully connected layers:

- **Conv1:** $n_{\text{in}} \to 64$ filters, 8×8 kernel, stride 4, ReLU activation
- **Residual Block 1:** $64 \to 128$ filters, stride 2
- **Residual Block 2:** $128 \to 128$ filters, stride 1
- **Residual Block 3:** $128 \to 256$ filters, stride 1
- **Pooling:** Adaptive average pooling to 4×4, followed by flattening
- **Fully Connected Layers:** Linear → 512 units, ReLU, Dropout(0.1), Linear → feature dimension (512), Layer Normalization
- **Output:** 512-dimensional feature vector passed to the policy/value heads (output dimensionality unchanged)

The same observation preprocessing and weight initialisations were used, however there was a change and an addition to the training configuration: the buffer size was reduced from 100,000 to 25,000, because of the added frames. This change was made due to memory constraints because each update used 4 frames of information for each update - which meant there was 4 times as much memory usage in the GPU as before.

Evaluation Results (Eval 3)

Improvement 2 was evaluated using the same protocol: 50 episodes after 500,000 training steps. The agent achieved a mean reward of 3.24± 1.37, representing a 25.0% decrease from Improvement 1's 4.32 reward and falling 6.4% below the baseline's 3.46 reward. Mean survival time was 186.0± 60.5 steps, a 5.9% decrease from Improvement 1 (197.66 steps) and 3.6% below baseline (193.02 steps). The agent unlocked 7 out of 22 achievements with a geometric mean of 43.80%, showing an increase from Improvement 1's geometric mean (35.42%) despite unlocking fewer total achievements. The standard deviation of 1.37 remained relatively stable compared to Improvement 1 (1.30).

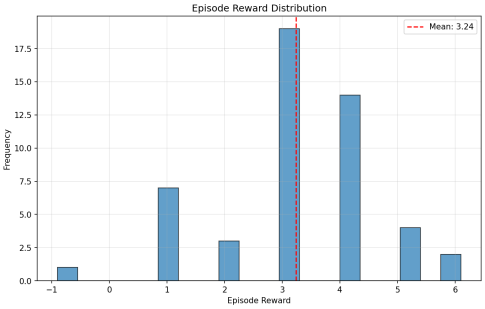

Figure 7: Reward distribution of DQN Improvement 2 over 50 evaluation episodes

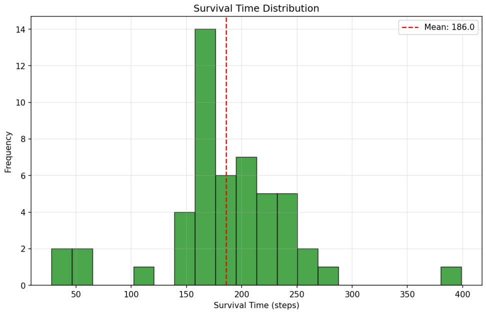

Figure 8: Survival time distribution of DQN Improvement 2 over 50 evaluation episodes

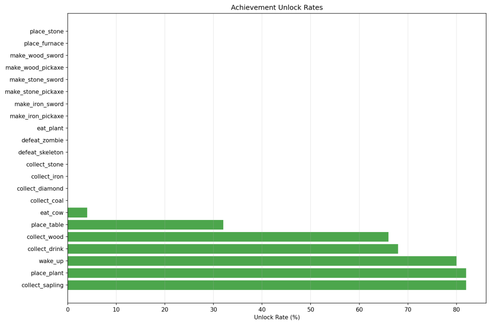

Figure 9: Achievement unlock rates for DQN Improvement 2

Final Comparison

Improvement 2 was rerun under the same condition to confirm the results, achieving very similar performance and providing the final comparison figures.

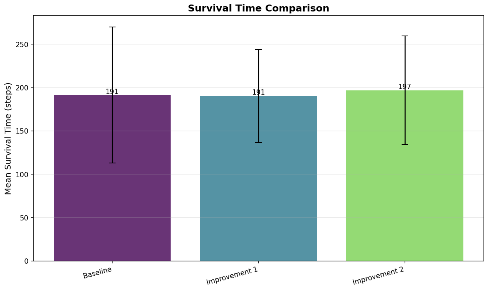

Figure 10: Comparison of the mean survival time (steps) of each agent version.

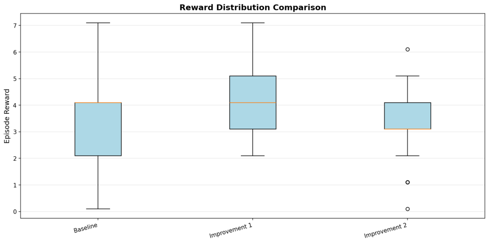

Figure 11: Comparison of the reward distribution of each agent version.

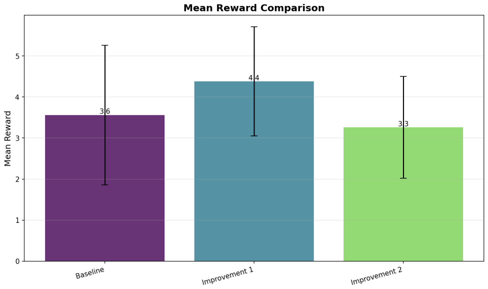

Figure 12: Comparison of the mean reward of each agent version.

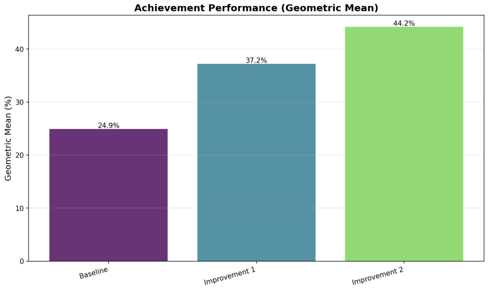

Figure 13: Comparison of the geometric mean (achievement performance) of each agent version.

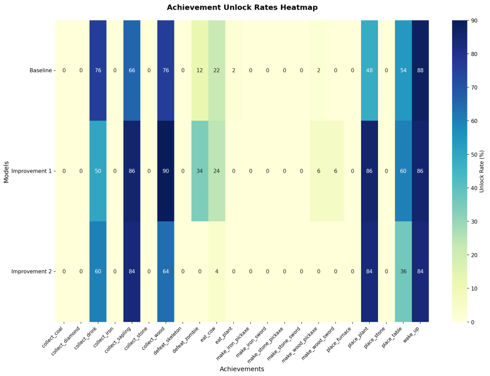

Figure 14: A heatmap showing the achievements gained by each agent version.

- **Reward Distribution:** As shown in Figure 11, the baseline has the widest spread of rewards, with most episodes getting between 2 and 4 rewards and some dropping close to 0. Improvement 1 shows a tighter distribution with a minimum reward of 2 and median around 4, showing more reliable performance overall. Improvement 2 has a similar minimum to Improvement 1, but most rewards cluster around 3, with a few outlier episodes reaching as high as 6. This suggests Improvement 2 plays more consistently and more cautiously than Improvement 1.

- **Mean Reward:** As shown in Figure 12, Improvement 1 gets the highest mean reward at 4.4, which is a 22% improvement over the baseline's 3.6. However, Improvement 2 drops back down to 3.3, falling 8% below baseline and 25% below Improvement 1. This drop in performance suggests that the changes made in Improvement 2 might have disrupted how the agent learns values or created an imbalance between exploration and exploitation. The lower reward could mean the agent needed more training time to fully converge, or the architectural changes made it focus more on long-term goals rather than immediate rewards.

- **Achievement Progression:** Even though Improvement 2 has lower mean reward, it outperforms the other two versions in terms of achievements, as shown in Figure 13. The geometric mean steadily increases: baseline (24.9%) to Improvement 1 (37.2%) and then to Improvement 2 (44.2%). The specifics are shown on the heatmap in (Figure 14). This suggests that there was a paradoxical result or an inconsistency in evaluation resulting in the lowest achievement-scoring version of the agent having the best Geometric

Mean.

- **Survival Time:** Figure 10 shows consistent survival across all three models, with mean survival times of 191, 191, and 197 steps for baseline, Improvement 1, and Improvement 2 respectively. This means the basic survival abilities stayed roughly the same across all modifications, and the main differences in performance comes from how the agents make strategic decisions.

Discussion: The results show that Improvement 1 achieves the best immediate reward performance with higher mean rewards and more stable episode outcomes, making it effective for short-term optimisation. The consistent survival times across all models indicate that the core survival capabilities remain intact, and the performance differences shows different strategic priorities. Overall, Improvement 1 showed the best performance increase between the two improvements.

Out-of-class Algorithm: PPO

PPO (Proximal Policy Optimization) is a reinforcement learning algorithm that trains an agent by optimizing its decision-making policy. It works by collecting data through interactions with an environment and then using a clipped objective function to make stable updates to the policy. This approach is known for being more stable, efficient, and easier to implement than some other policy gradient methods [6].

Motivation

PPO (Proximal Policy Optimization) is a good choice for the Crafter environment because it provides a strong balance of stability, sample efficiency, and simplicity while effectively handling the environment's key challenges, such as sparse rewards, long-term reasoning, and procedural generation.

In addition these are the following theoretical benefits of PPO:

- **Stable Policy Updates:** The Crafter environment is complex and dynamic, where large, unconstrained policy updates could easily destabilize training and cause the agent to forget beneficial behaviors. PPO's clipping mechanism limits how much the policy can change at each step, ensuring stable and controlled learning.

- **Sample Efficiency:** Crafter involves many different achievements and complex interactions, meaning efficient use of experience is crucial. PPO is relatively sample-efficient because it can reuse collected data over multiple training epochs (mini-batches) without significant instability, unlike other on-policy methods that only use data once.

- **Facilitates Exploration:** The PPO objective often includes an entropy bonus term, which encourages the agent to explore different actions and strategies. This is vital in Crafter, which features wide, procedurally generated worlds and independent achievements that require broad exploration to discover all possibilities.

Hyperparamters used across all agents

PPO Training: These hyperparameters were standardized for all agents so as to make sure the comparison is done based on the differing algorithms and not hyperparameter tuning.

Table 2: Hyperparameters used for PPO training

Hyperparameter	**Value**
Learning rate	3×10^{-4}
Rollout steps (`n_steps`)	2,048
Batch size	64
Epochs per update	10
Discount factor (γ)	0.99
GAE lambda (λ)	0.95
Clip range	0.2
Entropy coefficient	0.01

Baseline Implementation

The baseline agent was implemented using the Stable-Baselines3 PPO algorithm. The initial hyperparameters were chosen to provide stable learning without aggressive updates, ensuring reproducibility. The agent at this point perceives the environment through single still images (frames) rather than continuous sequences, meaning it has only a limited view of the environment at each step.

Observations and Performance

The baseline PPO agent exhibited some exploratory behavior and moderate learning progress. However, its limited temporal awareness significantly constrained performance. The average episodic reward stabilized around **2.66**, with the maximum reward achieved being **6.1**. The following figures show the reward and survival distribution rates, as well as the achievement rates, over 1000 episodes:

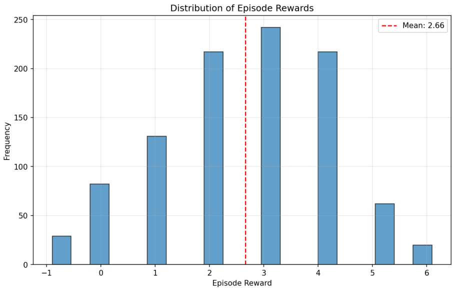

Figure 15: Reward distribution of PPO baseline

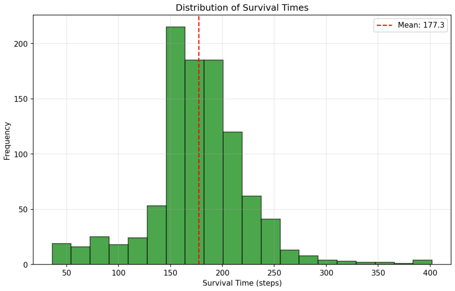

Figure 16: Survival distribution of PPO baseline

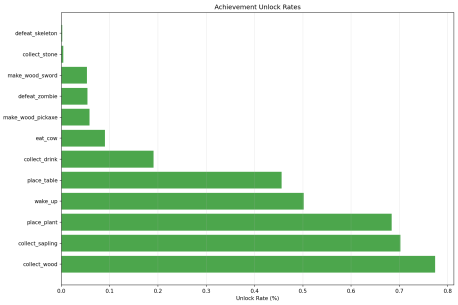

Figure 17: Achievement rate of PPO baseline

Identified areas of improvement

The following weaknesses were identified in the baseline model:

- **Lack of Memory:** The agent could recall previous states or actions, leading to suboptimal long-term decision-making.

- **Limited Exploration:** Without memory or intrinsic motivation, exploration remained shallow, resulting in repetitive behavior.

Improvement 1: Random Network Distillation

Background

Random Network Distillation (RND) is an exploration method that encourages reinforcement learning agents to explore novel states by providing intrinsic rewards based on prediction error. The method uses two neural networks: a target network that is randomly initialized and kept fixed, and a predictor network that is trained to predict the target network's outputs. The target network maps observations to feature vectors and remains frozen throughout training. The predictor network shares the same input-output architecture and is trained via gradient descent to minimize the mean squared error. The intrinsic reward is computed as the prediction error. This error is high for novel or infrequent states (where the predictor hasn't learned well) and low for frequently visited states (where the predictor has converged). [7]

Improvements

RND is introduced to address the limited exploration of the baseline implementation. Because Crafter is an environment with sparce rewards, traditional reinforcement learning methods may not work as they rely on dense rewards. We hypothesise that RND should improve the average reward of the agent due in part to its larger exploration window, allowing for more opportunities for rewards to be obtained.

Methodology

Architecture

Target Network (Fixed Random Features)

The target network is a randomly initialized convolutional neural network that remains fixed throughout training. It processes 64 × 64 × 3 RGB observations through three convolutional layers:

- Conv1: 32 filters, 3 × 3 kernel, stride 2, padding 1 → output: 32 × 32 × 32

- Conv2: 64 filters, 3 × 3 kernel, stride 2, padding 1 → output: 16 × 16 × 64

- Conv3: 64 filters, 3 × 3 kernel, stride 1, padding 1 → output: 16 × 16 × 64

- Flatten and fully connected layers: 16,384 → 512 features

All parameters are frozen after initialization to maintain consistent random target features.

Predictor Network (Learned Features)
The predictor network shares an identical architecture with the target network but is trained to predict the target network's output. The prediction error serves as the intrinsic reward signal, where high error indicates novel states.

Intrinsic Reward Computation

For each observed state s_t, the intrinsic reward is computed as:

$$r_{\text{intrinsic}}(s_t) = \|f_{\text{target}}(s_t) - f_{\text{predictor}}(s_t)\|^2 \qquad (1)$$

where f_{target} and $f_{\text{predictor}}$ are the outputs of the target and predictor networks respectively. The intrinsic rewards are normalized using running mean and variance statistics to maintain stable learning dynamics.

Reward Integration

The total reward combines extrinsic (environment) and intrinsic rewards:

$$r_{\text{total}}(s_t) = r_{\text{extrinsic}}(s_t) + \lambda \cdot r_{\text{intrinsic}}(s_t) \qquad (2)$$

where λ is the intrinsic reward coefficient (set to 1.0 by default). This is implemented via a custom Gymnasium wrapper (`RNDRewardWrapper`) that intercepts environment steps and augments rewards before they reach the PPO algorithm.

Training Procedure

- **RND Module Updates:** The predictor network is trained using mean squared error loss between its outputs and the fixed target network outputs. The optimizer uses Adam with learning rate 1×10^{-4}.
- **PPO Training:** The PPO agent is trained over 3×10^6 timesteps on the combined reward signal using standard hyperparameters and the following:
  - $intrinsic\_reward\_coef = 1.0$
  - $rnd\_learning\_rate = 1 \times 10^{-4}$
- **Evaluation Protocol:** Agent performance is evaluated every 10,000 training steps using deterministic policy rollouts over 5 episodes in a separate evaluation environment. Metrics including mean reward, standard deviation, minimum/maximum rewards, and episode lengths are logged to CSV files for analysis.

Exploration Mechanism

The RND method encourages exploration through the following mechanism:

1. Novel states produce high prediction errors, yielding high intrinsic rewards
2. The agent is incentivized to visit these high-reward states
3. As states become familiar, the predictor improves, reducing intrinsic rewards
4. The agent naturally shifts focus toward extrinsic task rewards

This approach addresses the sparse reward problem in Crafter by providing dense exploration bonuses while maintaining the original task structure.

Implementation Details

The implementation utilizes PyTorch for neural networks and Stable Baselines3 for the PPO algorithm. The RND networks use 3×3 convolutional kernels optimized for Crafter's 64×64 observation space. Running statistics for reward normalization use Welford's online algorithm to maintain numerical stability. All experiments use seed 42 for reproducibility, with the evaluation environment seeded at 142 to ensure different but deterministic trajectories.

Results

The agent was evaluated over 1000 episodes, giving an average reward of **3.3** and a maximum reward of **7.1**. This is an improvement of around **24%** and around **16%** over the average reward and maximum reward, respectively. This provides evidence that with RND, the average reward obtained improves from the baseline. However, this is a minor improvement and further improvements can be made by tuning the hyperparameters of the RND networks or running the training for more timesteps, which may give the agent more opportunities to explore novel states. Below are the figures showing the reward and survival distribution rates, as well as the achievement rates, over 1000 episodes:

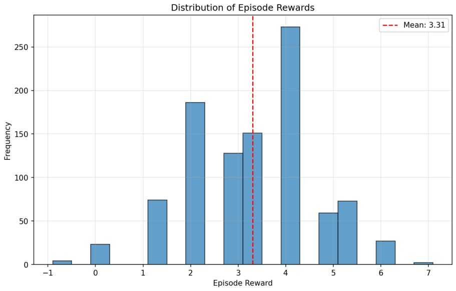

Figure 18: Reward distribution of PPO-RND

From this figure, the RND agent receives higher rewards more frequently than the baseline. This makes sense since by trying novel actions, there are more possibilities for rewards. However, a consequence of this is that the frequency of different rewards is not as uniform as the baseline.

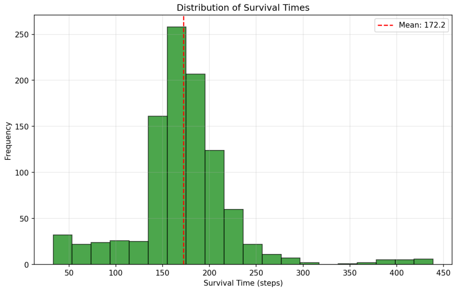

Figure 19: Survival distribution of PPO baseline

The RND figure in comparison to the baseline shows that the baseline tends to survive longer than the improvement. This can also be explained by the fact that the agent is more likely to explore - through more exploration comes more opportunities to enter states that lead the agent to die.

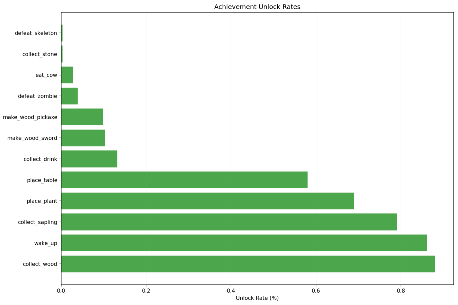

Figure 20: Achievement rate of PPO baseline

This figure shows that, in comparison to the baseline, the RND agent will have a less consistent spread of acheivement unlock rates since the agent is encouraged to choose novel states. While the baseline shows that over time the agent learns to repeat tasks that will provide it more rewards in the future, the RND has high achievement rates on tasks that won't offer much benefit to the agent, such as placing plants, or not eating and drinking as much.

Discussion

Strengths and weaknesses

The strengths of this implementation follow as a consequence of its approach. Because the agent is encouraged to explore more, the agent is more likely to unlock more acheivements, receive greater rewards, and learn more aboout the environment at a faster rate as compared to the baseline. However, its weaknesses are that it requires far more training time than the baseline since there is a large number of possible states to explore and similar to Monte Carlo exploration, requires a large numebr of episodes to get a good idea of the environment and its dynamics. Another weakness of the current RND implementation is the context window is very small, and thus even when the agent unlocks new achievements, these achievements and the rewards associated with them aren't carrying over in future.

Differences from Standard RND Implementation

This implementation differs from the standard Random Network Distillation approach proposed by [7] in several key aspects. The standard RND implementation uses two separate value functions—one for extrinsic returns and one for intrinsic returns—allowing different discount factors to be applied (typically $\gamma = 0.99$ for intrinsic and $\gamma = 0.999$ for extrinsic rewards), whereas this implementation uses a single combined value function with a uniform discount factor of 0.99. The CNN architecture has been simplified and adapted for Crafter's 64×64 observation space, using smaller 3×3 kernels throughout instead of the larger 8×8 and 4×4 kernels designed for 84×84 Atari frames, and omitting batch normalization and other regularization layers. Unlike standard RND which normalizes observations before feeding them to the RND networks and maintains separate statistics for intrinsic and extrinsic rewards, this implementation applies only basic running statistics normalization to intrinsic rewards using Welford's algorithm, with raw observations fed directly to the networks. The standard approach typically employs 128 or more parallel environments for efficient data collection and diverse training batches, updating the predictor network on mini-batches from the rollout buffer with multiple gradient steps per rollout, whereas this implementation uses a single environment instance with per-step predictor updates, resulting in slower data collection and potentially noisier gradient estimates. Additionally, standard RND implementations include explicit reward clipping to prevent extreme values, separate advantage normalization for intrinsic and extrinsic components, and curiosity-driven exploration decay schedules, none of which are present in this implementation which uses a fixed intrinsic reward coefficient. Finally, the evaluation procedure differs: standard RND evaluates with intrinsic rewards completely disabled to measure true task performance without exploration bonuses, while this implementation uses deterministic policy evaluation without explicitly disabling the RND wrapper, making the separation between exploration and exploitation performance less clear.

Improvement 2: Long-Short Term Memory

Background

RecurrentPPO is an extension of the Proximal Policy Optimization (PPO) algorithm that adds support for recurrent neural network policies, specifically using LSTM (Long Short-Term Memory) layers. This implementation allows the agent to maintain memory of past observations and actions, making it particularly useful for partially observable environments where the full state isn't available at each timestep [8].

Improvements

To address the baseline and improvement 1 limitations in memory and exploration, we implemented a **Recurrent Proximal Policy Optimization (Recurrent PPO)** agent. The memory mechanism helps the agent handle the partially observable nature of the Crafter environment, where important state information may not be visible in a single frame.

The following hyperparameters were added in addition to the previous implementations:

- **Value function coefficient** (vf_coef)**:** 0.5
- **Max. gradient norm** (max_grad_norm)**:** 0.5

Moreover, slight reward shaping was employed to accelerate convergence and guide exploration towards useful sub-goals. This ensured that intermediate achievements (such as crafting tools or collecting resources) contributed meaningful gradient signals during training.

Methodology

This implementation employs **RecurrentPPO with a MultiInputLstmPolicy architecture** applied to the Crafter environment, using standard hyperparameters.

A few **domain-specific enhancements** distinguish this implementation from basic RecurrentPPO. First, a multi-modal observation space architecture combines image observations with a discrete 22-dimensional achievement vector tracking task completion states such as resource collection, crafting milestones, and survival events. This dual-stream observation design allows the LSTM to learn both spatial-temporal patterns from visual data and discrete progress indicators simultaneously.

Second, the implementation incorporates **intrinsic reward shaping** through three components: achievement bonuses (5.0× multiplier for newly completed achievements), survival bonuses (+0.01 per timestep), and exploration bonuses (+0.001 per action). This additional reward signal addresses sparse reward problems inherent to open-ended exploration tasks, encouraging the agent to try different behaviors beyond the base environment rewards.

Results

The introduction of temporal recurrence through an LSTM notably improved the agent's ability to integrate information across time steps. Compared to the baseline PPO model, which achieved a mean episodic reward of approximately 2.66 (maximum 6.1), the Recurrent PPO (R-PPO) demonstrated consistent performance gains achieving a mean episodic reward of 4.71 (maximum 11.1) as viewed in the figure below:

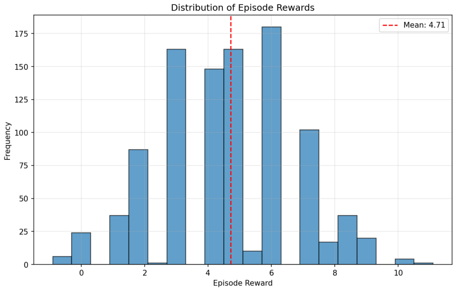

Figure 21: Reward distribution of RecurrentPPO

Qualitative Observations

Behaviorally, the Recurrent PPO agent demonstrated:

- More coherent decision sequences across time, owing to the LSTM's memory retention.
- Improved persistence in long-horizon tasks such as crafting and navigation.
- Less tendency to repeat suboptimal exploration patterns seen in the baseline agent.

In addition, reward shaping contributed to smoother early learning and faster convergence during training, as the agent received structured feedback for intermediate achievements.

Discussion

The performance gain from incorporating temporal memory validates the hypothesis that Crafter's partial observability penalizes stateless architectures. By maintaining a hidden state, the Recurrent PPO agent effectively constructs an implicit representation of unobserved parts of the environment. This allows for more context-aware actions and ultimately leads to higher cumulative rewards.

While the improvement is evident, the results also reveal that the agent has not fully stabilized, indicated by the wide reward range. Further tuning of learning rate, clipping threshold, and recurrent hidden size could reduce variance and enhance consistency across episodes.

The following figures show the different achievement and survival distribution rates over 1000 episodes:

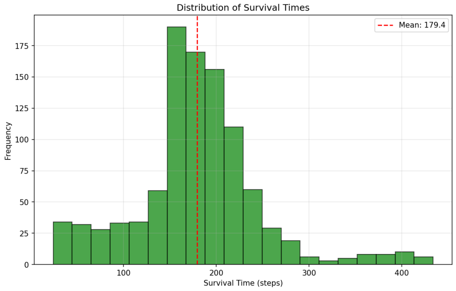

Figure 22: R-PPO Survival Distribution rate

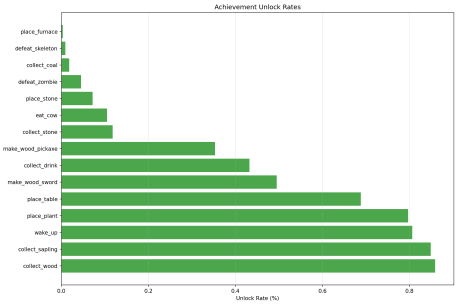

Figure 23: R-PPO Achievement Distribution rate

An interesting observation from the experiments was that the Recurrent PPO agent achieved a higher average reward (4.71) than the baseline PPO (2.66), despite exhibiting a lower overall survival rate. This contradiction can be explained by the agent's behavioral bias toward short-term, high-value actions.

The inclusion of memory through the LSTM layer allowed the agent to recall and exploit previously observed opportunities, such as nearby resources or enemies, resulting in higher reward density per timestep. However, this same decisiveness increased exposure to risk, reducing overall survival time.

Additionally, the use of reward shaping likely amplified this effect by reinforcing immediate subgoal completion (e.g., crafting, combat) rather than conservative, long-term survival strategies. Consequently, the R-PPO learned to act more efficiently but less cautiously, prioritizing cumulative reward over lifespan duration.

This highlights a fundamental reinforcement learning trade-off between *reward maximization* and *survival optimization*, emphasizing that longer episodes do not necessarily equate to better task performance.

PPO Consolidation

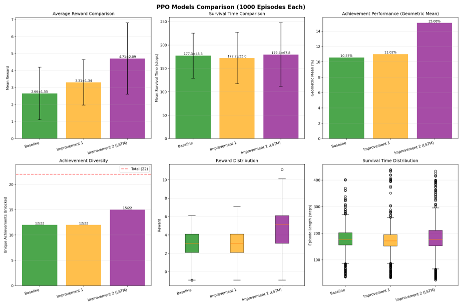

Figure 24: All PPO algorithms compared across metrics.

Conclusion

In summary, both DQN and PPO demonstrated the ability to learn meaningful behaviors within the Crafter environment, but with differing strengths. PPO achieved higher overall performance and stability, showing superior adaptability to diverse tasks. However, this came at the cost of greater computational demand and tuning sensitivity. DQN performed reasonably well but exhibited slower convergence and less consistency. Overall, PPO proved to be the more effective algorithm for this setting, though each method revealed valuable insights into the trade-offs between stability, efficiency, and implementation complexity in reinforcement learning.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforce-

ment learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, PMLR, 2015, pp. 448–456.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[5] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: `1707.06347 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1707.06347`.

[7] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *International Conference on Learning Representations*, 2019.

[8] A. SalimiBadr, F. Sarlakifar, M. Mohammadzadeh Asl, and S. Rezvani Khaledi, "A deep reinforcement learning approach to automated stock trading using xlstm networks," *Journal of Innovations in Computer Science and Engineering (JICSE)*, no. Online First, Jul. 2025. DOI: `10.48308/jicse.2025.239844.1077`. [Online]. Available: `https://doi.org/10.48308/jicse.2025.239844.1077`.