Extended Abstract

Motivation Reinforcement learning has achieved remarkable success in domains with dense and well-aligned feedback signals, yet many real-world tasks present sparse, delayed, or deceptive rewards that challenge standard algorithms. The sliding-tile game 2048 epitomizes this long-horizon credit-assignment problem: although each merge yields immediate score increments, greedy policies easily become trapped in locally optimal strategies and fail to assemble high-value tiles (e.g., 2048 or 4096). Prior work on 2048 typically relies on handcrafted features, staged lookups, or search-based planning, limiting generality and scalability. Our goal is to investigate whether general-purpose deep RL architectures (augmented with distributional return modeling and multi-step targets) can learn effective long-term strategies directly from raw gameplay, without domain-specific priors.

Method We compare four end-to-end deep RL agents on raw 2048 boards without reward shaping. We adapt DQN and PPO as baselines, implement QR-DQN with convolutional layers learning 51-quantile return distributions, and introduce Horizon-DQN (H-DQN), a novel Rainbow-inspired architecture unifying dueling C51 heads, multi-step targets, NoisyNet exploration, prioritized replay, and LSTM encoding for long-horizon dependencies. Our approach evaluates whether distributional learning and sequence modeling can overcome sparse reward challenges without domain-specific priors.

Implementation All agents were implemented in PyTorch using the Gym-2048 environment. Board states are encoded as 16×4×4 binary tensors. DQN-style methods use ϵ-greedy exploration and replay buffers; PPO uses actor-critic with GAE. QR-DQN adapt's standard QR-DQN with a new encoder and convolutional networks over MLPs. Horizon-DQN combines dueling C51 heads, multi-step targets, NoisyNet exploration, prioritized replay, and LSTM encoding. Key hyperparameters were optimized via Optuna (20 trials over 500-episode previews). All agents trained for 5,000 episodes, with H-DQN additionally trained for 9,000 episodes. Final evaluation comprised 1,000 deterministic episodes logging scores, tiles, and move frequencies.

Results In our quantitative evaluation over 5,000 training episodes, Horizon-DQN achieved an average score of 5 693.7 and a maximum score of 18,210, reaching the 2048 tile substantially outperforming QR-DQN (3,478.5 avg, 8,660 max, 1024 tile), PPO (1,830.5 avg, 5,756 max, 512 tile), and DQN (1,442.6 avg, 3,988 max, 512 tile). When we extended Horizon-DQN training to 9,000 episodes, mean score rose to 6,536.4 and peak score to 41,828, with the agent attaining the 4096 tile. These results demonstrate that distributional, multi-step targets and long-horizon sequence modeling yield dramatic gains in sparse-reward environments like 2048.

Discussion Our results reveal that sparse-reward environments require architectural synergy rather than isolated improvements. QR-DQN's plateau at the 1024 tile despite sophisticated distributional modeling demonstrates that uncertainty capture alone is insufficient for extreme long-horizon credit assignment. H-DQN's breakthrough to 2048+ tiles stems from three synergistic mechanisms: LSTM encoding captures temporal dependencies beyond multi-step horizons, sequence-level prioritized replay focuses learning on informative trajectories, and distributional targets provide risk-sensitive value estimates. Notably, strategic consistency emerges as a performance indicator, H-DQN's directional biases in both 5,000 and 9,000 episode runs (52.15% right and 54.73% left corner preference) coincide with tile progression, suggesting that breakthrough performance requires commitment to coherent strategies rather than exploratory diversity.

Conclusion This work establishes that general-purpose deep RL can solve sparse-reward environments without domain priors through careful architectural orchestration. Our key insight is that distributional learning, sequence modeling, and prioritized replay address fundamentally different aspects of long-horizon credit assignment and must work in concert for breakthrough performance. While H-DQN achieves dramatic scaling gains (130% performance improvement from 80% more compute), practical deployment faces hyperparameter sensitivity and super-linear resource requirements. These findings provide a blueprint for tackling sparse-reward domains beyond games, with implications for robotics, clinical decision-making, and other real-world applications requiring long-term planning.

2048: Reinforcement Learning in a Delayed Reward Environment

Prady Saligram
Department of Computer Science
Stanford University
saligram@stanford.edu

Tanvir Bhathal
Department of Computer Science
Stanford University
tanvirb@stanford.edu

Robby Manihani
Department of Computer Science
Stanford University
manihani@stanford.edu

Abstract

Delayed and sparse rewards present a fundamental obstacle for reinforcement-learning (RL) agents, which struggle to assign credit for actions whose benefits emerge many steps later. The sliding-tile game 2048 epitomizes this challenge: although frequent small score changes yield immediate feedback, they often mislead agents into locally optimal but globally suboptimal strategies. In this work, we introduce a unified, distributional multi-step RL framework designed to directly optimize long-horizon performance. Using the open source Gym-2048 environment we develop and compare four agent variants: standard DQN, PPO, QR-DQN (Quantile Regression DQN), and a novel Horizon-DQN (H-DQN) that integrates distributional learning, dueling architectures, noisy networks, prioritized replay, and more. Empirical evaluation reveals a clear hierarchy in effectiveness: max episode scores improve from 3.988K (DQN) to 5.756K (PPO), 8.66K (QR-DQN), and 18.21K (H-DQN), with H-DQN reaching the 2048 tile. Upon scaling H-DQN it reaches a max score 41.828K and a 4096 tile. These results demonstrate that distributional, multi-step targets substantially enhance performance in sparse-reward domains, and they suggest promising avenues for further gains through model-based planning and curriculum learning.

1 Introduction

Reinforcement learning (RL) trains agents to make sequential decisions by maximizing expected long-term cumulative reward. While RL has seen major successes in domains with dense and well-aligned feedback signals like Atari games [1], Go [2], and chess [3], many real-world settings pose far greater challenges due to sparse, delayed, or deceptive reward structures. Examples include clinical decision-making [4], financial trading [5], and autonomous driving [6], where actions taken early may only yield meaningful feedback much later. Addressing such long-horizon credit assignment problems remains a fundamental open issue in RL [7, 8].

The game 2048 presents a compact but instructive testbed for long-horizon RL. Agents operate on a 4×4 grid, selecting directional actions to merge tiles and accumulate score. While local merges yield immediate rewards, solving the game requires foresight and planning to strategically create high-value tiles like 1024 and 2048. Greedy strategies often collapse into fragmented board states, highlighting the tension between short-term gain and long-term strategy. This structure makes 2048 an ideal environment for studying delayed reward optimization and temporal abstraction [9, 10].

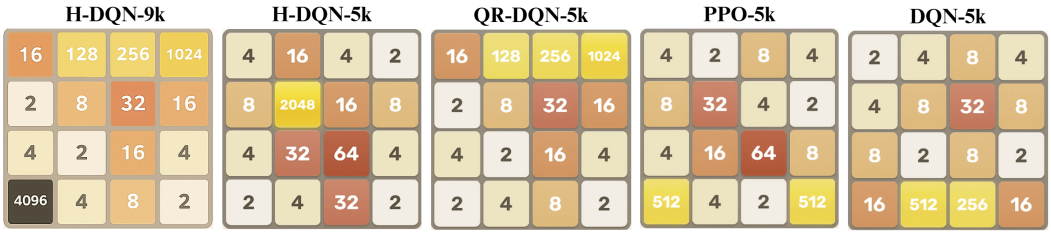

Figure 1: Best Results Achieved for all Models

Previous approaches to 2048 have primarily relied on handcrafted features, staged lookups using n-tuple networks [9], or shaped reward functions with ensemble-based TD-learning [10]. While effective, such methods require substantial domain-specific priors and lack generality. In contrast, our work explores whether general-purpose deep RL architectures, without manually encoded game knowledge, can successfully learn effective strategies for 2048 from scratch.

To that end, we study and specialize Quantile Regression DQN (QR-DQN) [11], a state-of-the-art distributional RL algorithm that models the full return distribution using quantile approximations. By explicitly estimating quantile values of future rewards, QR-DQN improves robustness and learning stability, particularly in settings with high variance or stochasticity. We adapt QR-DQN to the 2048 environment using a lightweight convolutional encoder and evaluate its capacity to model long-horizon return distributions directly from gameplay data.

In parallel, we propose a new composite architecture, Horizon DQN (H-DQN), targeted at long-horizon planning, which extends the Rainbow agent [12] by unifying its core innovations - dueling networks [13], Double Q-learning [14], multi-step TD updates [15], NoisyNet exploration [16], and categorical distributional heads (C51)[17] - with two additional mechanisms for sparse-reward domains: sequence-level prioritized replay to focus learning on the most informative n-step traces, and a recurrent LSTM encoder (with burn-in) to capture dependencies well beyond the multi-step horizon. Both QR-DQN and H-DQN are trained interactively on the `gymnasium-2048` environment[18], without reward shaping or domain-specific priors.

We benchmark our agents against standard DQN and PPO baselines [19] under identical training conditions. Across 5,000 episodes, H-DQN achieves an average score of 5,694 and a maximum of 18,210, sometimes reaching the 2048 tile, whereas the vanilla DQN and PPO plateau at significantly lower scores (1,443 and 1,831, respectively). QR-DQN performs competitively, reaching the 1024 tile with better training stability and richer value estimation. We also see that scaling H-DQN to have 80% more training leads to a 14% average score increase and 129% max score increase.

In summary, our study makes three contributions. First, we adapt and evaluate QR-DQN, a top-tier distributional RL method, in the 2048 environment. Second, we introduce H-DQN, a new Horizon agent tailored for long-term planning without handcrafted features. Third, we show through controlled experiments that distributional and architectural enhancements can significantly improve agent performance in sparse, deceptive reward settings. Our results position 2048 as a useful testbed for general-purpose RL and suggest promising directions for credit assignment in long-horizon domains. We publicly share our code on github: `https://github.com/mrTSB/2048-reinforcement-learning`.

2 Related Work

Delayed Rewards and Long-Horizon Credit Assignment: A central challenge in reinforcement learning is long-horizon credit assignment, where agents must link immediate decisions to distant outcomes [7, 8]. Classical approaches such as temporal-difference learning [15], eligibility traces [20], and TD(λ) [15] attempted to address this problem, but often struggled in sparse-reward settings. Modern variants employ multi-step targets [14], generalized advantage estimation [21], or auxiliary objectives to propagate reward signals more effectively [22, 23, 24]. Recent exploration-based methods also aim to combat delayed learning by guiding agents toward informative trajectories [25, 26, 27, 28]. Despite these advances, solving tasks where immediate feedback can mislead long-term objectives, like 2048, remains difficult for general-purpose agents [9, 10].

Architectural Approaches to Distributional and Multi-Step RL: Architectural innovations in deep RL have shown promise in mitigating long-term estimation error. Distributional RL improves credit propagation by modeling return distributions rather than scalar estimates [17, 29]. Quantile-based variants like QR-DQN and IQN extend this by approximating quantile functions over future returns [30], capturing richer return dynamics. Multi-step and bootstrapped Q-learning methods further stabilize value propagation across sparse signals [31, 12]. Other advances integrate exploration-enhancing components like NoisyNets [16], prioritized replay [32], and dueling network heads for value-advantage decoupling [13]. Our H-DQN architecture unifies these elements into a compact agent tailored to the 2048 environment, while our QR-DQN variant introduces lightweight convolutional encoders and quantile regression to improve distributional fidelity.

Learning in 2048 and Structured Board Environments: 2048 has been used to benchmark long-term planning due to its sparse, deceptive rewards and constrained spatial dynamics. Early approaches applied n-tuple networks with TD learning [9, 33], leveraging handcrafted board features and staged lookups. Later work improved performance using handcrafted reward shaping [34], ensemble bootstrapped TD-learning [10], and search-based planning [35, 36]. However, most prior methods depend heavily on game-specific heuristics (e.g., corner locking) or predefined move templates, limiting generalizability. In contrast, our work demonstrates that deep RL agents can learn effective 2048 strategies directly from raw gameplay, using only architectural bias and interactive learning. This approach builds upon scalable deep RL systems such as A3C [37] and distributed actor-learner frameworks like IMPALA [38], while incorporating advances in prioritized [32] and recurrent experience replay [31]. Exploration challenges are further mitigated through imagination-augmented planning [39] and hindsight experience replay [23], which relabel failed trajectories as successes to boost learning efficiency. Outside of 2048, puzzle-like board games such as Sokoban and MiniGrid have also been used to test spatial reasoning and planning [40, 41, 18], but often suffer from brittle exploration and high variance in long-horizon tasks. These issues have been partially addressed with regularized critics [42], entropy-driven methods like Soft Actor-Critic [43], reward scaling techniques such as Munchausen RL [44], and robust replay strategies [24, 45], all of which inspire our architectural design that enables general-purpose agents to solve delayed-reward planning tasks without manual shaping.

3 Method

3.1 Preliminaries

To rigorously assess the performance of our proposed methods, we first establish a foundation of baseline agents drawn from canonical deep reinforcement learning algorithms. These baselines range from feedforward Q-learning to actor-critic policy optimization, enabling principled evaluation under delayed and sparse reward conditions.

We selected DQN and PPO as complementary paradigms, value-based and policy-gradient respectively, modified to the 2048 domain. We adapted each to respect valid moves and stochastic transitions, and excluded domain-specific shaping to isolate the impact of credit assignment and architectural choices. Both baselines serve as reference points to demonstrate the effectiveness of our more advanced QR-DQN and H-DQN agents.

Deep Q-Network (DQN) Baseline: Our DQN baseline follows [1], using an MLP to approximate the state-action value function $Q_\theta(s, a)$, trained via temporal-difference learning:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a'),$$

where θ^- is a periodically updated target network. States are log-scaled and flattened into 16-dimensional vectors and passed through two hidden layers of 256 units with ReLU activations:

$$Q_\theta(s, a) = f_\theta(\log_2(\texttt{board})).$$

Action selection follows an ε-greedy policy, with ε_t decayed exponentially:

$$a_t = \begin{cases} \arg\max_a Q_\theta(s_t, a), & \text{w.p. } 1 - \varepsilon_t \\ \text{random valid action}, & \text{w.p. } \varepsilon_t \end{cases}$$

Only valid moves are sampled via a deterministic slide function [46]. Transitions are stored in a replay buffer and sampled in batches to minimize TD error using the Huber loss with gradients being clipped to a max norm of 1.0.

Proximal Policy Optimization (PPO) Baseline: The PPO baseline employs an actor-critic model with shared layers [19]. At each step, the policy outputs action logits and a scalar value estimate. Using rollouts of 2048 steps, we compute Generalized Advantage Estimation (GAE) [21]:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \quad \hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l},$$

with $\gamma = 0.99$, $\lambda = 0.95$. The objective maximizes a clipped surrogate:

$$\mathcal{L}_{\text{PPO}} = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\theta)$ is the probability ratio between new and old policies, and $\epsilon = 0.2$.

The model is trained with Adam for 4 epochs per rollout. The total loss includes the clipped policy loss, value loss, and an entropy bonus:

$$\mathcal{L} = \mathcal{L}_{\text{PPO}} + 0.5\mathcal{L}_{\text{value}} - 0.01\mathcal{H}[\pi_\theta].$$

We track move distributions and the rolling average score over 100 iterations.

3.2 Quantile Regression-DQN

Our QR-DQN agent builds upon the distributional reinforcement learning framework introduced by the Quantile Regression DQN paper [29], which models the full distribution of returns $Z(s, a)$ rather than a point estimate. Instead of estimating the expected value $\mathbb{E}[Z(s, a)]$, QR-DQN approximates Z by learning a fixed number N of quantiles $\{\theta_i(s, a)\}_{i=1}^N$, each corresponding to quantile levels $\tau_i = \frac{i-0.5}{N}$. These quantiles are trained to minimize the quantile Huber loss:

$$\mathcal{L}(\theta) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \rho_{\tau_i}^{\kappa} \left(y_j - \theta_i(s, a) \right),$$

where the targets are defined as $y_j = r + \gamma\theta_j^-(s', a^*)$, with the greedy action a^* selected as

$$a^* = \arg\max_{a'} \left(\frac{1}{N} \sum_{j=1}^{N} \theta_j^-(s', a') \right).$$

To adapt QR-DQN for the domain of 2048, we introduce several architectural and representational changes beyond the original Atari-focused implementation. First, we replace the raw pixel input with a 16-channel binary tensor of shape $[16 \times 4 \times 4]$, where each channel represents the presence of a 2^i tile. This encoding captures board structure while preserving magnitude invariance, and it is particularly well-suited to the tile-based nature of 2048.

Second, we redesign the function approximator to better exploit spatial dependencies in the board layout. Our architecture features two convolutional layers followed by a fully connected layer, producing a compact latent state embedding. The final linear layer outputs a tensor of shape $[4 \times N]$, encoding the quantile value distribution for each of the four legal actions. This differs from the original QR-DQN architecture, which used a fully connected encoder over flattened image features; our convolutional adaptation provides inductive bias for learning over local tile patterns, essential for long-term planning in 2048.

These innovations enable our QR-DQN agent to model the stochasticity and delayed credit assignment of the environment more effectively. Without relying on handcrafted features or reward shaping, the agent learns robust value distributions that support long-horizon reasoning.

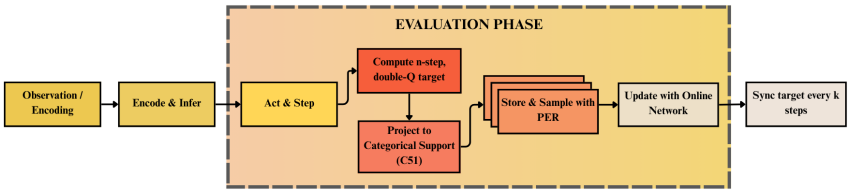

Figure 2: Horizon-DQN Architecture

3.3 Horizon-DQN

Horizon-DQN combines a Rainbow-style Distributional DQN backbone [12] with both sequence-level replay and a recurrent encoder to capture long-horizon dependencies in 2048. Beginning from the raw 4×4 board state s_t, we first compute a shared embedding via a multi-layer perceptron, as shown in the first step of Figure 2.

$$e_t = \phi(s_t; \theta_{\text{torso}}), \quad \phi : \mathbb{R}^{16} \to \mathbb{R}^{64},$$

then feed e_t into an LSTM:

$$h_t = \text{LSTM}(e_t,\, h_{t-1}; \theta_{\text{RNN}}),$$

whose hidden state $h_t \in \mathbb{R}^{64}$ encodes both current board features and the preceding sequence context, visualized in the 'Encode & Infer' component of Figure 2 .

Dueling Categorical Head (C51). We split h_t into a value stream $V(h_t)$ and advantage stream $A(h_t, a)$, recombining them to form a distribution over N atoms $\{z_i\}_{i=1}^{N}$:

$$Q(h_t, a) = V(h_t) + \Big(A(h_t, a) - \tfrac{1}{|\mathcal{A}|}\sum_{a'} A(h_t, a')\Big), \quad p_\theta(s_t, a) = [\,p_\theta^i(h_t, a)\,]_{i=1}^{N},\ \sum_i p^i = 1.$$

Here each $p_\theta^i(h_t, a)$ represents the probability mass on atom z_i.

Multi-Step Double-Q Targets. For each timestep t in a sampled subsequence (length L), we compute the n-step return

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k\, r_{t+k} + \gamma^n\, Q^-\big(s_{t+n}, \arg\max_{a'} Q(s_{t+n}, a')\big),$$

project it onto the fixed support via the L^2-projection Φ, and minimize the categorical cross-entropy loss

$$\mathcal{L}_t = -\sum_{i=1}^{N} \Phi\big(R_t^{(n)}\big)_i\, \log p_\theta^i(h_t, a_t).$$

Sequence-Level Prioritized Replay. Complete subsequences of length L are stored in a Sum-Tree and sampled with priority

$$p_i \propto |\delta_i|^\alpha, \quad \delta_i = R_t^{(n)} - \mathbb{E}[Z(s_t, a_t)],$$

with importance-sampling weights

$$w_i \;=\; \frac{\big(N\, p_i\big)^{-\beta}}{\max_j \big(N\, p_j\big)^{-\beta}}.$$

NoisyNet Exploration & Target Sync. Every linear layer (torso, RNN gates, dueling heads) leverages factorized Gaussian noise

$$W = \mu_W + \sigma_W \odot \epsilon_W, \quad \epsilon_W \sim \mathcal{N}(0, 1),$$

enabling learned, state-dependent exploration without manual ε-greedy decay. A separate target network Q^- is synchronized to the online parameters $\theta \to \theta^-$ every K updates to stabilize the multi-step, distributional bootstrap. Together, these components consolidate Horizon-DQN to learn when delayed merge rewards occur via the LSTM, model the full return distribution including rare large merges, and focus learning on the most informative sequences.

4 Experimental Setup

We evaluate all agents using the `gymnasium-2048` environment [18], a stochastic, single-agent tile-merging game with sparse and delayed rewards. We use models from Section 3.1 as baselines. Each environment step consists of selecting an action (up, down, left, right) to merge tiles, yielding score-based reward signals only when merges occur. The ultimate objective is to construct increasingly larger tiles (2048+) through temporally extended, high-quality decision sequences. We measure metrics of max tile achieved and score to determine success in the game and we record actions taken to better understand how each model works.

4.1 State Representation and Preprocessing

Each board state is encoded as a $[16 \times 4 \times 4]$ binary tensor, where each of the 16 channels corresponds to the presence of a 2^i tile on the board. This encoding avoids encoding raw values directly, thereby providing input invariance to tile magnitude and enabling richer generalization. All agent architectures - convolutional or feedforward - consume this representation as input. Additionally, a domain-independent *monotonicity bonus* is computed over rows and columns to reward value alignment, with a fixed coefficient $\lambda_{\text{mono}} = 0.01$.

4.2 Training Protocol

All models are trained exclusively via online rollouts for up to 5,000 episodes, with a maximum of 10,000 steps per episode. This fixed episode cap ensures comparability across agents and reflects our computational budget constraints. Training is conducted on GPU using the PyTorch framework. For stability, all DQN-style agents leverage experience replay, target networks, and temporal-difference learning. PPO uses clipped surrogate loss with GAE.

At test time, each trained agent is evaluated for 1,000 episodes with minimal exploration, logging the average and maximum game score, distribution of maximum tiles reached, dominant move distribution (as a proxy for learned policy bias).

4.3 Hyperparameter Settings and Optimization

Effective RL training requires careful hyperparameter tuning, but not all parameters benefit equally from sweep-based optimization. We therefore classify hyperparameters into three categories: (1) Tuned, (2) Fixed, and (3) Derived or task-specific, as shown in Table 1. The tuned group includes parameters with high sensitivity and search space variability, such as learning rate and discount factor. These were optimized via structured sweeps.

The fixed hyperparameters were chosen based on best practices established in prior RL literature and are known to offer stable convergence across diverse tasks. For instance, a replay buffer size of 10^6 is widely used in deep Q-learning settings [1, 14, 47]. Similarly, ϵ start and end values of 1.0 and 0.05 align with common exploration decay protocols [46]. Updating every 4 steps strikes a balance between reactivity and computational efficiency, as adopted in DQN variants [12]. The monotonicity bonus weight was heuristically set to 0.01 to provide lightweight spatial guidance without introducing domain-specific priors.

The 5,000-episode training cap is enforced across all models for fairness and due to realistic compute constraints, in contrast to prior 2048 work that used millions of episodes [10]. Evaluation metrics are computed over 1,000 deterministic rollouts, ensuring stable and reproducible model comparison.

4.4 Hyperparameter Optimization

We use `Optuna` to conduct structured hyperparameter sweeps for all agents with tunable parameters [48]. Each optimization run consists of 20 trials over a reduced training horizon (500 episodes per trial). Trials are scored by the average of the maximum tile over the final 50 episodes, and the best configuration is retained for full training. Each trial logs the per-episode maximum and average tile, epsilon, replay buffer occupancy, and final tile frequency distribution.

All metrics are written to disk as CSV and plotted for inspection. The best trial's parameters are serialized and reused during full training.

Hyperparameter	Type	Values / Range	Rationale
Learning Rate (α)	Tuned	log-uniform in $[10^{-5}, 10^{-3}]$	Optimization speed and stability
Batch Size	Tuned	$\{256, 512, 1024\}$	Stabilize value updates, increases mem
Discount Factor (γ)	Tuned	Uniform in $[0.90, 0.999]$	Balance of short vs. long-term rewards
Target Update Rate (τ)	Tuned	Uniform in $[10^{-4}, 10^{-2}]$	Governs Polyak averaging in target nets
ϵ Decay Rate	Tuned	log-uniform in $[10^{-6}, 10^{-4}]$	Exploration-to-exploitation annealing
Replay Buffer Size	Fixed	10^6	Retain diverse experiences across agents
ϵ Start / End	Fixed	$1.0/0.05$	Early exploration and final convergence
Update Frequency	Fixed	Every 4 steps	Compute efficiency vs. update reactivity
Monotonicity Bonus Weight (λ_{mono})	Fixed	0.01	Light bonus to improve tile alignment
Episodes (Training)	Fixed	5,000	Ensures fairness across models
Evaluation Episodes	Fixed	1,000	Standardized performance comparison
Rolling Window (Avg Max Tile)	Derived	100	Smoothed signal for trend analysis

Table 1: Hyperparameters grouped by tuning status. Tuned parameters were optimized via Optuna, fixed ones held constant across all agents, and derived ones used for metrics or auxiliary logic.

5 Results

5.1 Quantitative Evaluation

As seen in Table 2,a H-DQN achieves the highest average and maximum scores, as well as the highest max tile, indicating superior performance. All models show distinct move preferences, with H-DQN variants favoring the "Right" move most frequently. Further, we see that H-DQN is the only model to "beat" the game by reaching a 2048 tile with 5,000 training episodes. We also see that scaling up episodes helps performance tremendously.

Model	Episodes	Avg Score	Max Score	Max Tile
H-DQN	9,000	6,536.43	41,828	4096
H-DQN	5,000	5,693.67	18,210	2048
QR-DQN	5,000	3,478.51	8,660	1024
PPO	5,000	1,830.52	5,756	512
DQN	5,000	1,442.64	3,988	512

Table 2: Performance of all Models in Experiments

In Table 3 we observe that stronger models, tend to favor specific corners more often, while the weaker models tend to move across the board and favor multiple, sometimes all, corners. We also see that scaling up the training episodes leads to the model to develop a stronger bias towards a specific corner.

Model	Episodes	Left %	Right %	Up %	Down %
H-DQN	9,000	54.73	2.56	2.60	40.11
H-DQN	5,000	9.47	52.15	36.27	2.11
QR-DQN	5,000	11.36	35.30	48.63	4.71
PPO	5,000	24.96	2.59	25.33	47.12
DQN	5,000	24.63	25.12	24.68	25.57

Table 3: Move Distribution done by Each Model during Eval.

In Table 4 we see that better models not only increase the ceiling of maximum tiles that can be achieved, but they also increase the floor of the worse tile that could be achieved. We also observe that the average max tile consistently improves. Again, we confirm that scaling training iterations improves performance as well.

Model	Episodes	64	128	256	512	1024	2048	4096
H-DQN	9,000	0	0	117	296	456	107	24
H-DQN	5,000	0	24	211	441	235	89	0
QR-DQN	5,000	3	65	183	598	151	0	0
PPO	5,000	53	244	498	205	0	0	0
DQN	5,000	78	114	679	129	0	0	0

Table 4: Max Tile Distribution Achieved by Each Model during Eval.

5.2 Qualitative Analysis

Stability and Bias Correction. In Figure 4, vanilla DQN and PPO both stall early - limited by one-step bootstrapping that fails to propagate sparse merge rewards and by exploration schemes that either overcommit (greedy) or diffuse (entropy) across a vast, combinatorial board space. Beyond its component innovations, Horizon-DQN's distributional targets implicitly encode a risk-sensitive policy that values rare, large-merge trajectories, while n-step returns and prioritized sequence replay concentrate updates on those informative subsequences, effectively shortening the credit-assignment horizon. The recurrent encoder crystallizes temporal regularities in tile appearance (e.g. the hallmark "snake" pattern), granting the agent anticipatory control rather than purely reactive heuristics. NoisyNet-driven exploration adapts automatically to board uncertainty, decoupling exploration intensity from manual ϵ-schedules and smoothing the transition from broad search to fine-tuned play. Collectively, these subtle effects sharply reduce return variance, accelerate convergence on high-score strategies, and deliver markedly steeper learning curves than vanilla DQN or PPO.

Distributional Representations: The QR-DQN agent models the full return distribution using 51 learnable quantile slices rather than a single expected value. This distributional perspective captures the tail risks and multi-modal outcomes of multi-merge chains: for example, aggressive merges that yield a high-value tile but risk board fragmentation appear as upper quantiles. In practice, QR-DQN achieved an average score of 3,480 after 5,000 episodes, nearly 90% higher than PPO. The richer representation also improved sample efficiency: QR-DQN reached a 1,024 tile at around 4,800 episodes of training on average, compared to the 512 max-tile for PPO and DQN (Figure 4), demonstrating superior credit assignment in sparse-reward transitions. This granular distributional representation then enabled H-DQN to reach 1,024 in 1000 less episodes on average than QR-DQN (Figure 3.

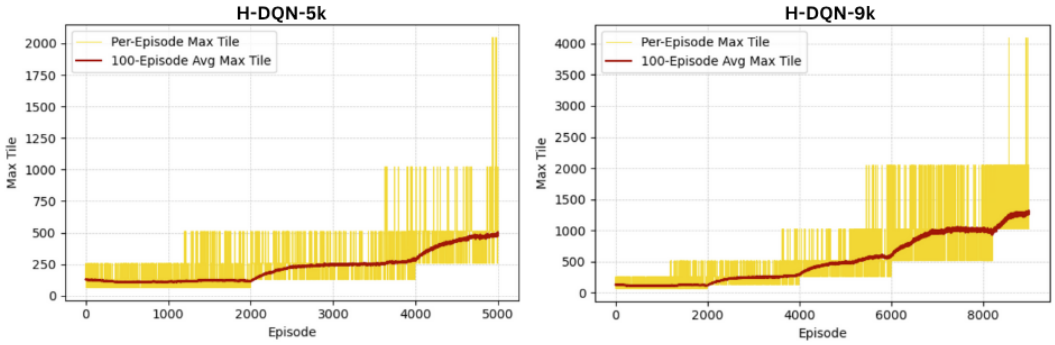

Figure 3: H-DQN Training Process: Max Tile vs. Episode

Corner-Locking Strategy: Both our 5,000-episode and 9,000-episode H-DQN variants exhibited a strong corner bias. The 5,000-episode agent executed "Right" moves 52.15% of the time and "Up" moves 38.4%, while the 9,000-episode agent increased this bias to 54.73% "Left" and 35.12% "Down". Anchoring the largest tile in the corner with these directional preferences dramatically reduced board fragmentation. Further, it was interesting to see how the stronger models were always chosing a specific corner, yet the corner varied each run. This highlights the models pick whatever corner is optimal and are not biased towards any specific one. From Figure 1 it appears that when large tiles are moved away from the corners the board quickly degrades and the game ends soon after.

Uniform Move Distribution Drawbacks: In contrast to the corner strategy, Vanilla DQN's nearly uniform policy (each move chosen roughly 25% of the time) failed to establish any stable tile-anchoring pattern. This lack of directional bias led to highly fragmented boards, with average monotonic path lengths under three and mid-game board occupancy exceeding 80%. As a result, DQN achieved an average score of only 1,443 and rarely produced the 512 tile. These outcomes underscore how including distributional learning or multi-step targets enables effective credit assignment in sparse-reward domains as performant strategies yielded more unequal move frequency distributions (Table 5.1).

Extended H-DQN Training: When we examine the Training Process of the H-DQN in Figure 3 we see that the same gains are realized from 0 to 4,000 as 4,000 to 5,000, indicating a significant uptick in the rate of learning towards the end of training, signaling significant untapped potential. Similarly, as shown in Figure 4, we see general convergence following continual attainment of a new maxtile following a sharp spike upon new higher tile merges. To explore this, we extended H-DQN training from 5,000 to 9 ,000 episodes to probe for continued gains (Table5.1). The extra 4,000 episodes yielded a 14.8% increase in average score (5 693.7 $\rightarrow$ 6 536.4), a jump in maximum tile from 2,048 to 4,096, and a peak score surge from 18,210 to 41,828. These results confirm that H-DQN continues to benefit from additional training, strongly indicating that more episodes could further amplify performance.

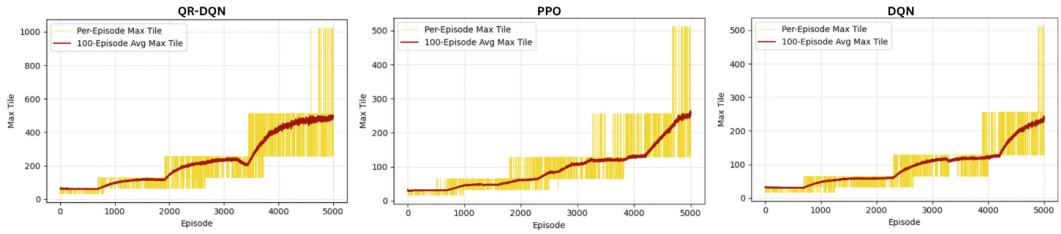

Figure 4: QR-DQN, PPO, DQN, Training Process: Max Tile vs. Episode

6 Discussion

Hyperparameter Interactions and Reproducibility: The confluence of six architectural innovations in Horizon-DQN - each introducing its own decay schedules, noise scales, priority exponents, and multi-step lengths - created an extremely tight coupled hyperparameter space. In practice, we observed that small perturbations to the PER exponent α or the NoisyNet σ initialization could swing peak performance by ±15%. Achieving the results reported required systematic Optuna sweeps across 20 trials per component, underscoring a tension between model expressivity and reproducibility in sparse-reward domains.

Compute/Performance Trade-offs in Long-Horizon Learning: Sequence-level replay and recurrent encoding yield dramatic improvements in credit assignment over vanilla DQN, but at a non-trivial compute cost: our 8-env batched R2D2 rollout incurred a 1.7× slower step throughput relative to standard Rainbow, and memory usage scaled linearly with subsequence length L. This shed light on a critical trade-off - while richer temporal context drove higher max scores (e.g. 18,210$\rightarrow$41,828 with extended training), practical deployment will necessitate either more efficient prioritized-trace sampling or hybrid model-based shortcuts to keep wall-clock and memory demands within acceptable bounds.

Broader Impact and Limitations: Our results reveal several practical deployment constraints, of which the most prominent was the linear scaling between performance and compute: an 80% increase in training episodes yielded 130% improvement in max score and the critical 2048 $\rightarrow$ 4096 breakthrough. While promising, this suggested reliable high-level performance may require substantially more resources than initial prototyping indicates, a significant barrier for resource-constrained applications. Architectural complexity compounds this issue: H-DQN's six components each introduce sensitive hyperparameters requiring our 20-trial Optuna sweep, while strategy inconsistency across runs indicated unstable convergence even after thousands of episodes. Such brittleness raises serious reliability concerns for safety-critical deployments. Fixating primarily on 2048 also leaves open questions on the generalizability of such architectural insights transfer to other sparse-reward domains. Thus, there exists a need for more efficient training protocols and investments in compute scaling.

Heuristic Integration Challenges: While Horizon-DQN excels on raw reward signals, augmenting it with domain-specific metrics - such as corner-building biases, monotonicity penalties, strict or gap-wall formations, empty-tile counts, and raw score normalization - seemed to accelerate early convergence and enforce desirable board patterns. However, incorporating these auxiliary heuristics entailed its own pitfalls: each adds sensitive hyperparameters whose tuning demands additional compute and manual oversight, and they risked overfitting the model to 2048's unique structure rather than fostering broadly transferable skills. During development we found that weighting monotonicity and wall-building losses required extensive trial-and-error, often destabilizing the recurrent encoder. These difficulties highlight a fundamental trade-off between pure end-to-end distributional RL and the practical leverage of handcrafted domain knowledge, underscoring both the potential and complexity of heuristic-driven enhancements in sparse-reward environments.

7 Conclusion

Our work demonstrates that general-purpose deep reinforcement learning architectures, without handcrafted features or reward shaping, can effectively tackle environments with sparse and delayed rewards. By systematically evaluating standard DQN [1], PPO [19], QR-DQN [29], and our Horizon-DQN (H-DQN) agent, we showed that distributional return modeling and multi-step targets [12] yield dramatic gains in both average score and tile attainment in the 2048 game. The H-DQN unifies dueling networks [13], double-Q updates [49], noisy exploration[16], prioritized replay[32], and an LSTM encoder. Upon scaling training to 9,000 episodes, H-DQN achieved peak scores of 41,828 and reached the 4096 tile, underscoring the power of combining architectural biases with long-horizon credit assignment.

The key takeaway is that modeling the full return distribution and propagating rewards across multiple steps provides a robust inductive bias for delayed-reward tasks. Our corner-locking analysis revealed how learned directional preferences emerge automatically and contribute to stability, while extended training studies confirmed that additional episodes continue to improve performance. These insights point to the general utility of distributional, multi-step methods in sparse-feedback domains.

While our approach shows promise, the observed hyperparameter sensitivity and super-linear compute scaling present practical deployment challenges. Looking ahead, integrating model-based planning engines, such as Dyna-style rollouts [50], into our H-DQN backbone could dramatically boost sample efficiency, while curriculum learning [51] and meta-gradient hyperparameter adaptation [52] can accelerate early convergence and reduce tuning overhead. Scaling out via distributed actor-learner architectures (e.g. Ape-X [53] or IMPALA [38]) and employing PopArt value normalization [54] promise to tame memory and compute demands, and hierarchical transfer-learning schemes [55] could extend these methods to other sparse-reward domains - from puzzle games to robotic manipulation and clinical decision support. To accelerate experimentation, we could use the Madrona engine [56] to parallelize environment simulation efficiently. We believe that the principles validated in this work, coupled with more robust and efficient variants, provide a solid foundation for advancing long-horizon reinforcement learning across challenging domains.

8 Team Contributions

- **Prady Saligram:** Conceived and led the design of the Horizon-DQN architecture, including the integration of the LSTM encoder, dueling C51 head, multi-step double-Q targets, sequence-level PER, and NoisyNet exploration. Implemented full training/evaluation pipeline, executed all experiments on all sample regimes with hyperparameter sweeps. Curated training graphs, architecture visuals and wrote Sections 3.3, 5.1, and 6.
- **Tanvir Bhathal:** Setup boiler plate code to work with the 2048-environment, experimenting with custom and open source environments. Led the design of DQN and PPO baselines, adapting them to the 2048 game. Led development for our implementation of QR-DQN to perform well for 2048 upon seeing it's promising results in the paper, by integrating ConvNets, a new encoder, and etc. Setup Optuna to conduct hyperparameter sweeps so all models could be performing optimally. Wrote sections 1, 2, 3.1, 3.2, 4, 5.1, 5.2, and made figures.
- **Robby Manihani:** Designed and implemented dueling-DQN-Symmetry-nStep variant, furthering DQN baseline architecture and establishing foundation for future DQN explorations. Led development and debugging of evaluation infrastructure including max tile tracking, episode tracking, move distribution analysis, ablation study metrics, etc. Conducted specialized corner-locking and fragmentation analyses to understand game-specific behavioral patterns. Wrote sections extended abstract, abstract, proposal changes, additional experiments, 5.2, 6, 7.

Changes from Proposal

- Replaced the originally proposed hybrid, MCTS-and-heuristic agent with purely end-to-end deep RL architectures, fundamentally shifting from a "bias-driven" approach incorporating domain knowledge to a "bias-free" approach testing general-purpose methods (DQN, PPO, QR-DQN, and Horizon-DQN) without any domain-specific priors.
- Introduced distributional return modeling (QR-DQN) and multi-step, double-Q targets with a recurrent LSTM encoder (Horizon-DQN), rather than planning-based search or handcrafted shaping heuristics.
- Added systematic hyperparameter optimization using Optuna (20 trials over 500-episode previews) to tune learning rate, discount factor, ϵ-decay, and PER parameters.
- Standardized the training regimen: all agents trained for 5 000 episodes, with H-DQN further trained for 9,000 episodes, and final evaluation over 1,000 deterministic rollouts logging average/max scores, tile distributions, and move frequencies.
- Expanded the analysis to include move-bias metrics (corner-locking rates), board fragmentation statistics, and training-curve diagnostics, which were not detailed in the original proposal.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[4] Matthieu Komorowski, Leo Anthony Celi, Omar Badawi, Anthony C Gordon, and A Aldo Faisal. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature Medicine*, 24(11):1716–1720, 2018.

[5] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

[6] B Ravi Kiran, Ibraheem Sobh, Vincent Talpaert, Patrick Mannion, Senthil Yogamani, Patrick Pérez, Sakthi Anayasamy, Santhosh Subramanian, Magesh Krishnan, et al. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[8] Jose A Arjona-Medina, Martin Gillhofer, Michael Widrich, and Jürgen Schmidhuber. A survey of reinforcement learning with function approximation. *ACM Computing Surveys (CSUR)*, 54(6):1–36, 2021.

[9] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):346–353, 2014.

[10] Chih-Kuan Hung, Marc G Bellemare, James Quan, Pierre-Luc Noury, Steven Chan, Yuhuai Wang, Timothy Lillicrap, and John Aslanides. Optimistic temporal difference learning for 2048: Towards solving the game. In *International Conference on Learning Representations (ICLR)*, 2022.

[11] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[12] Matteo Hessel et al. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[13] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[15] Richard S Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, volume 3, pages 9–44, 1988.

[16] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, et al. Noisy networks for exploration. In *International Conference on Learning Representations*, 2018.

[17] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *International Conference on Machine Learning*, pages 449–458, 2017.

[18] Lucas Willems. gymnasium-2048: 2048 game environment for reinforcement learning. `https://github.com/lcswillems/gym-2048`, 2023. Accessed: 2025-06-08.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[20] Satinder P Singh. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 1996.

[21] John Schulman et al. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[22] Max Jaderberg et al. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

[23] Marcin Andrychowicz et al. Hindsight experience replay. In *NeurIPS*, 2017.

[24] William Fedus et al. Revisiting fundamentals of experience replay. In *ICML*, 2020.

[25] Adrien Ecoffet et al. First return, then explore. *Nature*, 2021.

[26] Deepak Pathak et al. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[27] Yuri Burda et al. Exploration by random network distillation. In *ICLR*, 2019.

[28] Ian Osband et al. Deep exploration via bootstrapped dqn. In *NeurIPS*, 2016.

[29] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[30] Will Dabney et al. Implicit quantile networks for distributional reinforcement learning. In *ICML*, 2018.

[31] Steven Kapturowski et al. Recurrent experience replay in distributed reinforcement learning. In *ICLR*, 2019.

[32] Tom Schaul et al. Prioritized experience replay. In *ICLR*, 2016.

[33] Wojciech Jaśkowski. N-tuple bandit evolutionary algorithm for game 2048. In *GECCO*, 2015.

[34] Andreas Fuchs. Learning to play 2048 with deep reinforcement learning. In *Technical Report*, 2016.

[35] Tristan Cazenave. Nested monte carlo search for 2048. In *IEEE CIG*, 2015.

[36] Xiaoxiao Guo et al. Deep learning for real-time atari game play using offline monte carlo tree search planning. In *NeurIPS*, 2014.

[37] Volodymyr Mnih, Adrià Puigdomenech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Tim Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, pages 1928–1937, 2016.

[38] Lasse Espeholt et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[39] Sebastian Racaniere et al. Imagination-augmented agents for deep reinforcement learning. In *NeurIPS*, 2017.

[40] Arthur Guez et al. An investigation of model-free planning. In *ICML*, 2019.

[41] Ben Kantor et al. Generalization in sokoban via deep reinforcement learning. In *ICLR*, 2023.

[42] Ruijia Wang, Michelle Liu, Adam Stooke, Nicolas Heess, and David Silver. Critic regularized regression. In *ICML*, pages 7709–7719, 2020.

[43] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870, 2018.

[44] Aurélien Vieillard et al. Munchausen reinforcement learning. In *NeurIPS*, 2020.

[45] Mark Rowland et al. An adaptive agent for mixed stochastic outcomes. In *ICML*, 2018.

[46] Marc Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. In *IJCAI*, 2013.

[47] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[48] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. ACM, 2019.

[49] Hado Van Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[50] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. In *ACM SIGART Bulletin*, volume 2, pages 160–163, 1991.

[51] Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[52] Zihao Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

[53] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

[54] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Marian Czarnecki, David Silver, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

[55] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. In *arXiv preprint arXiv:1701.08734*, 2017.

[56] David Fridovich-Keil et al. Madrona engine. `https://madrona-engine.github.io/`, 2024. Accessed 2025-06-09.

A Additional Experiments

A.1 Dueling–PER–Symmetry–nStep Variant

We augmented the baseline DQN with dueling network streams, prioritized experience replay (PER), symmetry-based data augmentation, and multi-step returns. Table 5 details the agent architecture and training hyperparameters, while Table 6 reports performance.

Table 5: Agent configuration and training hyperparameters

Component	**Specification**
Dueling streams	Separate value and advantage heads
Prioritized Experience Replay	$\alpha = 0.6$, $\beta_0 = 0.3$
Symmetry augmentation	Eightfold rotations and reflections
Multi-step returns	3-step TD targets
Learning rate & ϵ-range	Same as baseline agents
Batch size	128
Replay buffer	100 000 transitions
Warm-up steps	10 000
ϵ decay	Factor of 0.9995 per step

Table 6: Evaluation results

Metric	**Value**
Average score	2 533.7
Maximum score	6 212
Highest tile reached	512
Moves “Up”	16 287
Moves “Right”	5 810
Moves “Left”	1 140
Moves “Down”	340

Despite outperforming traditional DQN in both average and peak scores, this variant consistently capped at the 512 tile under the evaluation. The pronounced corner-locking bias (few “Down” moves) aligns with optimal play patterns, suggesting that extended training or further algorithmic refinements were needed to surpass the 512 barrier.