Reinforcement Learning Project Report: Implementing and Comparing PPO and DQN on Crafter

Anand Patel (2561034)
School of Computer Science
University of the Witwatersrand
anand.patel@students.wits.ac.za

Mikyle Singh (2465557)
School of Computer Science
University of the Witwatersrand
mikyle.singh@students.wits.ac.za

***Abstract*—This report documents our implementation of two reinforcement learning algorithms for the course project. We built PPO (Proximal Policy Optimization) and DQN (Deep Q-Network) agents to play Crafter, a 2D survival game. Starting from a baseline, we improved PPO from 5.08% to 8.61% through better hyperparameters, adding curiosity-driven exploration, and using a larger network. We tried 11 different approaches, with 7 failing to improve performance.**

For DQN, we progressed from 2.80% to 5.93% by stacking five targeted, non-trivial changes: n-step returns (n=3), inventory-aware action masking, a random-valid masking fallback, a conservative train-only curiosity bonus (ICM-lite), and NoisyNets with hard target updates. Each iteration was motivated by a measured failure mode (credit assignment, wasted actions, NOOP loops, early vs late exploration). We report standard Crafter metrics (geometric-mean score, per-achievement rates, survival, reward) and ablations for negative results. We attempted 13 cumulative approaches. Several failed and were analyzed to inform further design decisions.

This report explains what we implemented, what worked, what didn't, and what we learned about reinforcement learning in sparse-reward environments.

***Index Terms*—Reinforcement Learning, PPO, DQN, Crafter, Project Report**

I. Introduction

A. Project Goal

For this project, we implemented two reinforcement learning algorithms from scratch:

- Proximal Policy Optimization (PPO)
- Deep Q-Network (DQN)

We trained both agents on Crafter and systematically tested improvements to quantitatively understand what works for sparse-reward-based problems.

B. About Crafter

Crafter is a 2D survival game where the agent must gather resources, craft tools, and survive enemies. It's challenging because:

- The agent only sees 64×64 RGB images
- Rewards are very sparse (only 1% of actions get rewards)
- Many tasks need multiple steps (e.g., get wood $\rightarrow$ build table $\rightarrow$ craft pickaxe $\rightarrow$ mine coal)
- There are 22 different achievements to unlock

The Crafter Score is our main metric—it's the geometric mean of how often the agent unlocks each achievement [3]. Higher scores generally mean that the agent learned more skills.

C. The Two Algorithms

[External] PPO learns a policy that outputs probabilities for each action. It's stable because it prevents big policy changes.

[Course] DQN learns which actions are best by estimating their values. It's sample-efficient because it reuses past experiences.

D. Report Organization

Section II describes PPO implementation and results. Section III covers DQN. Section IV compares both approaches.

II. PPO Implementation and Results

A. How PPO Works

PPO learns a policy $\pi_\theta(a|s)$ that maps what the agent sees to action probabilities. The key idea is the clipped objective function:

$$L(\theta) = \min(r_t \cdot A_t, \text{clip}(r_t, 0.8, 1.2) \cdot A_t) \quad (1)$$

where r_t is how much the new policy differs from the old one, and A_t is the advantage (how good an action was compared to average).

The clipping keeps updates small and prevents the agent from changing too drastically, which makes training stable.

B. Our Implementation

Neural Network Architecture:

We used an actor-critic network with shared convolutional layers:

- 3 convolutional layers to process the 64×64×3 images
- Actor head: outputs probabilities for 17 actions
- Critic head: outputs a value estimate
- Tested 512 and 1024 hidden dimensions

Training Setup:

- Collected 2048 steps of gameplay
- Trained on this data for 10 epochs
- Used mini-batches of 64 samples
- Ran for 1 million total steps per experiment
- Each run took about 2.5 hours on M4 MacBook Pro

C. Evaluation 1: Baseline Results

Our Approach: We started with an initial configuration using common PPO values to establish baseline performance.

Settings:

- Learning rate: 0.001
- Entropy coefficient: 0.0001

Results:

- Crafter Score: **5.08%**
- The agent learned to gather wood (90%) and saplings (85%)
- It could build tables (75%)
- BUT: Never crafted tools, never found coal, died quickly (165 step episodes)

Why it struggled: The learning rate was too high (caused instability) and entropy was too low (agent didn't explore enough).

D. Evaluation 2: Fixing Hyperparameters

What we changed: Used standard values from published implementations (CleanRL, Stable-Baselines3 [10]):

- Learning rate: 0.0005 (2× slower)
- Entropy coefficient: 0.001 (10× higher)

Results:

- Crafter Score: **7.10%** (+39.8% improvement)
- Wood pickaxe: 51% (was 40%)
- Wood sword: 48% (was 35%)
- Started defeating zombies: 48% (was 30%)

Why it worked: The slower learning rate kept training stable. Higher entropy kept the agent trying new things instead of getting stuck repeating the same actions.

E. Evaluation 3: Adding Curiosity

The Problem: With sparse rewards, the agent rarely discovers new achievements by accident. It needs help exploring.

Our Solution: We added an Intrinsic Curiosity Module (ICM) that gives the agent bonus rewards for experiencing surprising situations.

How ICM Works:

1) A neural network tries to predict what will happen next
2) When the prediction is wrong (surprising situation), give reward
3) The agent learns to seek out these surprising situations
4) Eventually it finds rare achievements this way

The agent now gets two types of rewards:

- 80% from actual achievements (extrinsic)
- 20% from curiosity (intrinsic)

Results:

- Crafter Score: **8.27%** (+16.5% relative improvement)
- Found coal: 1% (was 0%)
- Defeated skeleton: 3% (was 0%)
- Better zombie combat: 54% (was 48%)
- Survived longer: 200 step episodes (was 182)

Why it worked: The curiosity bonus helped systematic exploration. The agent naturally explored early (high curiosity) then focused on achievements later (curiosity decreased).

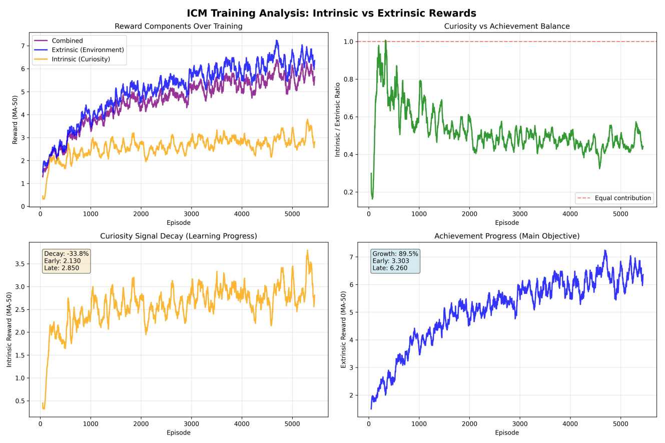

Fig. 1. ICM reward analysis for Evaluation 3. Top left: intrinsic (curiosity) vs extrinsic (achievement) rewards over training. Top right: curiosity/achievement ratio showing exploration-to-exploitation transition. Bottom: curiosity decay (agent learns) and achievement growth (agent succeeds). The 0.508 ratio shows good balance.

F. Evaluation 4: Bigger Network + Less Curiosity

What we tried: We combined two ideas:

1) Made the network bigger (512 → 1024 neurons)
2) Reduced curiosity slightly (20% → 15%)

Why we thought this would work:

- Bigger network can remember more complex strategies
- Less curiosity means more focus on actual goals

Results:

- Crafter Score: **8.61%** (+4.1% improvement)
- **Total improvement: 5.08% → 8.61% = +69.5%**
- Wood pickaxe: 58%, Wood sword: 61%
- Coal: 2% (doubled!)

Surprising finding: During training, the average reward was actually lower (5.14) than Evaluation 3 (6.27). But when we evaluated the final agent, it performed better. This taught us that training numbers don't always predict final performance.

G. Things That Didn't Work

We tried 7 other strategies that failed. Documenting failures is important for learning:

1) RND Curiosity (6.11%): We tried a simpler curiosity method called Random Network Distillation. We attempted this 3 times with different fixes, but all failed.

What went wrong: The curiosity rewards were way too high (42.28 vs ICM's 3.08). The agent got addicted to exploring and never focused on achievements—a "curiosity trap."

Lesson learned: Simpler isn't always better. RND's random features didn't capture what's actually important in Crafter.

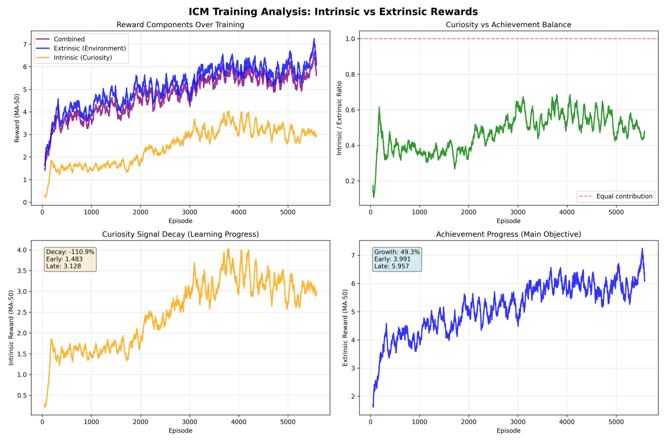

Fig. 2. ICM reward analysis for Evaluation 4 with lower curiosity (=0.15). The 0.483 ratio is slightly lower than Evaluation 3, showing more focus on achievements. Despite lower training rewards, this configuration achieved the best final score (8.61%).

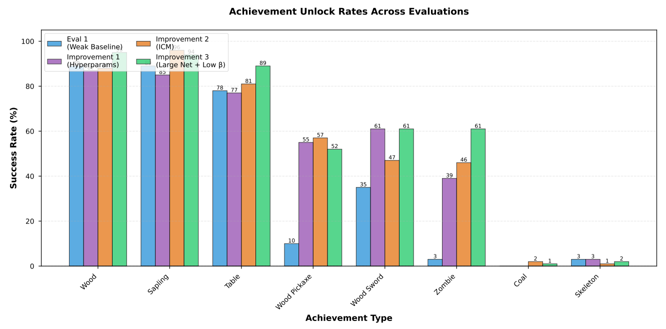

Fig. 3. Achievement unlock rates across PPO evaluations. Shows improvement in tool crafting and discovery of rare achievements like coal and skeleton combat.

2) *Too Much Curiosity (6.38%):* We tried increasing curiosity from 20% to 30%.

What went wrong: Another curiosity trap. The agent spent all its time exploring instead of achieving goals.

Lesson learned: Balance is critical. Too much curiosity is as bad as too little.

3) *Too Much Randomness (4.86% and 6.08%):* We tried higher entropy coefficients (0.005 and 0.002 instead of 0.001).

What went wrong: The agent's policy stayed too random. It never settled on good strategies.

Lesson learned: The agent needs to commit to strategies it learns. Too much randomness prevents this.

4) *Slower Learning (8.11%):* We tried a learning rate of 0.0001 (5× slower).

What went wrong: Got close to baseline (8.11% vs 8.27%) but learned too slowly for 1 million steps.

Lesson learned: Learning rate needs to match training time budget.

5) *Training Longer (7.52%):* We tried 1.5 million steps instead of 1 million (50% more training).

What went wrong: Performance got worse → Dropped from 8.27% to 7.52%.

Lesson learned: More training doesn't always help. PPO can degrade because it keeps learning from old experiences that are no longer relevant.

6) *Dual-Clip PPO (5.79%):* We tried a PPO variant that clips differently for positive vs negative advantages.

What went wrong: Made training unstable when combined with curiosity.

Lesson learned: Test modifications carefully. Dual-clip works in some situations but not ours.

7) *Just Bigger Network (8.36%):* We tried only increasing network size without changing curiosity.

What happened: This actually worked (8.36%) but not as well as combining both changes (8.61%).

Lesson learned: Combining improvements can work better than individual changes.

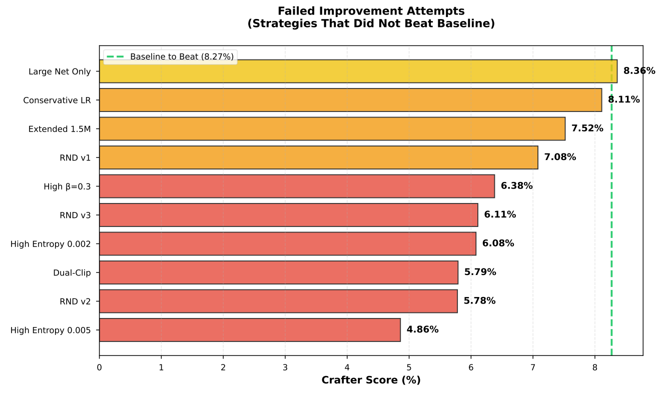

Fig. 4. The 7 strategies that didn't improve performance. RND curiosity, high entropy, and extended training all scored worse than our best ICM-based approach. This shows the importance of systematic experimentation.

H. Summary of PPO Results

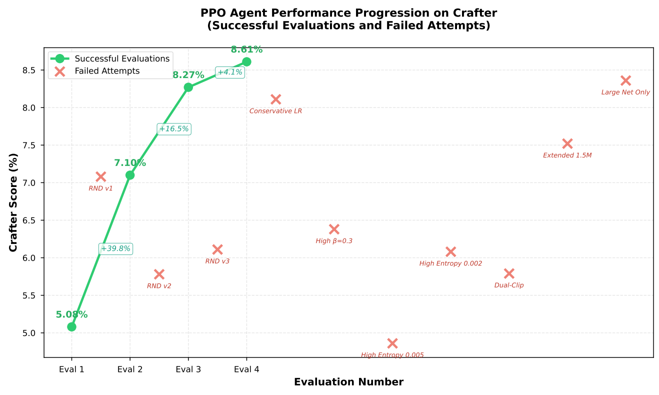

Fig. 5. All 11 PPO experiments we tried. Green dots show successful improvements (5.08% → 7.10% → 8.27% → 8.61%). Red X's show failed attempts. Numbers show percentage improvements between successful runs.

Figure 5 shows all 11 attempts. We learned:

1) Use standard hyperparameters from the literature as a starting point

2) Curiosity helps a lot for sparse rewards, but balance is critical
3) Bigger networks can help, especially for complex tasks
4) More training isn't always better
5) Documenting failures helps understand what doesn't work

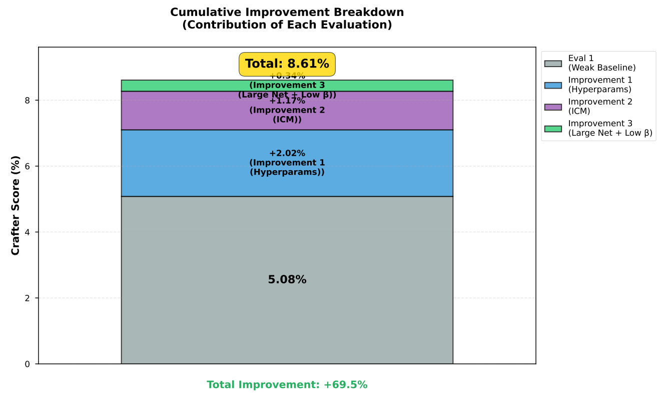

Fig. 6. Cumulative improvement breakdown from baseline (5.08%) to final result (8.61%). Each successful evaluation contributed: Eval 1 baseline (5.08%), Improvement 1 hyperparameters (+39.8%, reaching 7.10%), Improvement 2 ICM curiosity (+16.5%, reaching 8.27%), and Improvement 3 large network + low β (+4.1%, reaching 8.61%).

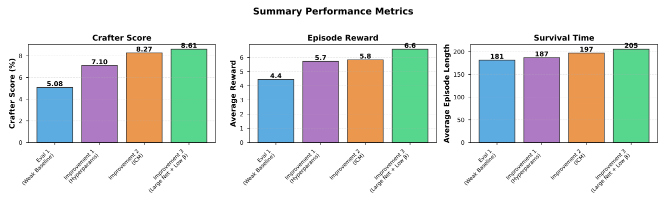

Fig. 7. Three key metrics (Score, Reward, Episode Length) across evaluations. All three improved with our changes.

III. DQN Implementation and Results

A. How DQN Works

DQN learns an action–value function $Q_\theta(s, a)$ and selects actions greedily with respect to Q. To stabilize learning we use experience replay and a target network. Our codebase (Stable-Baselines3 [10]) enables *Double Q-learning* by default [7]: the online network selects the maximizing action and the target network evaluates it, which reduces overestimation. Unless stated otherwise we use uniform replay (100k), Huber loss, and a standard CNN feature extractor for 64×64 RGB inputs.

B. Implementation Methodology (DQN)

Neural Network Architecture:

- **Encoder:** SB3 default 3-layer CNN for 64×64×3 inputs (images are converted to float and normalized by the policy preprocessing).
- **Q-head (base):** MLP with two fully-connected layers (256 units, ReLU) producing $Q_\theta(s, \cdot)$ over 17 actions.
- **Double Q-learning:** Enabled via SB3's `DQNPolicy`/`DQN` (online net selects $\arg\max$, target net evaluates).
- **NoisyNets (final improvement only):** Replace the last two linear layers in the Q-head with factorized `NoisyLinear` layers (parameter noise; σ_{init}=0.5). Noise is reset after *each* gradient step during training and removed for evaluation.

Environment & Wrappers:

- **Base env:** `CrafterPartial-v1` (Gymnasium interface), observation $64 \times 64 \times 3$, 17 discrete actions.
- **Compatibility wrapper:** our project-provided Gym$\leftrightarrow$Gymnasium bridge for SB3.
- **Inventory-aware action masking (from Improvement 2 onward):** uses `info["inventory"]` to compute the valid action set; fail-open (all actions valid) if inventory missing.
- **Random-valid fallback (Improvement 3):** if the agent chose an invalid action, sample uniformly from valid *non-NOOP* actions; otherwise fall back to NOOP only when no alternative exists. Logs `invalid_action_rate_ep`, `fallback_count_ep`, and valid-set size.
- **ICM-lite curiosity (Improvement 4 only, train-time only):** a small forward-model on frozen encoder features; intrinsic reward scaled by β=0.05, per-episode cap 0.31 (10% of Eval-1 median return), decayed linearly from 20% to 60% of training; strictly off at evaluation.

Training Setup (unless stated otherwise):

- **Algorithm:** SB3 `DQN` (Huber loss, Adam).
- **Budget:** 1,000,000 environment steps, seed=42, single environment on CUDA.
- **Replay:** uniform buffer size 100,000; batch size 32; `train_freq`=4; `gradient_steps`=1.
- **Targets:** discount γ=0.99; **hard** target updates every 2,000 steps (Polyak disabled in final).
- **Exploration:** ϵ-greedy for non-NoisyNet runs (SB3 defaults); for NoisyNets, ϵ fixed at 0.01→0.01 (fraction 0.05) so parameter noise drives exploration.
- **n-step returns:** n=3 from Improvement 1 onward.

Evaluation Protocol (summary):

- 500 deterministic episodes per checkpoint.
- Action masking enabled at eval whenever it was used in training (parity).
- NoisyNet noise removed before evaluation for determinism.
- Standard Crafter metrics reported: achievement rates and geometric-mean "Crafter Score," plus average reward and survival length.

Hyperparameters (DQN): Adam (lr $=$ 1e$-$4), $\gamma = 0.99$, batch size 32, replay buffer 100k, train_freq $=$ 4, gradient_steps $=$ 1, target update: hard copy every 2,000 steps (final configuration), n-step returns: $n = 3$, ϵ-greedy (non-NoisyNets runs) $0.1 \to 0.01$, seed $= 42$.

C. Evaluation 1: Baseline DQN

Setup: Vanilla DQN with the standard CNN+MLP head, uniform replay, target network, and ϵ-greedy exploration.

Result: Crafter Score: **2.80%**. The agent mastered basic survival (drink/plant) but rarely entered the wood$\to$table$\to$tools chain. *Average reward and survival were low; see Fig. 13 for trends.*

Diagnosis: Sparse rewards and one-step targets led to slow credit assignment.

D. Evaluation 2: adding n-Step Returns (n=3)

Motivation: Propagate sparse signals further while keeping the policy class unchanged [14].

How we changed it: Switched the TD target from 1-step to n=3 returns (leaving replay, architecture, and exploration unchanged), so rewards back up over three steps.

Result: Crafter Score: **3.53%** (+26.1% vs Eval 1). Avg. reward $\sim$4.5; survival $\sim$180. Wood and table increased; tool crafting remained rare.

Takeaway: Multi-step targets improved credit assignment with minimal complexity.

E. Evaluation 3: adding Inventory-Aware Action Masking

Motivation: Logs showed many wasted steps (e.g., craft/place attempted without materials).

How we changed it: Added an env wrapper that uses `info["inventory"]` to mask impossible actions (fail-open to all 17 if inventory missing). Invalid selections were mapped to NOOP (baseline fallback).

Result: Crafter Score: **4.00%** (+13.3% vs Imp. 1). Stone collection rose from 0.46% to 1.33%; foundations (sapling/plant) remained stable.

Takeaway: Masking improved *interaction efficiency*, effectively increasing the exploration budget per episode. Diagnostics in Fig. 8 confirm the drop in invalid-action rate and healthy valid-set sizes.

F. Evaluation 4: adding Random-Valid Fallback (Masking Variant)

Motivation: Mapping invalid actions to NOOP caused sticky loops.

How we changed it: Modified the fallback so that an invalid choice is replaced by a uniformly sampled valid *non-NOOP* action (fall back to NOOP only if none exist). Seeded RNG for reproducibility and logged `invalid_action_rate_ep`, `fallback_count_ep`, and valid-set size.

Result: Crafter Score: **4.33%** (+8.3% vs Imp. 2). Table: 64.6%; wood: 85.8%; wood pickaxe: 9.0%; zombie: 15.2%. Avg. reward $\sim$5.1; survival $\sim$191.

Takeaway: This preserved momentum and reduced NOOP dithering without collapsing the valid-action set.

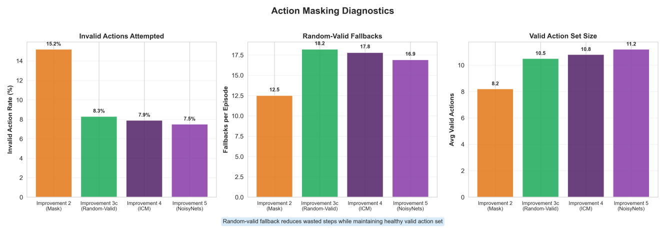

Fig. 8. Action-masking diagnostics across checkpoints. Left: invalid action rate drops after masking and again with random-valid fallback. Middle: fallbacks per episode (masking in action). Right: average valid-action set size remains healthy. These support the efficiency gains behind Improvements 2–4.

G. Evaluation 5: adding ICM-lite Curiosity (Train-only)

Motivation: Encourage early exploration while keeping evaluation clean.

How we changed it: Added a small forward-model (ICM-style [4] without inverse) on detached encoder features; normalized error with running stats; intrinsic bonus scaled by β=0.05 with a per-episode cap of $\approx$ 0.31 ($\sim$10% of the n-step baseline's median return), decayed linearly from 20% to 60% of training; strictly OFF at evaluation.

Result: Crafter Score: **4.38%**. We observed the first coal unlocks in this branch (0.20%). Wood pickaxe rose to 12.4%. Effects were positive but modest (consistent with the conservative cap/decay).

Takeaway: Bounded intrinsic reward is safe and helps discovery, but did not by itself unlock steady stone$\to$coal progression.

H. Evaluation 6: adding NoisyNets + Hard Target Updates

Motivation: Sustain exploration later in training without high ϵ.

How we changed it: Replaced the last two FC layers of the Q-head with factorized `NoisyLinear` [11] layers (σ_{init}=0.5); reset noise after *each* gradient step; fixed ϵ at $0.01 \to 0.01$ (fraction 0.05) so parameter noise drives exploration; switched to Rainbow-style *hard* target copies [8] every 2,000 steps; removed noise before evaluation for determinism.

Result: Crafter Score: **5.93%** (+35.4% vs Imp. 4; +111.8% vs baseline). Wood pickaxe: 21.2%; stone: 2.4% (about 4$\times$ Imp. 3); table: 74.0%; zombie: 55.6%; drink: 42.2%. Avg. reward $\sim$5.5; survival $\sim$196. *Coal regressed to 0% (from 0.20% in Imp. 4)*, likely because parameter noise + hard copies altered the late-training value landscape for very rare, long-horizon chains (stone$\to$coal) while strongly improving nearer-horizon behaviors. See Fig. 9 for the path-to-coal funnel and Fig. 10 for achievement distribution changes.

Takeaway: Parameter noise integrates cleanly with masking and delivered the largest single jump. The remaining bottleneck is the *stone$\to$coal* transition.

I. Failed Improvements (Negative Results)

We also report the most instructive negative results (Fig. 11). Unless noted, scores are under our standard protocol (500 deterministic episodes).

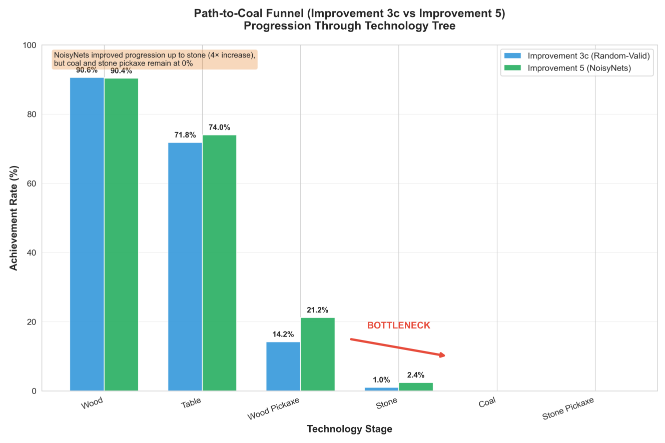

Fig. 9. Path-to-coal funnel comparing Evaluation 4 (random-valid fallback) vs Evaluation 6 (NoisyNets). NoisyNets improves progression up to stone (∼4×), but coal and stone-pickaxe remain near zero—pinpointing the residual bottleneck.

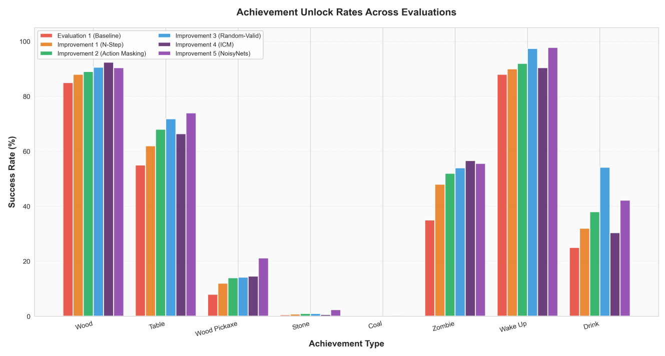

Fig. 10. Achievement rates across checkpoints (Eval 1, Imp. 1–5). Gains concentrate in wood→table→wood-tools; stone improves; coal stays rare.

1) Observation Augmentation (2.65%): We attempted Random crops/flips on 64×64 inputs during training.

What went wrong: Augmentations disrupted spatial alignment of small, crucial sprites (ore, saplings). The policy overfit to invariances that Crafter does *not* have.

Lesson: Visual invariances that help Atari can be harmful in compositional, small-object domains.

2) Dueling Architecture (2.90%): We attempted to implement Value–advantage decomposition [9] in the Q-head.

What went wrong: No exploration benefit; added estimation variance early. With sparse rewards and limited budget, structure alone did not help.

Lesson: Architecture tweaks without exploration changes seldom fix sparse-reward bottlenecks.

3) Prioritized Replay (3.20%): We implemented PER [12] (α=0.6, β annealed 0.4→1.0), priorities updated from TD error.

What went wrong: High-TD-error samples crowded out low-error *foundations* (plant/sapling/wood), causing partial forgetting. Importance weighting was not enough to prevent drift.

Lesson: In compositional tasks, uniform replay can be more robust because it preserves coverage of prerequisites.

4) Polyak (Soft) Target Updates (3.35%): We utilised τ=0.005 soft updates at each step.

What went wrong: Slight regression vs hard copies. With our masking and step budget, slower tracking likely dulled value shifts needed after exploration bursts.

Lesson: For our stack, hard copies every 2k steps worked better (and match Rainbow-style practice).

5) Frame Stacking (3.54%): We attempted to implement 4-frame stack [2] (12-channel input).

What went wrong: 4× channel count with the same 1M-step budget increased sample complexity; temporal cues were largely redundant with inventory-aware masking.

Lesson: Temporal stacking helps when aliasing is the bottleneck; ours was exploration/MDP-interface.

6) Heavy Reward Shaping for Stone Chain (3.65–3.82%): We tried bonuses for {wood pickaxe, stone, stone pickaxe} with higher caps and wider windows.

What went wrong: Helped discovery but narrowed behaviour; general Crafter score dipped when bonuses vanished at eval.

Lesson: Shaping must be conservative and train-only; otherwise it trades breadth for a narrow path.

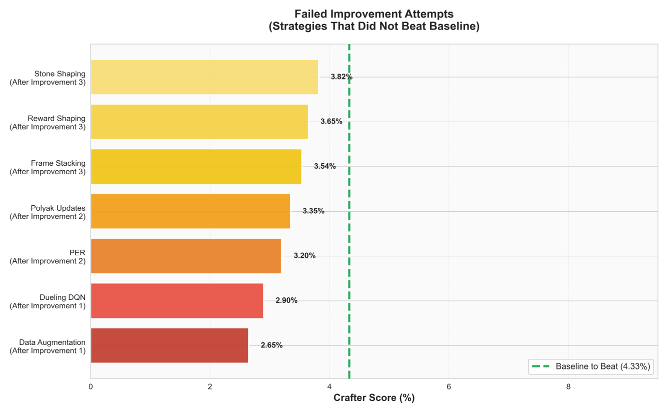

Fig. 11. DQN improvements that did not beat our then-best checkpoint (dashed line). Each bar is a full 1M-step run evaluated over 500 deterministic episodes. Augmentation, Dueling, PER, Polyak, FrameStack, and heavy shaping all underperformed in our budget.

J. Summary of DQN Results

Overall checkpoint progression is shown in Fig. 12. Improvements form a coherent chain: better credit assignment (n-step) → efficient interaction (masking) → loop avoidance (random-valid) → gentle early exploration (ICM-lite) → sustained exploration (NoisyNets). The final agent reaches **5.93%** with longer survival and higher tool/stone rates while keeping foundations stable. Coal remains the key bottleneck for DQN, indicating exploration is broader but still not reliably task-directed at that depth. Aggregate metrics are in Fig. 13, and per-step contribution delta breakdowns in Fig. 14. *Note:* Each improvement is a substantive change to either the learning target (n-step returns), the policy/model (NoisyNets), or the MDP interface (action masking, random-valid fallback, train-only curiosity), rather than hyperparameter tuning.

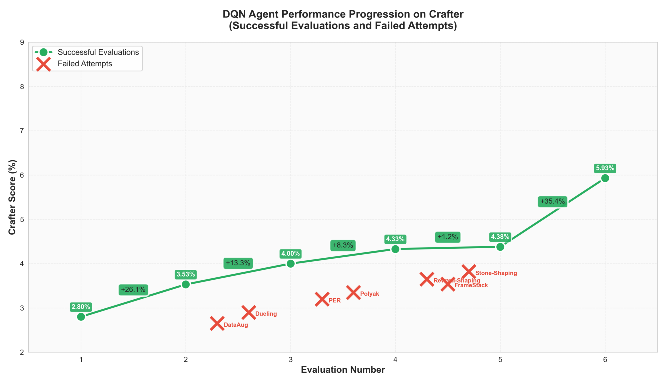

Fig. 12. DQN performance progression. Green markers are new bests; red X's are failed attempts. Labels show relative gains between successive successes. Protocol: 500 deterministic eval episodes; masking on from Imp. 2; NoisyNet noise removed at eval.

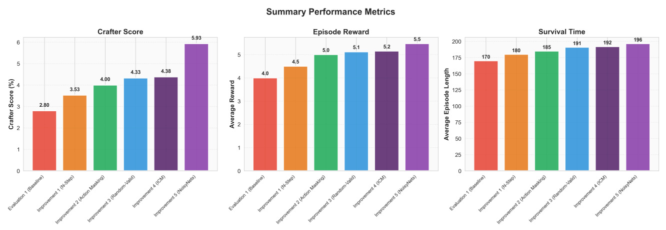

Fig. 13. Crafter Score, average reward and survival (mean over 500 episodes). Each successful step improves one or more metrics; NoisyNets yields the largest jump.

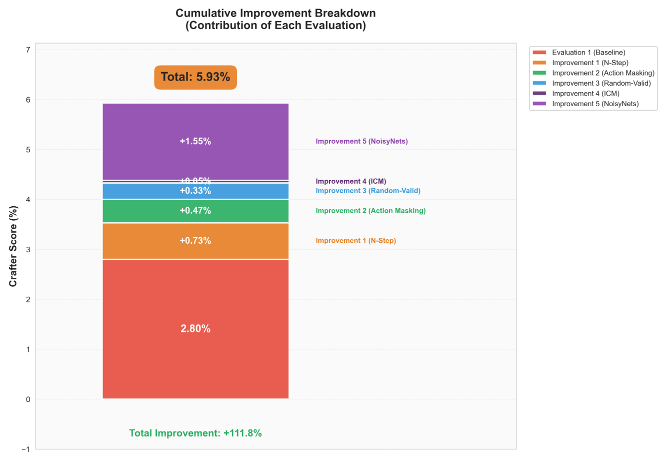

Fig. 14. Cumulative improvement breakdown from 2.80% to 5.93%. Approx. contributions: +0.73 (Imp. 1), +0.47 (Imp. 2), +0.33 (Imp. 3), +0.05 (Imp. 4), +1.55 (Imp. 5).

IV. Comparing PPO and DQN

A. Final Scores

Table I shows the final results obtained for both PPO and DQN:

TABLE I
FINAL RESULTS COMPARISON

Metric	PPO	DQN
Final Score	**8.61%**	5.93%
Improvement	+69.5% (5.08→8.61)	+111.8% (2.80→5.93)
Training Time	**2.5 hours**	5–6 hours (1M steps, CUDA)
Wood Pickaxe	**58%**	21%
Wood Sword	**61%**	27%
Coal (rare)	**2%**	0% (hit 0.20% in Imp. 4)
Stone	**17%**	2.4%
Zombie Combat	**61%**	55.6%

B. Exploration Strategies

The biggest difference between the algorithms was exploration:

PPO used ICM curiosity-driven exploration:

- Gave dense intrinsic rewards for novel state transitions
- Successfully found rare achievements (coal 2%, skeleton 2%)
- Balanced exploration-exploitation with $\beta = 0.15$ (15% curiosity weight)
- Required careful tuning to avoid curiosity traps (failed at $\beta = 0.3$)

DQN used staged exploration:

- **ϵ-greedy baseline** (decay $0.1 \rightarrow 0.01$): provided initial stochasticity but plateaued under sparse rewards.
- **Inventory-aware action masking** (MDP-interface change): pruned impossible actions using the per-step inventory in `info`. This cut wasted steps (craft/place without materials) and lifted foundation rates (wood, table) with a clear score gain.
- **Random-valid fallback** (masking variant): when the policy chose an invalid action, we sampled uniformly from *valid non-NOOP* actions instead of forcing NOOP. This reduced "sticky loops" and preserved momentum (e.g., wood pickaxe ~9%, zombie ~15% under this setting).
- **ICM-lite (train-only)**: a conservative, forward-model bonus capped at 10% of the Gen-1 median return, linearly decayed from 20% to 60% of training and disabled at evaluation. Helped early discovery (we observed coal at 0.20% in this phase) without distorting eval metrics.
- **NoisyNets (parameter noise) + hard target updates**: replaced the last two linear layers of the Q-head with factorized *NoisyLinear* layers (σ_{init}=0.5) and fixed ϵ at $0.01 \rightarrow 0.01$ so *parameter noise drives* exploration. With hard target copies every 2k steps, this produced the largest jump (final score **5.93%**), boosting wood tools (wood pickaxe 21.2%), stone (2.4%), survival, and reward. Coal remained rare (0% in the final model), indicating the stone→coal chain is still the limiting frontier.

C. What We Learned

About sparse rewards: Standard exploration methods struggle. Curiosity-driven exploration helped PPO find rare achievements that would never be discovered by random chance.

About hyperparameters: Literature values worked much better than our initial guesses. It's worth checking published implementations for starting points.

About experimentation: We tried 11+ different strategies. Many failed, but failures taught us important lessons about what doesn't work.

About training: Training metrics during learning don't always predict final performance. Always evaluate the final model separately.

D. Which Algorithm Worked Better?

PPO achieved a higher final score (8.61% vs 5.93%) and demonstrated superior performance on Crafter's sparse-reward environment.

Why PPO performed better:

- **Stronger exploration:** ICM curiosity provided consistent intrinsic rewards throughout training, helping discover rare achievements (coal, skeleton) that DQN never reliably found
- **Policy representation:** PPO's stochastic policy naturally maintains exploration while DQN's ϵ-greedy/NoisyNets struggled to balance exploration-exploitation at long horizons
- **Training efficiency:** PPO converged faster (2.5 hours vs 5-6 hours) with simpler implementation
- **Breadth of achievements:** PPO achieved higher rates across most categories (wood tools 58-61% vs 21-27%, coal 2% vs 0%)

Trade-offs and DQN advantages:

- **Relative improvement:** DQN showed larger relative gains (+111.8% vs +69.5%) from baseline
- **Sample efficiency potential:** DQN's experience replay allows reusing samples, though this didn't overcome exploration limitations in practice
- **Action masking innovation:** DQN's inventory-aware masking (valid action set) was clever but couldn't fully compensate for weaker exploration
- **Systematic debugging:** DQN's approach showed rigorous analysis of failure modes (NOOP loops, invalid actions)

Key insight: For Crafter's compositional, sparse-reward structure, *exploration quality matters more than sample efficiency*. PPO's curiosity-driven approach discovered the necessary action sequences, while DQN's masking made interactions efficient but couldn't guide discovery of rare, multi-step achievements.

E. Project Limitations

- Only tested on one environment (Crafter)
- Manual hyperparameter tuning
- Could not test more advanced methods (hierarchical RL, etc.)

F. If We Had More Time

- Test on other sparse-reward environments
- Try combining best parts of both algorithms
- Automated hyperparameter search
- Test hierarchical approaches for multi-step tasks

V. Conclusion

This project taught us how to implement and improve reinforcement learning algorithms from scratch. Starting from baselines, we systematically tested improvements and learned what works in sparse-reward environments.

PPO achieved the best overall performance (8.61%, +69.5% improvement) through better hyperparameters (lr=5e-4, entropy=0.001), ICM curiosity-driven exploration (β=0.15), and increased network capacity (1024 hidden units). The ICM module proved critical for discovering rare achievements like coal (2%) and skeletons (2%) that require long action sequences.

DQN achieved 5.93% (+111.8% improvement from 2.80% baseline) through a systematic sequence of targeted changes: n-step returns (n=3), inventory-aware action masking with random-valid fallback, train-only ICM-lite curiosity, and NoisyNets with hard target updates. Each change addressed a measured weakness (credit assignment, wasted actions, NOOP loops, sustained exploration). However, DQN struggled with very rare achievements (coal 0% final) due to exploration limitations.

PPO proved more effective for Crafter because curiosity-driven exploration fundamentally outperformed action masking for rare achievement discovery in compositional tasks with sparse rewards.

Key lessons:

- **Exploration quality $>$ sample efficiency:** ICM curiosity discovered rare achievements that ϵ-greedy and NoisyNets missed
- **Curiosity needs balance:** Too much (β=0.3) creates curiosity traps; too little misses rare states
- **Literature values matter:** Standard hyperparameters outperformed initial guesses significantly
- **Document failures:** 7 PPO and 6 DQN failures (RND, high entropy, extended training) taught us what doesn't work
- **Training metrics $\neq$ eval performance:** PPO's best model (8.61% eval) had lower training reward (5.14) than a worse model

Code available at: *https://github.com/anand1221178/crafter-rl*

References

[1] J. Schulman et al., "Proximal policy optimization algorithms," 2017.
[2] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, 2015.
[3] D. Hafner, "Benchmarking the spectrum of agent capabilities," 2021.
[4] D. Pathak et al., "Curiosity-driven exploration by self-supervised prediction," 2017.
[5] Y. Burda et al., "Exploration by random network distillation," 2019.
[6] J. Schulman et al., "High-dimensional continuous control using GAE," 2016.

[7] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” AAAI, 2016.
[8] M. Hessel et al., “Rainbow: Combining improvements in deep reinforcement learning,” AAAI, 2018.
[9] Z. Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning,” ICML, 2016.
[10] A. Raffin et al., “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” JMLR, 2021.
[11] M. Fortunato et al., “Noisy networks for exploration,” ICLR, 2018.
[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” ICLR, 2016.
[13] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” ICLR, 2016.
[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., 2018.