Software Engineering for Artificial Intelligence

Reinforcement Learning Sustainability Benchmark

Report v1.0

Luca Strefezza*

February 13, 2025

*l.strefezza1@studenti.unisa.it

Abstract

This project explores the energy consumption of deep reinforcement learning (DRL) algorithms and their impact on the environment and business costs. Beginning with the development of Deep Q-Networks (DQN) by DeepMind, numerous algorithms have been proposed to enhance the performance of DRL agents. However, the energy implications of these improvements have not been extensively studied. This project aims to fill this gap by benchmarking the energy consumption and performance of several widely used DRL algorithms.

We train various reinforcement learning algorithms on the same task: Atari 100k, a widely recognized benchmark in the DRL community. The selected algorithms include both value-based methods (such as DQN and RAINBOW) and policy gradient methods (such as REINFORCE with baseline and PPO). Each algorithm is evaluated on 8 different Atari games, with 4 runs per game using different random seeds, to ensure statistically significant results.

To track performance and energy consumption, we utilize Weights and Biases, TensorBoard, and CodeCarbon. The development environment is based on CleanRL, a library that provides single-file implementations of some DRL algorithms, ensuring consistency and reproducibility.

The preliminary version of this report covers the context, goals, and methodological steps of the project. Future sections will present detailed results and analyses of the trade-offs between performance and energy consumption, contributing to a broader understanding of the sustainability implications of DRL technologies.

Contents

1 Context **4**

2 Goals **4**

3 Methodological Steps **5**

3.1 Algorithms Selection 5

3.1.1 Value-Based Methods 5

3.1.2 Policy Gradient Methods 6

3.2 Task Selection 7

3.3 Experiment Setup 7

3.3.1 Number of Runs 7

3.3.2 Data Logging and Storage 9

3.3.3 Development and Execution Environment 10

3.3.4 Atari Environment Configuration 11

3.3.5 (Hyper)Parameter Configurations 13

3.3.6 Evaluation 14

3.4 Data Analysis and Visualization 15

3.4.1 Log Collection and Merging 15

3.4.2 Normalization of Returns 15

3.4.3 Interpolation and Aggregation 16

3.4.4 Plot Generation and CSV Output 16

4 Preliminary Results and Findings **17**

4.1 Experiment Setup Adjustments 17

4.2 DQN-Based Algorithms 18

4.2.1 Deep Q-Network (DQN Baseline) 18

4.2.2 Double DQN 24

4.2.3 Prioritized Experience Replay 31

4.2.4 Dueling DQN 37

4.2.5 Categorical DQN (C51) 43

4.3 Overall Comparison of DQN-Based Algorithms 49

4.4 Policy-Based Algorithms 55

4.4.1 REINFORCE 55

4.4.2 PPO (Proximal Policy Optimization) 55

4.4.3 SAC (Soft Actor-Critic) 55

4.5 Overall Comparison of Policy-Based Algorithms 56

4.6 Cross-Category Comparison: DQN vs. Policy Gradient 56

5 Implications of the Results **56**

5.1 General Observations 56

5.2 Energy Efficiency vs. Performance Trade-Off 57

5.3 Practical Implications for AI Sustainability 57

5.4 Limitations and Future Work 58

6 Conclusions **58**
6.1 Summary of Findings 58
6.2 Final Thoughts on Energy-Efficient Reinforcement Learning 59
6.3 Future Research Directions 59

References **59**

1 Context

This project addresses the energy consumption of deep reinforcement learning (DRL) solutions and their impact on the environment and business costs.

Beginning with the resurgence of the field following the development of *Deep Q-Networks* (DQN) by DeepMind in the early 2010s [1][2], there have been a number of algorithm proposals over time that with minor modifications to DQN or using a completely different paradigm (such as policy gradient methods) sought to improve the performance achieved by the learning agent.

Although the performances of the various solutions have been extensively studied and tracked, little effort has been directed toward understanding how the tweaks to the DQN introduced to improve performance impacted energy consumption, or what the cost of the alternative approaches developed was, per se and in comparison with previous solutions.

The motivation behind this project is to fill this gap by evaluating the trade-offs between performance and energy consumption for several widely used deep reinforcement learning (DRL) algorithms. Understanding these trade-offs is crucial for businesses and researchers who aim to optimize both performance and sustainability in their applications. This project aims to provide valuable insights into the energy efficiency of different DRL approaches, enabling informed decisions about their use in various contexts.

To reach this goal we train various reinforcement learning algorithms on the same task, the choice of which is discussed in section 3.2 on page 7. Section 3.1 on the next page describes the selected algorithms, whose choice was made taking into account that DRL algorithms can be devided in two main categories: *value based* (i.e. algorithms based on the approximation of a value function, be it the state-value function or the action-value function) and *policy gradient*. The latter are methods that approximate directly the policy, and includes as a special case the *actor-critic methods*, which approximate simultaneously a policy (said actor) and a value function (said critic).

2 Goals

The primary goal of this project is to benchmark the energy consumption and performance of various deep reinforcement learning algorithms. Specifically, we aim to:

1. evaluate the energy consumption of different DRL algorithms when trained on the same task;

2. compare the performance of these algorithms in terms of their ability to achieve high scores on the given task;
3. analyze the trade-offs between performance and energy consumption to identify the most efficient algorithms;
4. provide a comprehensive report that can guide practitioners in selecting the appropriate DRL algorithms based on specific use-case requirements.

By achieving these goals, the project will contribute to the broader understanding of the sustainability implications of deep reinforcement learning technologies.

3 Methodological Steps

The methodology used follows from the basic idea of this benchmark: to execute all the algorithms for the same number of environment interactions, so that we can compare the score they achieve and the energy consumption of each one of them. Additionally, a good comparison would be to take the score obtained by the lowest performer in this initial trial and re-train all the algorithms until they reach that score. This would allow us to compare how much time and energy each algorithm requires to achieve the same performance level. Unfortunately, time and resources constraints make retraining all algorithms unfeasible, so we will approximate this second comparison by using the returns from the logging of the training during the first trial. This logging includes the `global_step`, indicating the environment interaction we are at, and the `episodic_return`, which is the return of the episode (i.e., the score on which to compare), as well as all performance and power consumption data up to that point. By analyzing these logs, we will estimate how much time and energy each algorithm would take to reach the score obtained by the lowest performer in the initial trial.

The following sections outline the several key steps involved in the methodology adopted for this project.

3.1 Algorithms Selection

As stated, in our benchmark we consider both value-based methods and policy gradient methods. The selected algorithms are chosen to represent a wide range of approaches within both categories.

3.1.1 Value-Based Methods

Value-based methods are algorithms based on the approximation of a value function. The following algorithms were considered in this category (but due to time and computational constraints, only a subset of 5 of them were fully trained and evaluated):

- *Deep Q-Network (DQN)*: the first example of success in deep reinforcement learning, will serve as a sort of baseline for our benchmark.

- *RAINBOW* [3]: an advanced method that combines several improvements to the original DQN, that will also be tested individually to assess their individual contributions to energy consumption and performance. These are listed hereafter:
  - Double Q-Learning (Double DQN) [4];
  - Prioritized Experience Replay [5];
  - Dueling Network Architectures [6];
  - Multi-step / N-step Learning [7];
  - Distributional RL [8];
  - Noisy Nets [9];
- *Self-Predictive Representations (SPR)* [10]: a more advanced method introduced in recent research, which leverages self-predictive representations to improve efficiency.

3.1.2 Policy Gradient Methods

Policy gradient methods approximate the policy directly and include as a special case the actor-critic methods, which simultaneously approximate a policy and a value function. The algorithms considered in this category are:

- *REINFORCE* [11, Chapter 13]: a basic policy gradient method, or its variant REINFORCE with baseline (also known as Vanilla Policy Gradient, VPG).
- *Proximal Policy Optimization (PPO)* [12]: a popular and efficient policy gradient method that uses a clipped objective to improve training stability.
- *Deep Deterministic Policy Gradient (DDPG)* [13]: an algorithm that combines policy gradients with deterministic policy updates for continuous action spaces.
- *Twin Delayed DDPG (TD3)* [14]: an improvement over DDPG that addresses function approximation errors through various techniques, such as delayed policy updates and target policy smoothing.
- *Soft Actor-Critic (SAC)* [15]: an extension of DDPG that incorporates entropy regularization to encourage exploration. SAC, like TD3, uses two Q-networks to reduce overestimation bias, but it differs by optimizing a stochastic policy instead of a deterministic one. This makes SAC more sample-efficient and stable in continuous control tasks. It is also more easily adapted to discrete action spaces.
- *Data-Regularized Q (DRQ)* [16]: a method that incorporates data augmentation to regularize the training of Q functions, improving performance and stability.

3.2 Task Selection

Regarding the task on which to compare the algorithms, there were several suitable candidates: Atari 100k [17], one of the continuous control task of the DeepMind Control Suite, or one of the many other task (besides Atari) included in OpenAI Gymnasium (formerly Gym), and so on. After various tests and research we opted for the Atari 100k benchmark, a discrete task that consists of playing selected Atari games for only 100 000 environment interactions.

The reason for this choice is multifaceted. Atari 100k is a widely used benchmark in the DRL community, the wealth of prior research and baseline results available facilitates a more straightforward validation and comparison of our experimental results with those from other studies and algorithms. It is also well suited for evaluating the performance of almost all popular DRL algorithms, ensuring a comprehensive assessment. Additionally, Atari games provide a range of different challenges, including planning, reaction time, and strategy, making it a robust benchmark for assessing general DRL capabilities.

Moreover, the discrete nature of Atari 100k simplifies the implementation and comparison of algorithms, as continuous control tasks often require additional considerations and modifications. Finally, the 100 000 interactions limit strikes a balance between providing enough data for meaningful evaluation and being computationally feasible within our resource constraints, especially considering the large number of experiments required for each algorithm, as detailed in section 3.3.1.

These factors combined make Atari 100k a practical and effective choice for our benchmark, enabling us to achieve our project goals efficiently.

3.3 Experiment Setup

In this section we will address all the decisions made in the setup of the experiments.

3.3.1 Number of Runs

In determining how many runs to carry out during the experimentation and testing of a reinforcement learning algorithm, at least two fundamental aspects must be taken into account: the high variance of reinforcement learning, and thus its high susceptibility to randomness, and the evaluation of the generality of the algorithm, which must therefore be tested in several different environments in order to actually prove that it is capable of solving multiple problems and not just be ultra-specialized on a single use-case.

In addressing the first aspect we can refer to the literature to get an idea of how many runs with different seeds are usually performed to alleviate this problem. If in the early days of RL (and not DRL) the number of runs stood at around 100 and in any case did not fall below 30, at least until the introduction of ALE (Arcade Learning Environment) [18] included, with the advent of DRL the number of runs was consistently reduced to 5 or less because of the high cost in terms of time and resources per run. Although this has been the standard for years, a more recent work [19] has shown that this is the source of a problem. Practitioners use point estimates such as mean and median to aggregate performances scores across tasks to summarize the results of the various runs, but this metrics are not the best way to do so because they ignore the statistical uncertainty inherent in performing only a few runs.

In particular, the study points out that in the case of Atari at least 100 runs per environment are required to obtain robust results, a value that is, however, impractical in reality. To address this, the study recommends using alternative aggregation metrics, such as interquartile means, designed precisely to obtain more efficient and robust estimates and have small uncertainty even with a handful of runs, since they are not overly affected by outliers like the point estimates.

In our case we will be forced to limit ourselves to 4 runs per environment, so we will use, in addition to the more classic and popular metrics such as the point estimates mentioned above, the other metrics suggested in [19]. It should anyway be noted that a low number of runs is a less significant problem for us, since we are not attempting to advance the state of the art performance of DRL algorithms, but have instead a focus on energy consumption and emissions, which should in any case remain constant regardless of the actual learning of the agent, which is instead related to randomness.

With regard to the second aspect, namely, testing the algorithms on a variety of environments to evaluate their generality, Atari 100k once again comes to our aid, being constituted by 26 games. Moreover, the Arcade Learning Environment, built on top of the Atari 2600 emulator Stella and used by gymnasium, includes over 55 games. Unfortunately, again, we do not have the time and/or computational resources to test on all the Atari 100k's 26 games or all the ones available in ALE, so we selected for the benchmark a representative subset of 8 Atari games, trying to choose games that cover a range of difficulties and styles. Obviously, with so few games because of the constraints just mentioned, an exhaustive selection is difficult, but we nonetheless tried to provide a balanced benchmark, ensuring that the selected games cover a range of challenges to effectively evaluate different algorithms, while still not being excessively difficult. This last requirement is due to basic DQN and its more simple extensions, which have some limitations in only 100k interactions (the team that introduced the DQNs trained its model on millions of interactions to achieve interesting results).

Here are the 8 selected games, along with a rationale for their inclusion:

- *Alien* - involves exploration and strategic movement;
- *Amidar* - requires precise movement, quick decision-making and long-term planning;
- *Assault* - a fast-paced shooter testing reflexes and targeting accuracy;
- *Boxing* - visually simple yet requires precise timing and positioning;
- *Breakout* - a control-based game widely studied in RL;
- *Freeway* - simple ruleset, tests quick decision-making and reaction time;
- *Ms. Pac-Man* - emphasizes navigation, evasion, and planning;
- *Pong* - minimalistic, simple and well-understood game used as an RL baseline.

Although Alien and Ms. Pac-Man may appear similar in terms of overall theme, we decided to keep both in our selection due to their differing action space structures. Alien has a more complex movement and shooting action space, while Ms. Pac-Man involves navigation-based control with a different interaction model. Including both allows us to evaluate how reinforcement learning algorithms adapt to environments with distinct control dynamics, rather than just variations in visual style or game mechanics.

So, to summarize, each algorithm will be evaluated on 8 different Atari games, with 4 runs per game using different random seeds, for a total of 32 trainings per algorithm. This approach, with appropriate metrics, ensures that our results are statistically significant and account for the inherent variability in RL training processes.

3.3.2 Data Logging and Storage

Collecting comprehensive and accurate data is crucial for evaluating both the performance and energy consumption of the algorithms. We employ several tools and services to ensure robust data collection and analysis.

To track the performance metrics, we use both online and local tools. The online service *Weights and Biases* (wandb) is used for real-time monitoring and storage of experimental data. This platform allows for easy sharing and collaboration, as well as providing powerful visualization and analysis tools. Locally, we use *TensorBoard*, which integrates seamlessly with our training workflows and offers detailed insights into the training process through its rich set of visualizations.

In addition to tracking performance metrics, monitoring energy consumption and emissions is the key aspect of the project. For this we use *CodeCarbon*, a tool designed to measure the carbon footprint of computing activities. As stated in their documentation, this package enables developers to track emissions, measured as kilograms of CO_2-equivalents (CO_2eq) in order to estimate the carbon footprint of their work. CO_2-eq is a standardized measure used to express the global warming potential of various greenhouse gases: the amount of CO_2 that would have the equivalent global warming impact. For computing, which emits CO_2 via the electricity it is consuming, carbon emissions are measured in kilograms of CO_2-equivalent per kilowatt-hour [20]. See this page and section 4.1 for more information on their methodology.

Explained the tools, the metrics we collect through them include:

- *Global Step:* indicates the number of environment interactions during training.
- *Episodic Return:* the score achieved in each episode, providing a measure of the algorithm's performance.
- *Loss(es):* track the optimization process, giving insight into the learning dynamics of the algorithm.
- *Value Estimates:* such as Q-values or value function estimates, offering insight into the agent's decision-making process.

- *Policy Entropy:* measures the randomness in the policy and how much it differs from the previous one, useful for understanding exploration behavior and how much room for improvement is still left.
- *Learning Rate:* the rate at which the model learns, especially if it changes during training.
- *Emissions:* the amount of CO_2-eq emitted during training, tracked by CodeCarbon.

Weights and Biases facilitates a coarse aggregation and visualization of these metrics across multiple runs and environments, making it easier to compare results at a first glance and draw some first insights. TensorBoard provide supplementary local visualizations to help diagnose any issues during training and ensure the integrity of the collected data.

By using these tools in tandem, we aim to collect a comprehensive dataset that covers both the performance and energy consumption aspects of the algorithms, ensuring a thorough evaluation aligned with the goals of our project.

3.3.3 Development and Execution Environment

The development and execution environment for the project involves both hardware and software. In particular, we have made use of two different hardware setups due to constraints in energy tracking capabilities.

Initially, all configurations and the first fine-tuning of DQN and some other algorithms were performed on a machine with:

- **CPU**: 11th Gen Intel(R) Core(TM) i5-11400F @ 2.60GHz
- **GPU**: NVIDIA GeForce GTX 1050 Ti
- **RAM**: 16GB

However, due to CodeCarbon's lack of support for the GTX 1050 Ti in tracking GPU energy consumption, the main training experiments had to be conducted on a different machine with higher computational power and proper energy tracking support. The second setup consisted of:

- **CPU**: Intel(R) Core(TM) i9-10980XE @ 3.00GHz
- **GPU**: NVIDIA RTX A5000
- **RAM**: 64GB

While the first machine was sufficient for setting up the environment and running initial fine-tuning, this switch to the A5000 GPU in the second setup was necessary to ensure full compatibility with CodeCarbon and reliable, accurate measurement of energy consumption and carbon emissions during experimentation.

On the software side, after careful considerations and some testing with other alternatives like OpenAI's *Spinning Up*, we chose to base the implementation of the project on

CleanRL [21]. As the authors states, CleanRL is an open-source library that provides high-quality single-file implementations of Deep Reinforcement Learning algorithms. It provides an environment already complete with most dependencies a project like ours might need (like Gymnasium), has a straightforward codebase, and already integrates tools like Weights and Biases and TensorBoard, that help log metrics, hyperparameters, videos of an agent's gameplay, dependencies, and more.

The single-file implementation philosophy of CleanRL aims to make reinforcement learning research more accessible and reproducible and make the performance-relevant details easier to recognize. By consolidating every algorithm codebase into single files, it simplifies the understanding and modification of algorithms, which is particularly beneficial for both educational purposes and rapid prototyping, even though it comes at the cost of losing modularity and duplicating some code.

We leverage CleanRL's existing implementations where available, tweaking them to meet the specific requirements of our benchmarks. When an implementation for a particular algorithm is not available, we develop it from scratch, trying to adhere to CleanRL's philosophy and implementation principles. This approach ensures consistency and comparability across all tested algorithms.

In the end, the environment for our experiments should be efficient and easily reproducible, facilitating the accurate evaluation of both performance and energy consumption of various deep reinforcement learning algorithms.

3.3.4 Atari Environment Configuration

The Atari environment setup follows best practices outlined in [22] for training and evaluating agents in the Arcade Learning Environment (ALE). These decisions were made to ensure a standardized, reproducible, and robust experimental setting. Additionally, we incorporate relevant insights from the ALE documentation to refine our environment configuration. Many of these choices also align with those made in the first works on Deep Q-Networks (DQN), ensuring comparability with early research efforts.

Preprocessing and Standardization The preprocessing pipeline ensures consistent input representations across different Atari games, avoiding confounding factors that could skew results. Through the use of appropriate atari wrappers of Stable Baselines v3, the following steps are implemented:

Frame skipping: we use `MaxAndSkipEnv(skip=4)`, ensuring that actions are repeated for four frames and the maximum pixel values of consecutive frames are used. This stabilizes the input representation and allows agents to process meaningful changes in the game environment while reducing computational load.

Random no-op initialization: at the beginning of each episode, a random number (up to 30) of "do nothing" actions are executed (`NoopResetEnv(noop_max=30)`). This prevents deterministic policies from exploiting fixed starting conditions, improving generalization.

Episodic life and fire reset:

`EpisodicLifeEnv`: is used to reset the environment after each lost life instead of at the end of the full game. This makes training more efficient by exposing the agent to more starting states per episode.

`FireResetEnv`: is applied in games where a "FIRE" action is required to start (e.g., Breakout), ensuring proper initialization.

Observation preprocessing:

- raw RGB images are converted to grayscale and resized to 84×84 pixels (`GrayScaleObservation` and `ResizeObservation`).
- a history of the last four frames is stacked (`FrameStack(4)`) to provide temporal context, compensating for the lack of explicit memory.

Reward clipping: rewards are clipped between -1 and 1 (`ClipRewardEnv`) to standardize their scale across different games. This makes the algorithms able to work with all games without needing refinements to adapt to particularly high- or low-reward games, while stabilizing training.

Choice of the Environments Version The Arcade Learning Environment (ALE) [18] provides multiple versions of Atari environments [23] to address different research requirements and needs. These environments versions encapsulate the preprocessing steps we talked about, each one setting a different default value for them and some other aspects of the games. Two of the most widely used versions are *NoFrameskip-v4* and *v5*. The main distinction between these is the inclusion of *sticky actions* in *v5*, as recommended in [22]. Sticky actions introduce a 25% probability of repeating the previous action, adding stochasticity to the environment to prevent overfitting when training deterministic policies.

In this study, we use the *NoFrameskip-v4* environments [23]. This choice is motivated by several factors. First, *NoFrameskip-v4* ensures fully deterministic execution when a fixed random seed is used, which is crucial for the reproducibility of our experiments. This determinism allows us to conduct controlled comparisons of different algorithms while minimizing the influence of environment stochasticity on performance evaluation. Additionally, since our preprocessing pipeline explicitly applies `MaxAndSkipEnv(skip=4)` to handle frame skipping in a standardized way (as discussed in the previous section), the built-in frame skipping behavior of other environment versions is unnecessary and would introduce redundant processing.

The primary difference between our setup and the *v5* environments is the exclusion of sticky actions. While sticky actions can enhance generalization in long training regimes by preventing the agent from overfitting to deterministic game mechanics, their benefits are less relevant in our setting, where each run is limited to only 100k interactions. Under such a short training horizon, the additional stochasticity introduced by sticky actions would significantly degrade training stability and learning efficiency, leading to noisier performance estimates thus removing them is not only not problematic, but almost mandatory. Furthermore, the use of sticky actions is not as ubiquitous in the reinforcement learning literature as other preprocessing steps, making their exclusion a reasonable choice also for comparability with prior work.

By structuring our preprocessing pipeline around the NoFrameskip-v4 environments and following the best practices from [22], we ensure that our experimental results are robust, reproducible, and comparable to the large body of prior deep reinforcement learning research. The preprocessing steps applied in our implementation are widely used in reinforcement learning studies and enable fair performance evaluations across different Atari games. Furthermore, the decision to exclude sticky actions aligns with the constraints of our 100k iteration limit, ensuring meaningful training without excessive randomness hindering learning progress.

Feature	**NoFrameskip-v4 (Our Setup)**
Frame Skipping	Explicitly set via `MaxAndSkipEnv(skip=4)`
No-op Start	`NoopResetEnv(noop_max=30)`
Episodic Life	`EpisodicLifeEnv`
Fire Reset	`FireResetEnv` (if needed)
Observation Preprocessing	Grayscale + Resize (84x84) + FrameStack(4)
Reward Clipping	`ClipRewardEnv` (-1, 1)
Sticky Actions (`repeat_action_probability`)	Not Used (Fixed Action Selection)

Table 1: Comparison between our setup based on NoFrameskip-v4 and ALE v5 environments.

3.3.5 (Hyper)Parameter Configurations

We discuss in this section a set of parameters that influence the experiment setup but not directly the optimization process like hyperparameters do. Regarding the latter, we do discuss here our general approach in their initial setting and optimization, but we delay to section 4, in which we dedicate a section to every algorithm, the details regarding their fine tuning, so to have a more cohesive and complete presentation.

Parameters

Tracking and Logging: the flag `--track` ensures that training metrics are logged in `wandb`. The project name is set through the flag `--wandb-project-name` (in our case `rlsb`). The tracking in tensorboard is always enabled.

Device Usage: `--cuda` enables training on GPU, if this option is available.

Random Seed: the `--seed` value to use for this run.

Video Capture: the `--capture-video` flag is used to record the gameplay of the agent. We set it to `False`, indicating that video recording of agent behavior is not performed during training, but we enable it during evaluation.

Model Saving: the `--save-model` flag is set to `True`, this also automatically starts the evaluation process right after the model is saved.

Hyperparameters The hyperparameter selection for all implemented algorithms was primarily based on the configurations used in the original papers introducing each method and, when available, those from the CleanRL implementation. A key consideration across all algorithms was the adaptation of the hyperparameters strictly connected to the number of environment interactions. Since deep reinforcement learning algorithms are typically trained for 5 to 10 million interactions, whereas our study was constrained to 100 000 interactions, certain hyperparameters, such as `learning_starts` and `buffer_size`, required adjustment to ensure appropriate behavior in this limited training setting.

For the baseline Deep Q-Network, the hyperparameters closely matched those from [2] and the CleanRL repository, as both sources used highly similar settings. The main focus of the tuning was the aforementioned adaptation to a reduced number of interactions with the environment.

For the various DQN variants tested, the hyperparameters were initialized using the same configuration employed for the base DQN implementation. The tuning process primarily involved modifying parameters directly associated with the respective architectural or algorithmic tweak while keeping the overall structure as consistent as possible with standard DQN. This approach aligns with the methodology commonly adopted in prior research introducing these modifications, ensuring a fair and controlled comparison. By maintaining a shared foundation across variants, we were able to isolate the impact of each specific enhancement in terms of performance improvements and emissions cost.

3.3.6 Evaluation

After completing the training phase, we evaluate the agent by executing 10 episodes in the target environment. During these episodes, we collect the *episodic returns*, which serve as the primary metric for performance assessment.

The evaluation of the DQN-based methods is conducted with a fully deterministic policy, setting $\varepsilon = 0.00$ to disable exploration, ensuring that the agent exploits its learned policy without stochasticity. This allows for a clear assessment of how well the trained model generalizes to unseen episodes. The same is true for SAC (see also the discussion in 4.4.3), while Reinforce and PPO, being on-policy algorithm that optimize a stochastic policy, are evaluated on it, meaning actions are sampled from the learned distribution.

While, to avoid interference with the training process, we disable video recording during it, we enable it during evaluation by setting `capture_video=True`. This provides visual insight into the agent's behavior without incurring the computational overhead during learning.

The collected episodic returns undergo statistical analysis following the methods described in Data Analysis and Visualization. Specifically:

- We apply normalization to the collected data, both **human normalization** and **min-max normalization**, as detailed in 3.4.2.

- Basic statistics are computed on the normalized data, including mean, standard deviation, and median.
- The **interquartile mean (IQM)** is used as a robust estimator, as suggested in prior research.

The evaluation script loads the trained model, initializes a synchronized evaluation environment, and runs the agent for the specified number of episodes. It follows the same preprocessing pipeline used in training, ensuring consistency in observation space and action execution.

3.4 Data Analysis and Visualization

A critical part of this project involved consolidating and analyzing the training logs in a consistent and reproducible manner. Although *Weights & Biases* (W&B) and *TensorBoard* can both display metrics across runs, they each have limitations for comparative analysis—particularly when plotting multiple algorithms or combining results with additional metadata (e.g., hyperparameters, emissions data). Consequently, a custom data-processing pipeline was built to generate unified plots and aggregated statistics.

3.4.1 Log Collection and Merging

We collected detailed logs for each run: TensorBoard event files (containing metrics such as episodic returns, steps per second, losses, etc.) and W&B logs, which contains all the data of the TensorBoard logs (extrapolated from the uploaded TensorBoard logs), plus some other system related metrics. To work on this data in Python with its scientific tools we employed the `tbparse` library to parse the TensorBoard logs, making modifications to the library where necessary to handle deprecated NumPy types. These parsed logs resulted in two CSV files, one with all the metrics for all the runs, the other containing additional information about hyperparameters (e.g., learning rate, buffer size, and so forth) from the experimental configuration. We then merged the two files into a single, larger CSV dataset containing all runs from all algorithms.

3.4.2 Normalization of Returns

Since the raw episodic returns for Atari games vary widely in scale, a fair and not skewd comparison needed a normalization step. This is one of the reasons that prevented us from directly using tensorboard and wandb plots. We performed two distinct normalization procedures:

- **Human-Normalized Returns.** In this approach, for each game, we subtract the score of a random agent and divide by the difference between the human baseline and the random baseline. This is a standard practice in Atari benchmarks to contextualize performance relative to human play.

- **Min-Max Normalization.** A classic min-max scaling ($i.e.$, $x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$) on a per-game basis, where $x_{\min}$ and $x_{\max}$ come from the observed range of returns for that game.

These normalized returns facilitate more intuitive cross-game comparisons, ensuring that no single game with unusually high or low rewards dominates the overall analysis. Since we don't care about comparing the performance of our implementations with the literature, and the relative comparison between the algorithms is the same with both normalizations, we can use indifferently either one of them based on which one produce a clearer plot.

3.4.3 Interpolation and Aggregation

When generating metric curves (such as episodic returns vs. training steps), we needed a consistent x-axis across runs. Many runs log metrics at slightly different steps (due to stochastic episode lengths, logging frequencies, etc.). The management of this aspect from both wandb and tensorboard is not ideal or lacking, not always allowing precise control or easy export of aggregated data.

In our pipeline, we therefore:

1. **Filtered by Metric and Algorithm.** We grouped rows in the CSV by a specific tag (e.g., `charts/episodic_return`, `charts/SPS`) and by algorithm.

2. **Interpolated to a Common Grid.** For each subset, we created a uniformly spaced array of steps (e.g., 1000 points from 0 to 100 000. We then applied linear interpolation on each run's time series to ensure all runs aligned on this common step axis.

3. **Computed Statistics.** At each point on the new, shared step grid, we aggregated the interpolated run values to produce statistics such as *mean*, *min–max range*, *standard deviation*, etc.

The interpolation ensures that every run contributes to the curves at the same discrete set of training steps, simplifying the generation of *mean* or *min–max* envelopes. This was crucial for plotting aggregate performance over multiple runs.

3.4.4 Plot Generation and CSV Output

Following the interpolation and aggregation process, the final step was to produce consistent plots for each metric–algorithm pair. We used `matplotlib` to generate both raster (PNG) and vector (SVG) graphics, giving us flexibility for both online presentation and printed material. Additionally, the aggregated statistics for each plot were saved as a separate CSV file, allowing subsequent combinations of multiple algorithms on a single plot without re-running the entire pipeline.

Overall, this approach provided:

- Fine-grained control over which metrics and runs to include;

- A robust method (interpolation) to align metrics across stochastic training steps;
- Easy export to consistent plots and CSVs for further analysis.

By integrating custom plotting and data analysis with the logs from W&B and TensorBoard, we ensure reproducibility and enable deeper insights into the trade-offs between performance and energy consumption across all tested algorithms.

4 Preliminary Results and Findings

This section presents the results obtained from training a subset of the algorithms discussed in section 3.1. The selected algorithms include five DQN-based methods — DQN, Double DQN, Prioritized Experience Replay, Dueling DQN, and C51 — as well as three policy-based methods: REINFORCE, PPO, and SAC. Soft Actor Critic was preferred to DDPG and TD3 because it can simply be seen as a variation that works with a stochastic policy, but is more easily adapted to a discrete action space.

We first describe necessary modifications to the experiment setup, followed by a detailed analysis of each algorithm's performance and emissions. The results are then compared across different algorithm families.

4.1 Experiment Setup Adjustments

During initial training attempts, some adjustments were required to ensure reliable performance and energy tracking. One key reason for this was that employing CodeCarbon as a (next-)real-time emissions tracking tool significantly slowed down training (by a factor of 20 or more). As a result, we opted to record only total emissions at the end of training rather than tracking them continously. This adjustment allowed us to obtain meaningful comparisons without excessively increasing training time.

In addition to this, on Windows, CodeCarbon's CPU energy tracking relies on the Intel Power Gadget, which has been deprecated for several years. Furthermore, it does not support Intel Performance Counter Monitor (Intel PCM), the official successor to the Power Gadget. In such cases, CodeCarbon switches to a fallback mode, directly quoting from their documentation:

> - It will first detect which CPU hardware is currently in use, and then map it to a data source listing 2000+ Intel and AMD CPUs and their corresponding thermal design powers (TDPs).
> - If the CPU is not found in the data source, a global constant will be applied. CodeCarbon assumes that 50% of the TDP will be the average power consumption to make this approximation.
> - We could not find any good resource showing statistical relationships between TDP and average power, so we empirically tested that 50% is a decent approximation.

This approach should provide reasonable estimates for our project, since most of the workload is on the GPU, while the rest is mostly constant across the algorithms (like the environment simulations). This being said, one instance where this limitation may have had an impact is in tracking the Proximal Policy Optimization (PPO) algorithm, that employs a relatively small neural network but requires more CPU and RAM processing, the latter also explicitly stated to not be tracked satisfactorily.

Moreover, wandb collect systems data, but it also, while collecting kwh for the gpu, does not for cpu and ram, so we have to make do with what we have

Additionally, Weights & Biases collects system data during training, and while it tracks GPU energy consumption in kWh, it also does not do the same for CPU and RAM. As a result, while we can obtain excellent emissions estimates for the GPU, CPU and RAM energy tracking remains imprecise due to the aforementioned limitations. Consequently, energy consumption analyses must be interpreted with an understanding of these constraints.

4.2 DQN-Based Algorithms

We present the results for the five different DQN-based algorithms. Each algorithm is analyzed individually before an overall comparison.

4.2.1 Deep Q-Network (DQN Baseline)

(Hyper)Parameters Table 2 summarizes the main hyperparameters used for our DQN baseline. `env_id` and `seed` varied across runs (eight Atari games $\times$ four seeds), while the rest remained unchanged. Note in particular that `buffer_size` and `learning_starts` have been reduced relative to their usual millions-step values to accommodate the shorter 100k-step regime.

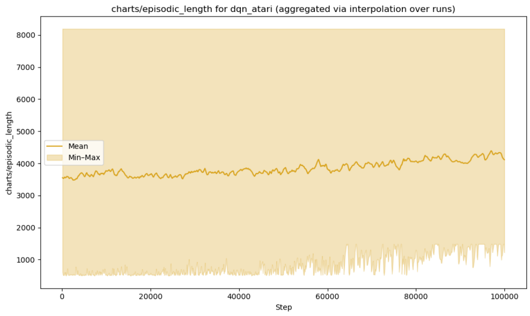
Hyperparameter Tuning We began with the *CleanRL* defaults (similar to Mnih et al.'s original DQN [1]) and scaled down parameters tied to a large number of environment interactions. For instance, `buffer_size` was tested at {10k, 20k}, and `learning_starts` at {800, 1000, 2000, 5000}. Empirically, a buffer of 10k and `learning_starts` of 1000 provided the best trade-off between stability and performance in the 100k-step setting. We kep τ equal to 1 for consistency with the original work. This parameter regulates the update of the target network weights by controlling the interpolation between those of the current Q-network and target network, following the update rule:

$$\vartheta_{\text{target}} = \tau\vartheta + (1 - \tau)\vartheta_{\text{target}}$$

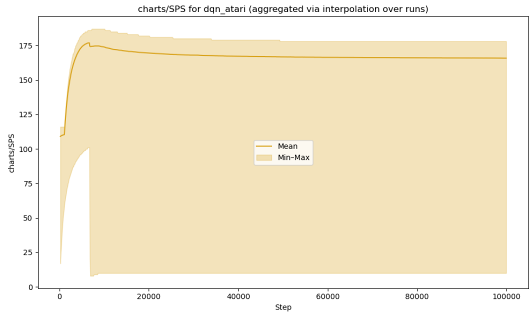
where $\vartheta_{\text{target}}$ are the weights of the target network and ϑ are the ones of the q-network. For $\tau = 1$, the target network is completely overwritten by the Q-network every time it's updated, as done in the original DQN.

Table 2: Key hyperparameters for the DQN baseline. Only env_id and seed change across runs.

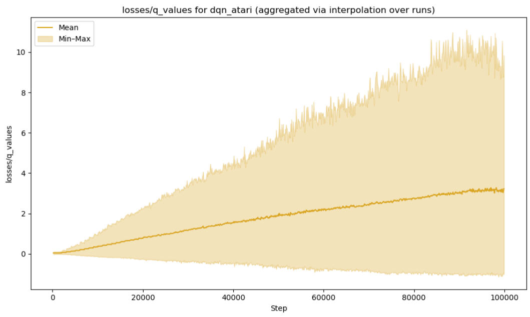
Parameter	Value
exp_name	dqn_atari
seed	1..4
torch_deterministic	True
cuda	True
track	True
wandb_project_name	rlsb
capture_video	False
save_model	True
upload_model	False
env_id	e.g. AlienNoFrameskip-v4
total_timesteps	100000
learning_rate	0.0001
num_envs	1
buffer_size	10000
gamma	0.99
tau	1.0
target_network_frequency	1000
batch_size	32
start_e, end_e	1.0 $\rightarrow$ 0.01
exploration_fraction	0.1
learning_starts	1000
train_frequency	4



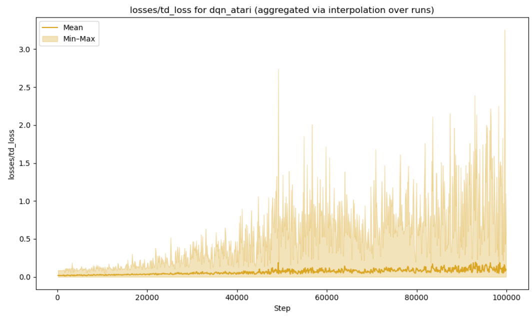
(a) *Aggregated episodic length for DQN over 100k steps (interpolation across 32 runs). The mean hovers around 3500–4000 steps, while the min–max envelope extends from near 0 to over 8000.*



(b) *Steps per second (SPS) for DQN. After an initial ramp-up, the mean SPS stabilizes around 170–180, with some runs dipping as low as 20 or spiking above 200.*



(c) *Estimated Q-values (`losses/q_values`) for DQN (aggregated over 32 runs). The mean Q-value climbs from near 0 up to ∼4–5, while some runs exceed 10.*



(d) *TD loss (`losses/td_loss`) for DQN. Losses grow with training, reaching above 3.0 in some runs, reflecting substantial variance near the final stages.*

Figure 1: Performance metrics for DQN over 100k steps, aggregated across 32 runs.

Training Dynamics (Aggregated Over 32 Runs) Figure 1a shows that episodes usually last in the 3000–4000 step range, but certain runs or environments have early terminations (very short episodes) or extremely long ones (around 8000 steps). Figure 1b indicates that, computationally, training stabilizes at a solid ∼170 steps per second on average, though hardware and environment differences introduce notable variance.

Q-Values and TD Loss Figures 1c and 1d show `losses/q_values` and `losses/td_loss`, respectively, across all runs. On average, Q-values increase steadily, suggesting the network's estimates of future returns keep growing with experience. However, the broad min–max band indicates some seeds or games diverge or plateau differently. The TD loss remains small in early training but spikes in certain runs, possibly due to volatile updates from the replay buffer once it's partially filled.

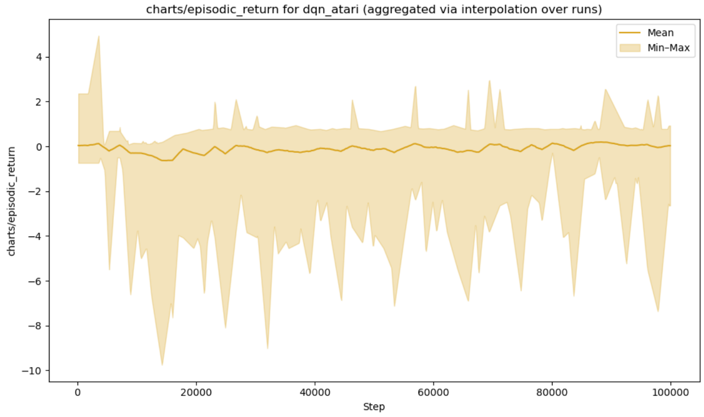


Figure 2: Aggregated DQN episodic return (human-normalized) over 100k steps. The shaded region represents min–max variation.

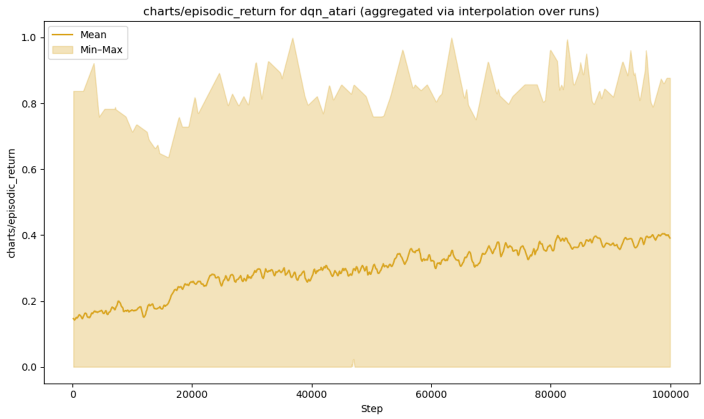


Figure 3: Aggregated DQN episodic return (min–max normalized).

Episodic Return (Human vs. Min–Max Normalized) We analyzed the collected episodic returns applying both the human normalization and min–max normalization schemes, as explained in section 3.4.2 on page 15. Figures 2 and 3 aggregate these returns across all 32 runs, while Figures 4 and 5 show per-game curves.

In the human-normalized plot, the mean hovers near zero, occasionally dipping negative due to poor performance on certain games. In the min–max plot, the average climbs from near 0.2 to around 0.4–0.5 by the end, indicating moderate relative progress.

Different environments see dramatically different results: *Freeway* often approaches high normalized scores, while *Pong* and *MsPacman* remain relatively low (especially in the human-normalized scale).

Emissions Table 3 presents the aggregated CO_2-eq for DQN (over all 32 runs). The mean is about **0.00647 kg**, with a minimum of 0.00616 and a maximum near 0.0070.

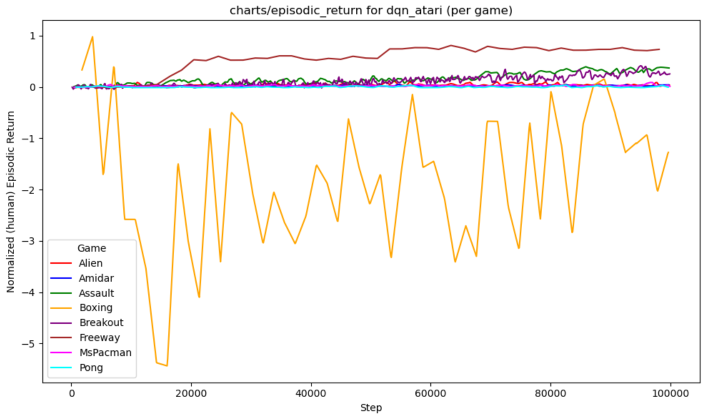


Figure 4: DQN returns (human-normalized) by game. Each line aggregates four seeds for that specific environment.

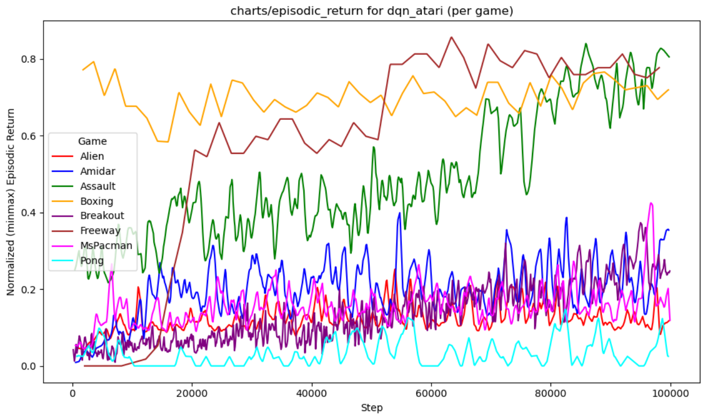


Figure 5: DQN returns (min–max normalized) by game.

Table 3: Carbon emissions (kg CO_2eq) for DQN across 32 runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmean
DQN	0.006469	0.0002609	0.006342	0.006296	0.006578	0.006162	0.006997	0.006369

Table 4: Overall final evaluation (10 episodes each) for DQN across all runs.

Normalization	mean	std	median	q25	q75	min	max	iqmean
Human	0.1353	0.7541	0.0338	0.00072	0.398	-5.024	4.738	0.1137
Min–Max	0.3802	0.3099	0.2899	0.0969	0.7143	0.0	0.9881	0.3426

Table 5: Per-game final evaluation for DQN (human- vs. min–max normalized). Each cell aggregates 10 episodes $\times$ 4 seeds = 40 total episodes in that game.

Game	Norm	mean	std	min	max
Alien	Human	0.0624	0.0752	0.0048	0.2636
	Min–Max	0.1607	0.1250	0.0650	0.4950
Amidar	Human	0.0226	0.0138	0.00072	0.0450
	Min–Max	0.2005	0.1065	0.0323	0.3733
Assault	Human	0.3167	0.1120	-0.0262	0.4920
	Min–Max	0.7216	0.1703	0.2005	0.9881
Boxing	Human	-0.4167	1.9504	-5.0238	4.7381
	Min–Max	0.7469	0.0635	0.5969	0.9147
Breakout	Human	0.3796	0.1246	0.1096	0.6080
	Min–Max	0.3454	0.0987	0.1316	0.5263
Freeway	Human	0.7162	0.0589	0.6419	0.8784
	Min–Max	0.7571	0.0622	0.6786	0.9286
MsPacman	Human	0.0099	0.0120	-0.0076	0.0262
	Min–Max	0.1047	0.0484	0.0340	0.1702
Pong	Human	-0.0083	0.0074	-0.01	0.0233
	Min–Max	0.0050	0.0221	0.0	0.1

Evaluation Results Table 4 aggregates final human-/min–max-normalized returns *over all 32 runs*. A game-by-game breakdown (Table 5) highlights large variability: *Freeway* can exceed 0.7 (human norm) or 0.75 (min–max), while *Boxing* sees a wide range from -5 to nearly $+5$ in human norm.

Observations In summary:

- **Episodic length** stabilizes around 3500–4000 steps on average, with some extreme runs either terminating quickly or persisting up to 8000 steps.
- **SPS** quickly rises to around 170–180, illustrating the efficiency of the implementation (though some runs are slower).

- **Q-values and TD loss** both exhibit broad variability. On average, Q-values climb steadily to 4–5, but certain runs exceed 10. The TD loss can spike above 3 for some seeds, indicating unstable updates.
- **Returns** show moderate success on easier tasks like *Freeway* and *Boxing*, but remain low in *Pong* or *MsPacman*. Overall, min–max mean is about 0.38, whereas human-normalized is only 0.14 (due in part to highly negative outliers on certain seeds).
- **Emissions** remain modest, at about 0.00647 kg CO_2-eq per run. This is unsurprising for a 100k-step budget, but still notable for comparing across algorithms in subsequent sections.

DQN thus provides a baseline—relatively simple and lightweight—to which we will compare Double DQN, Prioritized Experience Replay, Dueling DQN, and C51 in the next subsections, evaluating whether each extension justifies its additional complexity and energy usage.

4.2.2 Double DQN

(Hyper)Parameters Table 6 shows the main hyperparameters used in our Double DQN implementation. As with the baseline DQN (Section 4.2.1), `env_id` and `seed` vary across the 32 runs (eight Atari games × four seeds), while the rest remain unchanged. In particular, we again set `buffer_size`=10k and `learning_starts`=1000. The Double DQN update rule differs from standard DQN by separately selecting the action and evaluating its value, aiming to reduce the maximization bias that arises from using the same values to both choose and evaluate an action.

Hyperparameter Tuning To isolate the effect of Double DQN, we kept all settings identical to the baseline DQN, simply enabling the Double DQN update scheme. Following [4], we tested a higher `target_network_frequency` (e.g. 3000, scaled from the 10k–20k range in the original paper), but at 100k steps, performance was comparable or slightly better with 1000, so we retained the lower frequency.

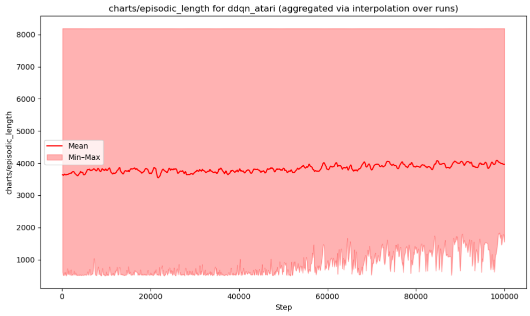
Training Dynamics (Aggregated Over 32 Runs) Figure 6 presents key metrics—episodic length, steps per second (SPS), estimated Q-values, and TD loss—aggregated across 32 runs (eight games, four seeds each).

Episodic length and SPS curves are very similar to baseline DQN's (Section 4.2.1). Meanwhile, the mean Q-values grow more modestly than DQN's (which often exceed 4–5 by the end), suggesting Double DQN's approach does mitigate overestimation somewhat. TD loss remains low overall, though some runs spike above 2.0 near late training.

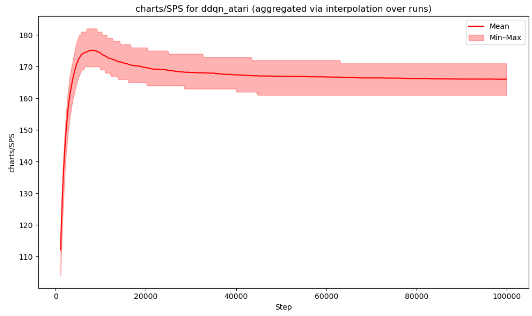
Episodic Return (Human vs. Min–Max Normalized) Figures 7 and 8 show Double DQN's aggregated episodic returns (human- and min–max-normalized, respectively). Figures 9 and 10 break these results down by game.

Table 6: Key hyperparameters for Double DQN. Only `env_id` and `seed` change across runs.

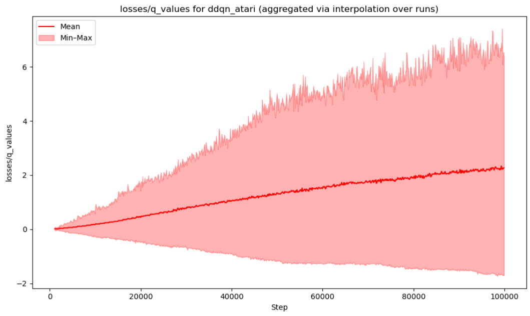
Parameter	**Value**
`exp_name`	ddqn_atari
`seed`	1..4
`torch_deterministic`	True
`cuda`	True
`track`	True
`wandb_project_name`	rlsb
`capture_video`	False
`save_model`	True
`upload_model`	False
`env_id`	e.g. AlienNoFrameskip-v4
`total_timesteps`	100000
`learning_rate`	0.0001
`num_envs`	1
`buffer_size`	10000
`gamma`	0.99
`tau`	1.0
`target_network_frequency`	1000
`batch_size`	32
`start_e`, `end_e`	1.0 $\rightarrow$ 0.01
`exploration_fraction`	0.1
`learning_starts`	1000
`train_frequency`	4



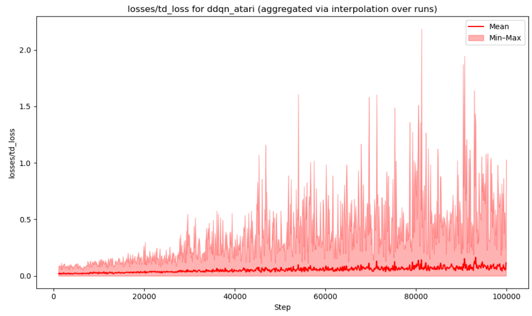
(a) Episodic length (`charts_episodic_length`). The mean sits around 3500–4000 steps, min–max ranges from near 0 up to 8000.



(b) Steps per second (SPS). After an initial climb near 180, the mean gradually settles around 165–170.



(c) Estimated Q-values (`losses/q_values`). The mean climbs from 0 to about 2–3, with upper outliers above 6.



(d) TD loss (`losses/td_loss`). Occasional spikes above 2.0 reflect instability on certain seeds.

Figure 6: Double DQN training metrics over 100k steps, aggregated over 32 runs.

Figure 7: Double DQN episodic return (human-normalized), aggregated across 32 runs.

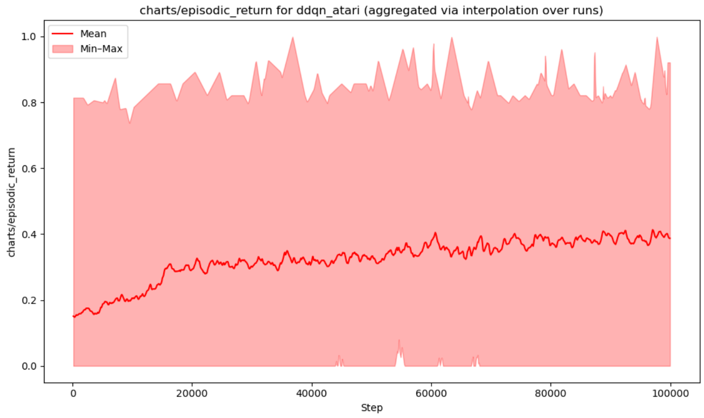


Figure 8: Double DQN episodic return (min–max normalized), aggregated across 32 runs.

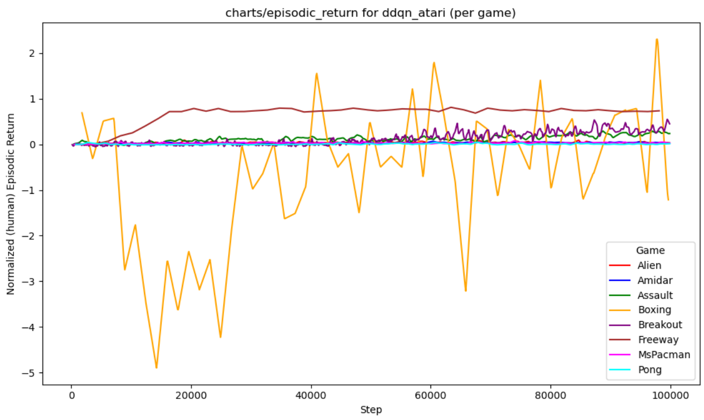


Figure 9: Double DQN returns per game (human-normalized). Some large negative dips occur in *Boxing*, while *Freeway* remains relatively high.

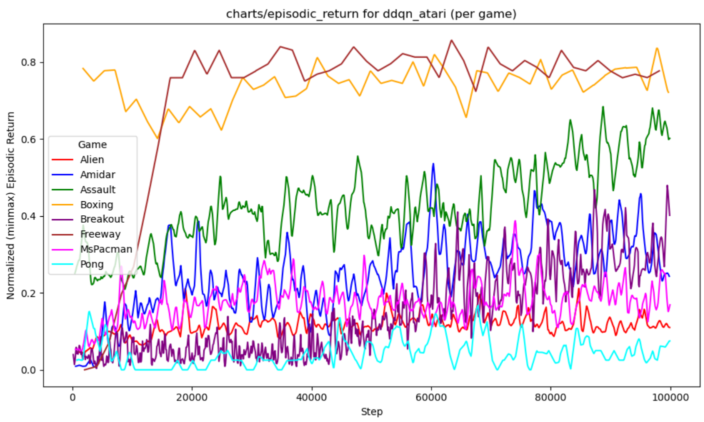


Figure 10: Double DQN returns per game (min–max normalized).

Table 7: Carbon emissions (kg CO_2eq) for Double DQN, aggregated over 32 runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmean
Double DQN	0.006672	0.000282	0.006549	0.006477	0.006755	0.006377	0.007267	0.006565

Table 8: Overall final evaluation (10 episodes each) for Double DQN across 32 runs.

Normalization	mean	std	median	q25	q75	min	max	iqmean
Human	0.0226	1.0083	0.0527	0.0127	0.2871	-8.5952	2.5952	0.0894
Min–Max	0.3737	0.2854	0.2887	0.1244	0.7054	0.0	1.0	0.3272

As in the baseline, *Freeway* can achieve near 0.7–0.8 in human norm, while *Boxing* causes occasional highly negative runs. Min–max normalized returns rise from ~ 0.2 to ~ 0.4, similar to DQN's overall trajectory.

Emissions Table 7 summarizes Double DQN's CO_2-eq emissions across 32 runs. The mean is about **0.00667 kg**, slightly above DQN's ~ 0.00647.

Evaluation Results Table 8 compiles final returns (human-/min–max normalization) aggregated over the 32 runs. Compared to DQN's $\sim$0.135 (human) and $\sim$0.380 (min–max), Double DQN attains 0.023 (human) and 0.374 (min–max). While the min–max average is comparable, the human-normalized mean is noticeably lower due to substantial negative outliers (again, notably *Boxing*).

Table 9 shows the game-by-game breakdown, indicating *Boxing* yields a min of -8.5952 and max of 2.5952 in human-normalized scale, dragging down the overall mean. Meanwhile, *Freeway* remains consistently high.

Comparison with Baseline DQN Beyond the final statistics, we can compare Q-values and TD loss directly via overlapping curves. Figure 11 on the following page shows the `losses/q_values` for both algorithms, with `ddqn_atari` in red and `dqn_atari` in gold; Double DQN's mean Q-values grow more slowly, suggesting less overestimation. Figure 12 on page 30 indicates TD loss remains similarly small for both, though DQN occasionally spikes higher. The barplot in figure 13 shows the mean emissions side-by-side.

Overall, Double DQN indeed moderates Q-value inflation compared to standard DQN, but under the 100k-step constraint, this reduction in overestimation does not strongly translate into consistently higher final returns.

Observations

- **Q-values and Losses:** Double DQN's Q-values peak lower than DQN's (about 2–3 vs. 4–5), aligning with the bias-reduction theory. TD losses remain small for both algorithms, with occasional spikes.

Table 9: Per-game final evaluation for Double DQN (human- vs. min–max normalized). Each row aggregates 40 total episodes (10 per seed).

Game	Norm	mean	std	min	max
Alien	Human	0.0514	0.0340	0.0094	0.1327
	Min–Max	0.1424	0.0565	0.0725	0.2775
Amidar	Human	0.0320	0.0222	0.0127	0.0953
	Min–Max	0.2729	0.1708	0.1244	0.7604
Assault	Human	0.2310	0.1071	-0.0427	0.4103
	Min–Max	0.5913	0.1628	0.1754	0.8640
Boxing	Human	-1.1607	2.4963	-8.5952	2.5952
	Min–Max	0.7227	0.0813	0.4806	0.8450
Breakout	Human	0.2666	0.1470	0.0764	0.8738
	Min–Max	0.2559	0.1165	0.1053	0.7368
Freeway	Human	0.7213	0.0493	0.6419	0.8784
	Min–Max	0.7625	0.0521	0.6786	0.9286
MsPacman	Human	0.0291	0.0245	-0.0037	0.0730
	Min–Max	0.1820	0.0986	0.0497	0.3586
Pong	Human	0.0100	0.0559	-0.01	0.3233
	Min–Max	0.0600	0.1676	0.0	1.0

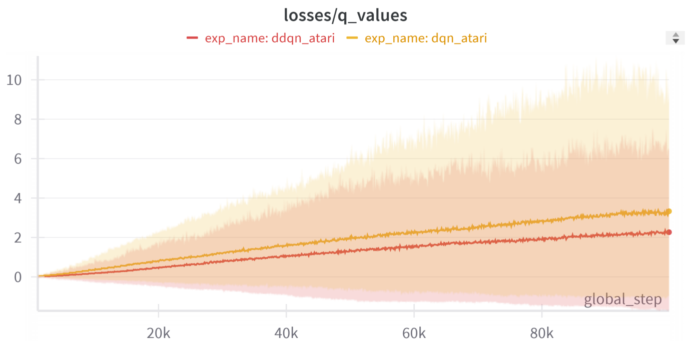


Figure 11: Comparison of mean Q-values (with min–max shading) for DQN (gold) vs. Double DQN (red).

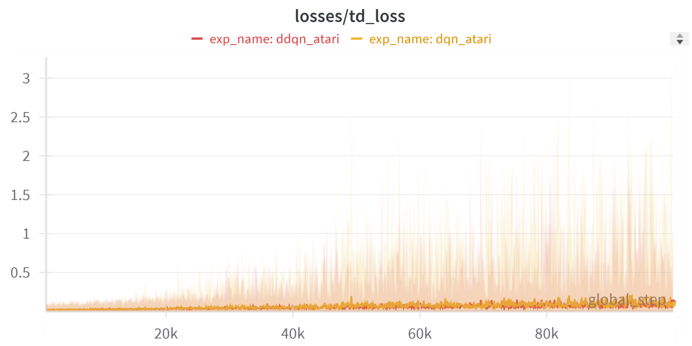


Figure 12: Comparison of TD loss for DQN (gold) vs. Double DQN (red). Both remain near 0 for extended periods, though DQN shows slightly higher spikes.

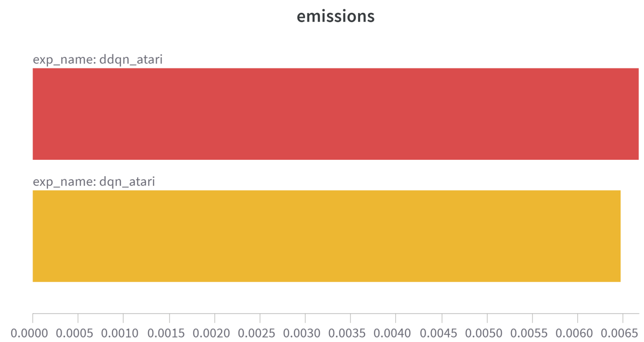


Figure 13: Mean emissions of DQN (gold) and Double DQN (red).

Table 10: Key hyperparameters for Prioritized Experience Replay (PER). Only `env_id` and `seed` vary across runs.

Parameter	Value
exp_name	per_atari
seed	1..4
torch_deterministic	True
cuda	True
track	True
wandb_project_name	rlsb
capture_video	False
save_model	True
upload_model	False
env_id	e.g. AmidarNoFrameskip-v4
total_timesteps	100000
learning_rate	0.0001
num_envs	1
buffer_size	10000
gamma	0.99
tau	1.0
target_network_frequency	1000
batch_size	32
start_e, end_e	$1.0 \to 0.01$
exploration_fraction	0.1
learning_starts	1000
train_frequency	4

- **Performance:** The min–max normalized mean (0.374) is nearly the same as DQN's (0.380), while human-normalized is actually lower (0.023 vs. 0.135) due to certain highly negative runs, especially in *Boxing*.
- **Emissions:** Average ~ 0.00667 kg CO_2-eq, slightly higher than DQN's 0.00647 kg.

Hence, although Double DQN successfully limits Q-value overestimation, its advantage does not fully manifest in higher aggregate returns at 100k steps—indicating that more extensive training or additional refinements may be needed to reap its potential performance gains.

4.2.3 Prioritized Experience Replay

(Hyper)Parameters

Hyperparameter Tuning We retained the same settings as baseline DQN (Section 4.2.1) to highlight PER's impact alone. Although [5] suggests lowering the learning rate by a

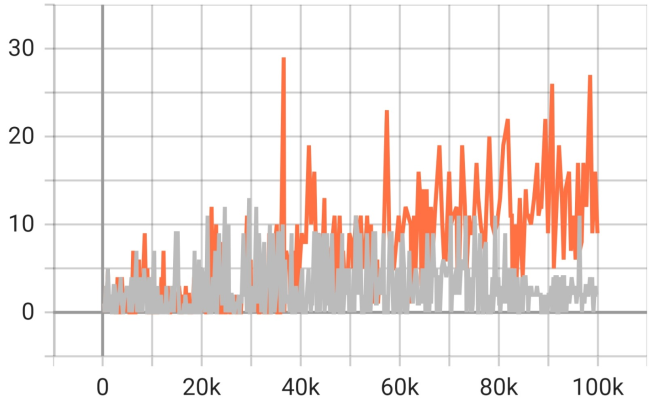


Figure 14: Comparison of learning rates for PER on *Breakout*. Orange curve: 1×10^{-4} (our final choice). Gray curve: $\frac{1}{4} \times 10^{-4}$ (as per [5]).

factor of four, our tests at 100k steps showed 1×10^{-4} worked better. Figure 14 compares these rates on *Breakout*, indicating faster convergence at 1×10^{-4}.

We also explored two implementations of the PER replay buffer—a `numpy`-based priority queue (for consistency) and a `torch`-based version. The `numpy` approach proved much slower, so we used the `torch` buffer in the final runs. Although the other DQN variants employ somewhat different replay optimizations (some from Stable Baselines), these differences did not appear to invalidate direct comparisons.

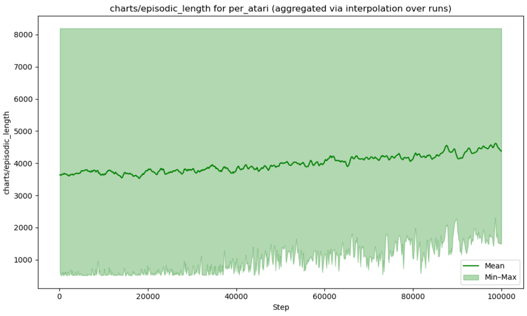
Training Dynamics Figure 15 aggregates four key metrics (episodic length, steps per second, Q-values, and TD loss) over 32 runs (eight games, four seeds each):

PER shows Q-values approaching or exceeding those of baseline DQN (which typically topped at 4–5). The TD loss is small on average but spikes in certain runs, suggesting that prioritizing high-error samples can exacerbate updates in some episodes.

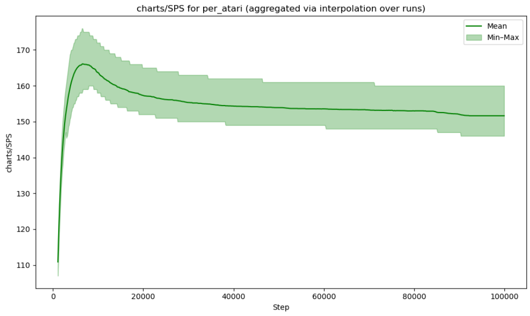
Episodic Return (Aggregated) Figures 16 (human-normalized) and 17 (min–max) show PER's aggregated episodic returns. The mean in human-normalized scale oscillates around zero, occasionally dipping below -2 or -3 in some seeds.

Per-Game Returns Figures 18 and 19 break down the performance by each Atari game (human vs. min–max normalization). We see, for instance, *Freeway* (orange line in min–max) steadily climbing to around 0.7–0.8, while *Alien* can drop below -3 in human scale.

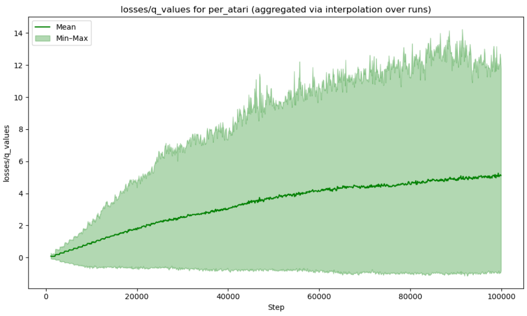
Emissions. Figure 20 shows a barplot comparing PER's mean emissions ($\approx 0.00725\,\text{kg}$) to DQN's ($\approx 0.00647\,\text{kg}$). The overhead of prioritized sampling may partly account for this higher footprint.



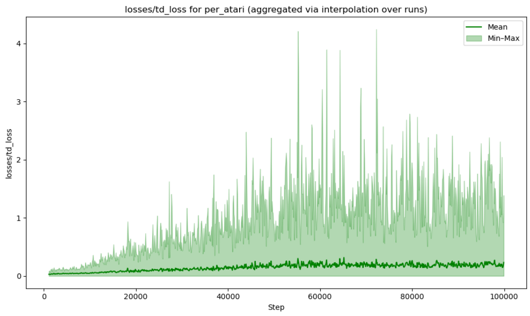
(a) Episodic length (`charts_episodic_length`). The mean hovers around 3500–4000, and min–max runs from near 0 up to 8000.



(b) Steps per second (SPS). The mean peaks above 170, then gradually declines to 150–160.



(c) Q-values (`losses/q_values`). The mean eventually surpasses 4–5, with outliers above 14.



(d) TD loss (`losses/td_loss`). Some runs spike above 3–4, showing instability in late training.

Figure 15: PER training metrics over 100k steps, interpolated across 32 runs.

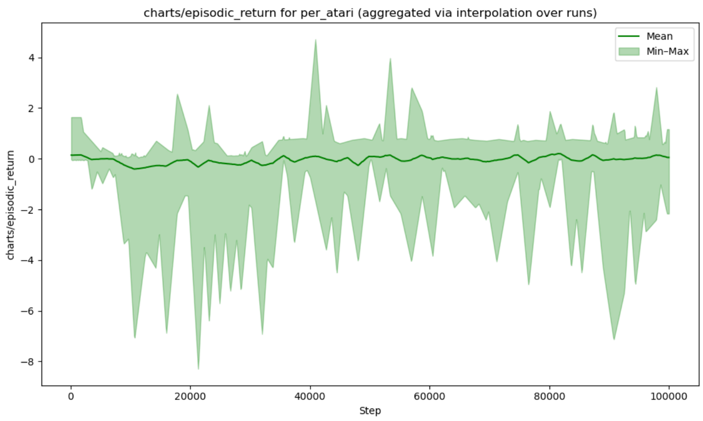


Figure 16: PER episodic return (human-normalized), aggregated over 32 runs. Negative outliers appear for certain seeds/environments.

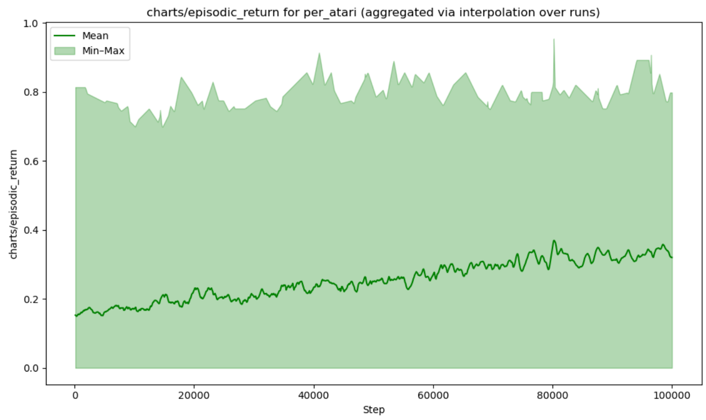


Figure 17: PER episodic return (min–max normalized), aggregated over 32 runs.

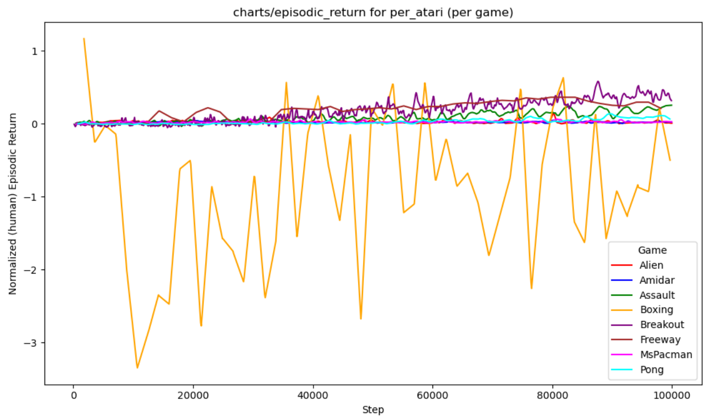


Figure 18: PER returns per game (human-normalized). *Alien* (orange) exhibits deep negative dips, while *Freeway*, *Assault*, and *Breakout* stay near or above zero.

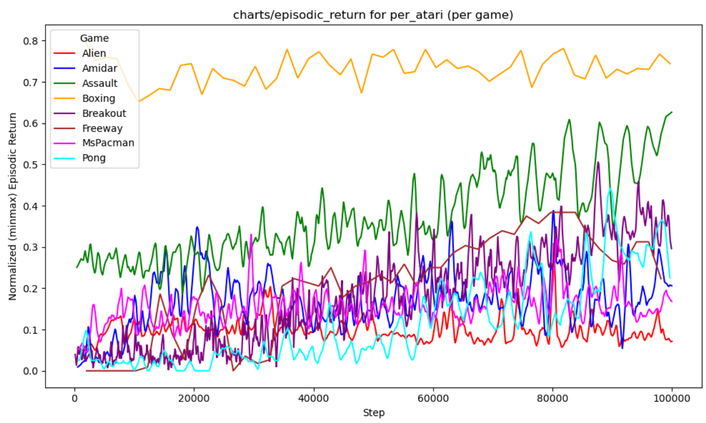


Figure 19: PER returns per game (min–max normalized). *Freeway* (gold line) and *Assault* (green) reach 0.7–0.8 near 100k steps.

Table 11: Carbon emissions (kg CO_2eq) for PER across 32 runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmean
PER	0.007254	0.000263	0.007146	0.007074	0.007354	0.006935	0.007819	0.007157

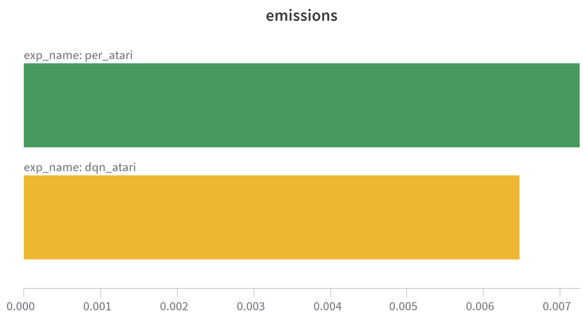


Figure 20: Mean emissions comparison: PER (green) at ~ 0.00725 kg vs. DQN (gold) at ~ 0.00647 kg.

Table 12: Overall final evaluation (10 episodes each) for PER across 32 runs.

Normalization	mean	std	median	q25	q75	min	max	iqmean
Human	0.0607	1.0170	0.0539	0.0175	0.2541	-10.2619	6.8809	0.0813
Min–Max	0.3533	0.2695	0.2583	0.1499	0.6265	0.0	0.9845	0.3087

Table 13: Per-game final evaluation for PER (human- vs. min–max normalized). Each row aggregates 10 episodes × 4 seeds per environment.

Game	Norm	mean	std	min	max
Alien	Human	0.0246	0.0315	-0.0147	0.0996
	Min–Max	0.0978	0.0523	0.0325	0.2225
Amidar	Human	0.0263	0.0220	-0.0029	0.0809
	Min–Max	0.2288	0.1693	0.0046	0.6498
Assault	Human	0.2135	0.0974	0.0232	0.3860
	Min–Max	0.5648	0.1480	0.2757	0.8270
Boxing	Human	-0.6667	2.7351	-10.2619	6.8809
	Min–Max	0.7388	0.0890	0.4264	0.9845
Breakout	Human	0.3854	0.2204	0.00997	0.9070
	Min–Max	0.3500	0.1746	0.0526	0.7632
Freeway	Human	0.4071	0.3381	0.0	0.8446
	Min–Max	0.4304	0.3574	0.0	0.8929
MsPacman	Human	0.0338	0.0111	0.0119	0.0515
	Min–Max	0.2008	0.0446	0.1126	0.2723
Pong	Human	0.0617	0.0682	-0.01	0.2233
	Min–Max	0.2150	0.2045	0.0	0.7000

Comparison with Baseline DQN. By final evaluation, PER's overall human-norm mean (0.0607) is lower than DQN's (0.135), and its min–max mean (0.353) lags behind DQN's 0.380. Figure 20 further shows PER's emissions exceed DQN's by ~ 0.0008 kg CO_2-eq on average—likely due to the overhead from prioritized sampling and somewhat longer average episodes.

Observations.

- **Implementation Details:** We used a `torch`-based PER buffer to avoid severe slowdowns in the `numpy` version.
- **Learning Rate:** $\frac{1}{4} \times 10^{-4}$, as suggested in [5], underperformed at 100k steps compared to 1×10^{-4} (Figure 14).
- **Performance:** PER did not consistently outperform baseline DQN within 100k steps: human-norm mean is 0.0607 vs. DQN's 0.135.
- **Emissions:** PER's overhead leads to slightly higher energy usage (0.00725 kg CO_2-eq) than DQN's 0.00647 kg.

In summary, while prioritizing high-error samples can yield benefits in longer training runs, our 100k-step Atari benchmark does not showcase a strong advantage. Further hyperparameter tuning or more extended runs might better reveal PER's strengths.

4.2.4 Dueling DQN

(Hyper)Parameters

Innovation: Dueling Architecture The *Dueling DQN* [6] architecture modifies the final layers of a Q-network to separate the estimation of state-value $V(s)$ from advantage $A(s, a)$. These two "heads" are combined to yield

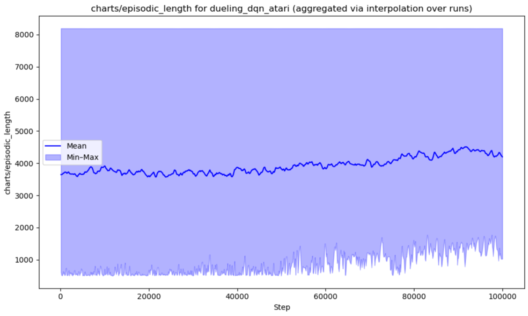
$$Q(s,a) = V(s) + A(s,a) \; - \; \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a'),$$

which aims to stabilize learning in states where the choice of action is less critical.
Network Size Note: Because the dueling architecture introduces separate streams for V and A, the model ends up with slightly more parameters. We considered reducing the size of the shared layer to keep the overall parameter count equal to the baseline, but the difference was only about 512 neurons, so we retained the original layer size for consistency with the other DQN variants. We also tested whether gradient clipping (as mentioned in some references) would improve stability, but found no clear benefit here. Finally, we did not combine dueling with Double DQN or PER, as our goal is to isolate the contribution of this single tweak in terms of performance and emissions.

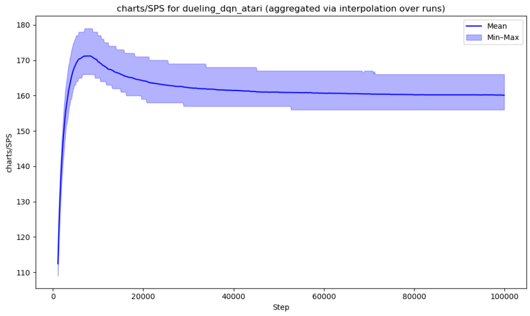
Training Dynamics

Table 14: Key hyperparameters for Dueling DQN. Only `env_id` and `seed` vary across runs.

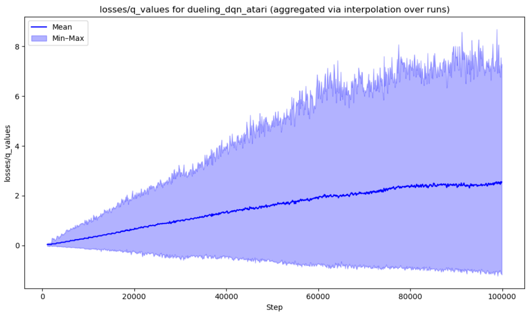
Parameter	**Value**
`exp_name`	dueling_dqn_atari
`seed`	1..4
`torch_deterministic`	True
`cuda`	True
`track`	True
`wandb_project_name`	rlsb
`capture_video`	False
`save_model`	True
`upload_model`	False
`env_id`	e.g. AlienNoFrameskip-v4
`total_timesteps`	100000
`learning_rate`	0.0001
`num_envs`	1
`buffer_size`	10000
`gamma`	0.99
`tau`	1.0
`target_network_frequency`	1000
`batch_size`	32
`start_e`, `end_e`	1.0 $\rightarrow$ 0.01
`exploration_fraction`	0.1
`learning_starts`	1000
`train_frequency`	4



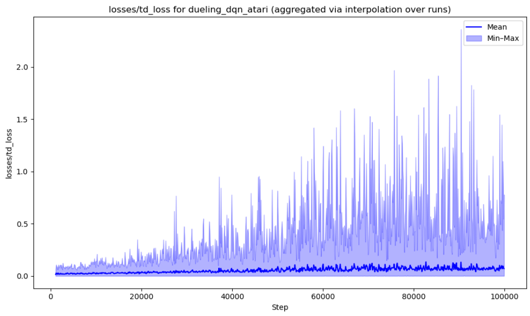
(a) Episodic length. The mean hovers around 3500–4000 steps, with min–max from near 0 up to 8000.



(b) Steps per second (SPS). After an initial ramp-up near 170, it stabilizes around 155–160.



(c) Q-values (`losses/q_values`). The mean grows beyond 4–5, with outliers near 8–10.



(d) TD loss (`losses/td_loss`). Some runs spike above 1.5–2.0, showing moderate variability late in training.

Figure 21: Dueling DQN training metrics over 100k steps, aggregated over 32 runs.

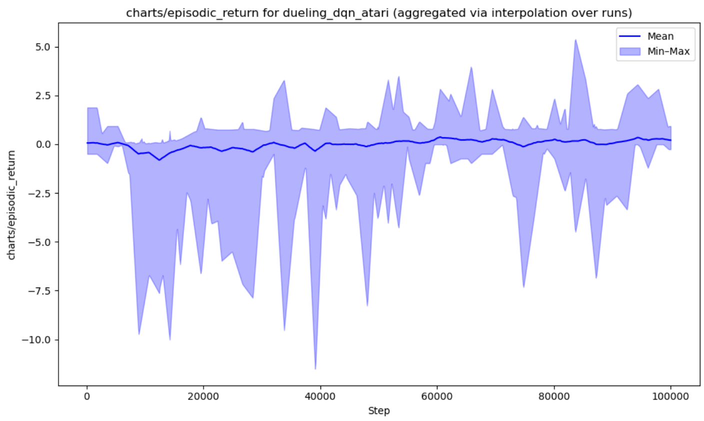


Figure 22: Dueling DQN episodic return (human-normalized) over 100k steps. Variance is significant, with some dips below -8.

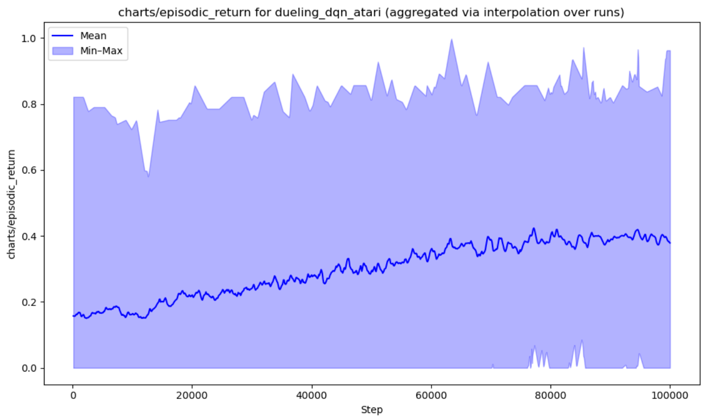


Figure 23: Dueling DQN episodic return (min–max normalized) aggregated over 32 runs.

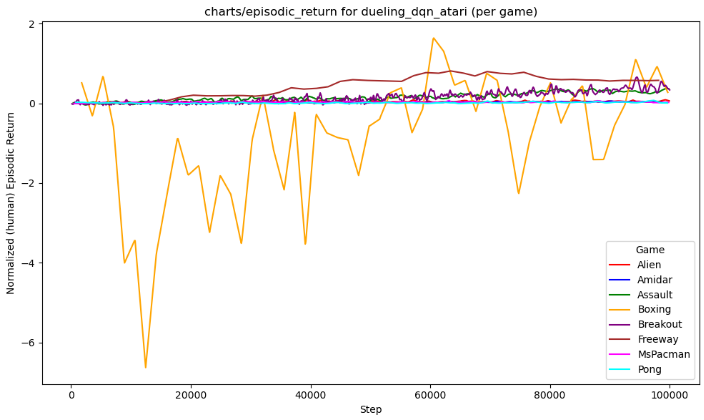


Figure 24: Dueling DQN returns per game (human-normalized). *Boxing* (orange) experiences deep negative dips, while *Freeway* and *Assault* can reach higher normalized scores.

Episodic Return (Aggregated)

Per-Game Returns

Emissions

Evaluation Results

Table 15: Carbon emissions (kg CO_2eq) for Dueling DQN across 32 runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmea
Dueling DQN	0.006893	0.0002901	0.006742	0.006672	0.007002	0.006617	0.007478	0.0067

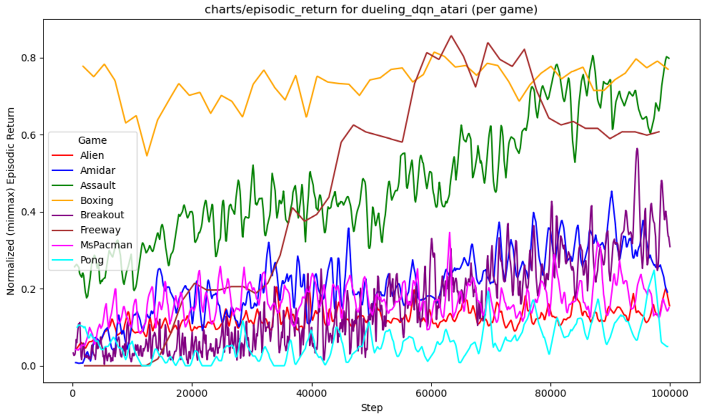


Figure 25: Dueling DQN returns per game (min–max normalized). *Freeway* (gold) and *Assault* (green) often exceed 0.7.

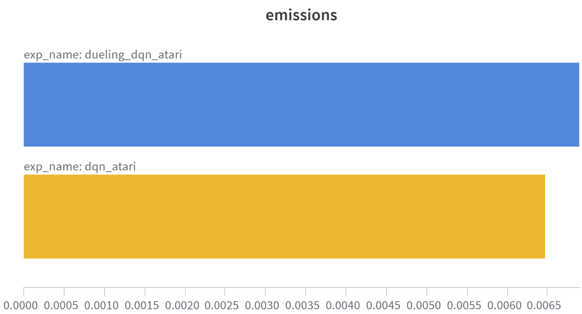


Figure 26: Mean emissions comparison: Dueling DQN (blue) at ~ 0.00689 kg vs. DQN (gold) at ~ 0.00647 kg.

Table 16: Overall final evaluation (10 episodes each) for Dueling DQN across 32 runs.

Normalization	mean	std	median	q25	q75	min	max	iqmean
Human	0.1860	0.5258	0.0402	0.0148	0.3530	-1.9286	3.5476	0.1020
Min–Max	0.3849	0.3056	0.2632	0.1100	0.7448	0.0	0.9523	0.3454

Table 17: Per-game final evaluation for Dueling DQN (human- vs. min–max normalized).

Game	Norm	mean	std	min	max
Alien	Human	0.0398	0.0346	-0.0072	0.1493
	Min–Max	0.1231	0.0574	0.0450	0.3050
Amidar	Human	0.0372	0.0193	0.0235	0.0935
	Min–Max	0.3127	0.1482	0.2074	0.7465
Assault	Human	0.3200	0.0985	0.1057	0.4684
	Min–Max	0.7266	0.1498	0.4010	0.9523
Boxing	Human	0.1190	1.3282	-1.9286	3.5476
	Min–Max	0.7643	0.0432	0.6977	0.8760
Breakout	Human	0.4020	0.2904	-0.0565	1.0066
	Min–Max	0.3632	0.2300	0.0	0.8421
Freeway	Human	0.5372	0.3162	0.0	0.7770
	Min–Max	0.5679	0.3342	0.0	0.8214
MsPacman	Human	0.0265	0.0204	0.00018	0.0736
	Min–Max	0.1713	0.0823	0.0654	0.3613
Pong	Human	0.0067	0.0185	-0.01	0.0567
	Min–Max	0.0500	0.0555	0.0	0.2

Comparison with Baseline DQN In final evaluation, Dueling DQN's *human*-norm mean is 0.1860 vs. DQN's 0.1353, and the *min–max* mean is 0.3849 vs. DQN's 0.3802—so we see a small positive difference on average. However, *Boxing* outliers (some runs exceed 3.5, others dip below −1.9) inflate the variance. Meanwhile, emissions at ~ 0.00689 kg CO_2-eq are higher than DQN's ~ 0.00647 but remain lower than some other variants (e.g. PER at 0.00725).

Observations

- **Architecture Impact:** The *dueling* approach separates state-value and advantage, intending to stabilize updates in states where action choices matter less.
- **Network Size:** We kept the same shared layer size as DQN, adding only a modest number of extra parameters.
- **Performance:** The final mean in human normalization (0.1860) is higher than baseline DQN's 0.1353, though min–max (0.3849 vs. 0.3802) is only slightly larger.
- **Emissions:** Dueling DQN uses ~ 0.00689 kg CO_2-eq, more than DQN's 0.00647 but less than PER's 0.00725.

4.2.5 Categorical DQN (C51)

(Hyper)Parameters

Innovation: Distributional RL Categorical DQN (**C51**) [8] models $Q(s, a)$ as a discrete probability distribution over possible returns, using `n_atoms` support points that span the interval $[v_{\min}, v_{\max}]$. By capturing more than a single expected value, C51 can potentially improve learning stability and performance, especially in risk- or variance-sensitive tasks.

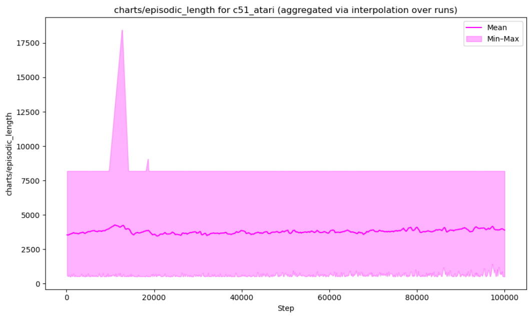
Hyperparameter Tuning We adopted CleanRL's `c51_atari` configuration with 51 atoms, $v_{\min} = -10$, $v_{\max} = 10$, and a learning rate of 2.5×10^{-4}. While we tested alternative rates, this default proved effective over only 100k steps.

We also experimented with `target_network_frequency` values of 1k, 5k, and 10k to see if less frequent updates might reduce overhead. Ultimately, 1k provided a good balance of stable returns and moderate emissions, aligning with other DQN-based variants.

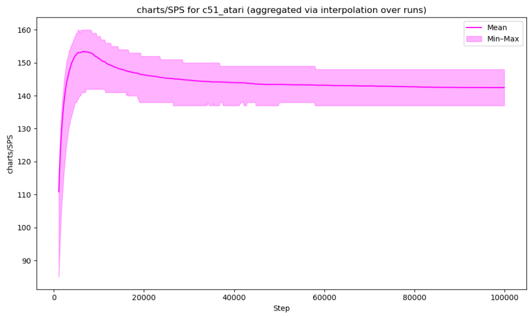
Training Dynamics Figure 27 shows that certain runs produce extremely long episodes (Figure 27a, subfloat a), while the overall `SPS` curve (subfloat b) hovers around 140–150 later in training, slightly lower than baseline DQN's ~ 160. The training loss (subfloat c) descends from near 4.0 to 2.0, while distributional `q_values` (subfloat d) broaden significantly.

Table 18: Key hyperparameters for the C51 algorithm. Only `env_id` and `seed` vary across runs.

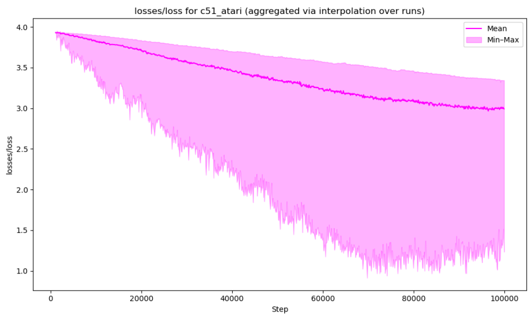
Parameter	**Value**
`exp_name`	c51_atari
`seed`	1..4
`torch_deterministic`	True
`cuda`	True
`track`	True
`wandb_project_name`	rlsb
`capture_video`	False
`save_model`	True
`upload_model`	False
`env_id`	e.g. AlienNoFrameskip-v4
`total_timesteps`	100000
`learning_rate`	0.00025
`num_envs`	1
`n_atoms`	51
`v_min`	-10
`v_max`	10
`buffer_size`	10000
`gamma`	0.99
`target_network_frequency`	1000
`batch_size`	32
`start_e`, `end_e`	$1.0 \rightarrow 0.01$
`exploration_fraction`	0.1
`learning_starts`	1000
`train_frequency`	4



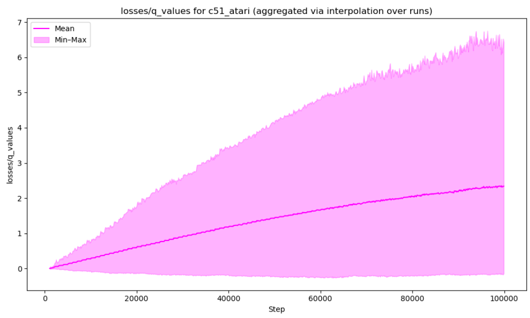
(a) Episodic length (`charts_episodic_length`). The mean is $\sim$ 3500–4000, while min–max occasionally spikes above 8000 or even 17500 in some runs.



(b) Steps per second (SPS). Mean peaks around 160–170, then slowly converges near 140–150.



(c) Overall loss (`losses/loss`). The mean steadily declines from about 4.0 toward near 2.0 by the end of training.



(d) Q-values (`losses/q_values`). The mean climbs from near 0 to $\sim$ 4–5, with outliers reaching 6–7.

Figure 27: C51 training metrics over 100k steps, aggregated (via interpolation) across 32 runs.

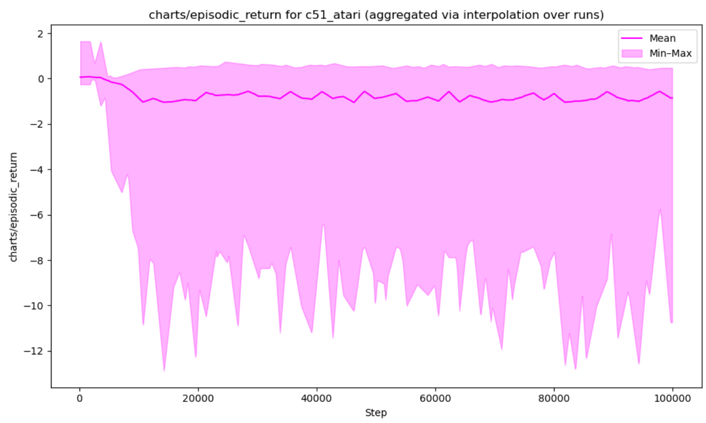


Figure 28: C51 episodic return (human-normalized), aggregated over 32 runs. Negative scores dominate, especially due to *Boxing*'s steep dips below -10.

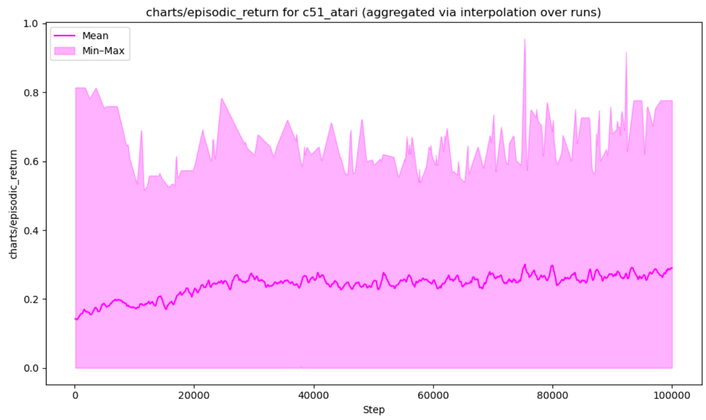


Figure 29: C51 episodic return (min–max normalized). The mean grows toward 0.25–0.30, indicating moderate performance relative to each game's min–max range.

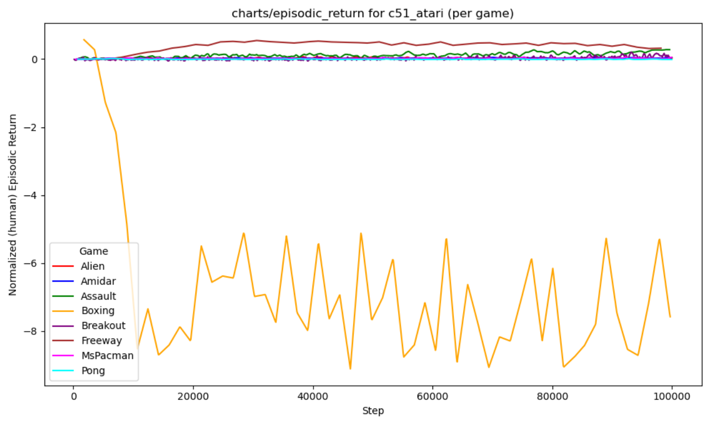


Figure 30: C51 returns per game (human-normalized). *Boxing* (orange) can plunge below -12, dwarfing improvements in other games.

Episodic Return (Aggregated) Due to extreme negative performance in certain games, such as *Boxing*, the human-normalized return (Figure 28) often falls below -1. In min–max (Figure 29), the mean approaches $\sim$ 0.25–0.30 by 100k steps.

Per-Game Returns

Emissions C51's mean emissions near 0.00775 kg exceed DQN's 0.00647 kg. Distributional overhead (computing 51 probability "atoms") likely increases GPU usage.

Evaluation Results

Comparison with Baseline DQN Overall, C51's human-normalized final mean is **-1.0811**, pulled down by severe negative performance in *Boxing* (around -9.39). Even

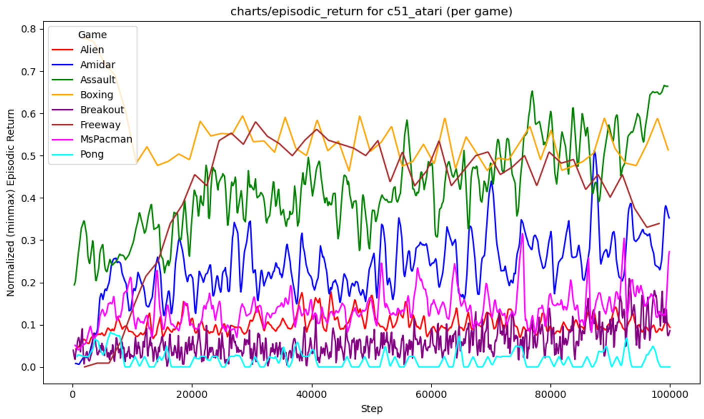


Figure 31: C51 returns per game (min–max normalized). *Assault* (green) climbs above 0.5, while *Pong* (cyan) remains near zero.

Table 19: Carbon emissions (kg CO_2eq) for C51 across 32 runs.

Algorithm	**mean**	**std**	**median**	**q25**	**q75**	**min**	**max**	**iqmean**
C51	0.007750	0.0003042	0.007655	0.007489	0.008099	0.007410	0.008257	0.007679

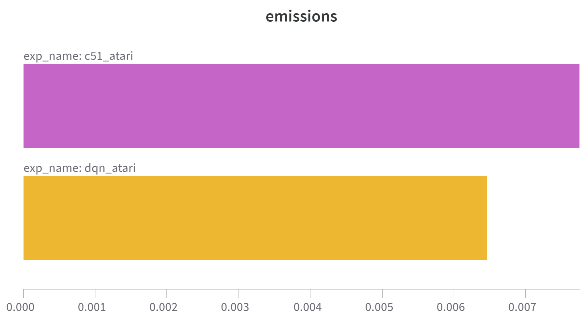


Figure 32: Mean emissions: C51 (magenta) at ~ 0.00775 kg vs. DQN (gold) at ~ 0.00647 kg.

Table 20: Overall final evaluation (10 episodes each) for C51 across 32 runs.

Normalization	**mean**	**std**	**median**	**q25**	**q75**	**min**	**max**	**iqmean**
Human	-1.0811	3.2862	0.0	-0.0132	0.0418	-12.8810	0.7770	0.00684
Min–Max	0.2503	0.2568	0.2005	0.0	0.4419	0.0	0.8270	0.13997

Table 21: Per-game final evaluation for C51 (human- vs. min–max normalized). Each row aggregates 40 total episodes (10 per seed).

Game	**Norm**	**mean**	**std**	**min**	**max**
Alien	Human	-0.0149	0.0127	-0.0343	0.00635
	Min–Max	0.0321	0.0211	0.0	0.0675
Amidar	Human	0.0336	0.0153	0.00431	0.0678
	Min–Max	0.2851	0.1181	0.0599	0.5484
Assault	Human	0.1994	0.1467	-0.0262	0.3860
	Min–Max	0.5434	0.2229	0.2005	0.8270
Boxing	Human	-9.3869	2.6728	-12.8810	-4.7857
	Min–Max	0.4548	0.0870	0.3411	0.6047
Breakout	Human	0.1478	0.2072	-0.0565	0.4751
	Min–Max	0.1618	0.1641	0.0	0.4211
Freeway	Human	0.3632	0.3688	0.0	0.7770
	Min–Max	0.3839	0.3899	0.0	0.8214
MsPacman	Human	0.0189	0.0176	-0.0057	0.0483
	Min–Max	0.1410	0.0707	0.0419	0.2592
Pong	Human	-0.01	5.27×10^{-18}	-0.01	-0.01
	Min–Max	0.0	0.0	0.0	0.0

ignoring that outlier, other titles do not significantly outperform baseline DQN's 0.1353. In min–max scale, C51's 0.2503 is also lower than DQN's 0.3802. Emissions, by contrast, are higher (~ 0.00775 kg vs. DQN's 0.00647), reflecting the overhead of computing a 51-atom return distribution each update.

Observations

- **Distributional Advantage:** Although distributional RL can better capture risk or reward variance, 100k steps may be insufficient to realize its full potential.
- **Performance:** Human-norm is -1.0811 vs. DQN's 0.1353, with many games remaining near/below zero. Min–max is 0.2503 vs. 0.3802.
- **Emissions:** C51 uses ~ 0.00775 kg CO_2-eq, higher than DQN's 0.00647, likely due to distributional overhead.
- **Target Network Frequency:** Testing 1k, 5k, and 10k found 1k gave decent stability and moderate power usage.

In summary, C51 did not outperform the baseline DQN in this 100k-step Atari benchmark, though distributional RL may yield advantages over longer runs or with additional tuning.

4.3 Overall Comparison of DQN-Based Algorithms

In this section, we synthesize the results of all five DQN-based algorithms:

- **Baseline DQN** (no additional tweaks)
- **Double DQN** (mitigating Q-value overestimation)
- **Prioritized Experience Replay (PER)** (sampling transitions by TD-error priority)
- **Dueling DQN** (separating state-value from advantage)
- **C51** (distributional RL with a categorical return distribution)

We compare them on two axes: final performance (evaluated in both human- and min–max-normalized scales) and carbon emissions.

Aggregated Final Returns and Emissions Table 22 compiles the final evaluation means from Sections 4.2.1, 4.2.2, 4.2.3, 4.2.4, and 4.2.5. We list each method's mean episodic return under both normalization schemes, along with its mean carbon footprint. For completeness, we include standard deviations (`std`) and other statistics where relevant.

Table 22: Overall final evaluation and emissions for DQN-based algorithms. Human-/min–max-normalized performance are the global means across 32 runs (8 games, 4 seeds). Emissions are reported in kg CO_2-eq. Lower or negative human-norm means indicate below-human performance on average (e.g. in *Boxing*), whereas higher min–max means imply better relative scores.

	Final Episodic Return (Mean)		**Emissions**
Algorithm	**Human Norm**	**Min–Max**	**(kg CO_2-eq)**
DQN (baseline)	0.1353	0.3802	0.00647
Double DQN	0.0226	0.3737	0.00667
PER	0.0607	0.3533	0.00725
Dueling DQN	0.1860	0.3849	0.00689
C51	−1.0811	0.2503	0.00775

Performance Discussion

- **Dueling DQN** achieves the highest human-norm mean (0.1860), though min–max is only slightly above baseline DQN (0.3849 vs. 0.3802). Its moderate overhead in computations yields emissions of 0.00689 kg, just above baseline.
- **Double DQN** has a low human-norm mean (0.0226) but a reasonably strong min–max mean (0.3737). Some games suffer from negative outliers (e.g. *Boxing*), but overall it matches baseline DQN in relative scale.
- **PER** does not strongly outperform DQN in short (100k-step) training, scoring 0.0607 human-norm and 0.3533 min–max, with slightly higher emissions (0.00725 kg). Prioritizing TD errors may show more benefit in longer runs.
- **C51** exhibits the largest negative dip on human-norm (−1.0811), largely due to extreme results in *Boxing* and a few other titles, but obtains 0.2503 in min–max. It also has the largest emissions (0.00775 kg) among these five, reflecting the overhead of a distributional approach with 51 "atoms."
- **Baseline DQN** remains a strong reference point. Though not best in any single metric, it has decent performance across games (especially 0.3802 in min–max), while keeping the lowest carbon footprint (0.00647 kg).

Table 23 extends the previous summary (Table 22) by including the interquartile mean (IQM) [19] for each of the three metrics:

- **Human-Normalized Returns** (mean and IQM)
- **Min–Max Normalized Returns** (mean and IQM)
- **Emissions (kg CO_2-eq)** (mean and IQM)

Recall that IQM is a robust estimator of central tendency, averaging only the middle 50% of data points (between the 25th and 75th percentiles), thus mitigating the impact of extreme outliers.

Table 23: DQN-Based Algorithms: Final Returns (Human & Min–Max Norm) *vs.* Emissions, including both Mean and Interquartile Mean (IQM). Data are aggregated over 32 runs (8 Atari games × 4 seeds). Negative human-norm means can stem from poor performance in certain games (e.g. *Boxing* with highly negative scores).

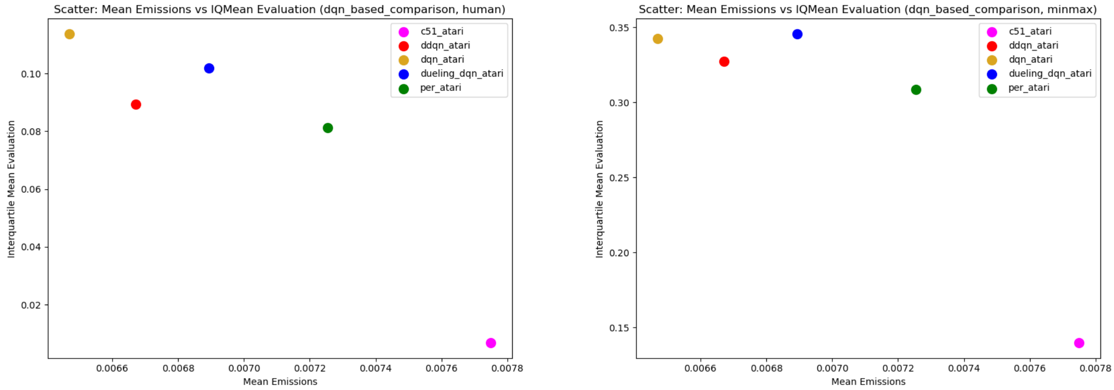
Algorithm	Human-Norm Return		Min–Max Return		Emissions (kg CO_2)	
	Mean	IQM	Mean	IQM	Mean	IQM
DQN	0.1353	0.1137	0.3802	0.3426	0.00647	0.00637
Double DQN	0.0226	0.0894	0.3737	0.3272	0.00667	0.00656
PER	0.0607	0.0813	0.3533	0.3087	0.00725	0.00716
Dueling DQN	0.1860	0.1020	0.3849	0.3454	0.00689	0.00678
C51	−1.0811	0.00684	0.2503	0.1400	0.00775	0.00768

Discussion.

- **Human-Norm vs. IQM.** Some methods (*e.g.*, Double DQN, C51) exhibit a large discrepancy between the mean and IQM in human-normalized returns, indicating a handful of extreme outliers (often due to specific games like *Boxing*).
- **Dueling DQN.** Its mean is highest in human-norm (0.1860), but the IQM (0.1020) is closer to baseline DQN's 0.1137, suggesting moderate overall gains once outliers are downweighted.
- **C51's Negative Mean.** With −1.08 in human-norm, C51 suffers from severely negative outliers; however, its IQM (0.00684) sits just above zero, reflecting that most runs or games are not quite so disastrous outside *Boxing*.
- **Emissions Overhead.** Distributional (C51) and prioritized (PER) methods do have higher carbon costs (around 0.0077 kg and 0.00725 kg, respectively) than vanilla DQN (0.00647 kg). The IQM for emissions shows a similar trend (0.00768 and 0.00716 vs. 0.00637).

Altogether, adding the IQM metric helps to highlight the presence of large outliers in short 100k-step runs. Dueling emerges as a modest improvement on baseline DQN once extreme seeds are downweighted, whereas C51 and PER do not yet exhibit strong benefits for the added cost in short training scenarios.

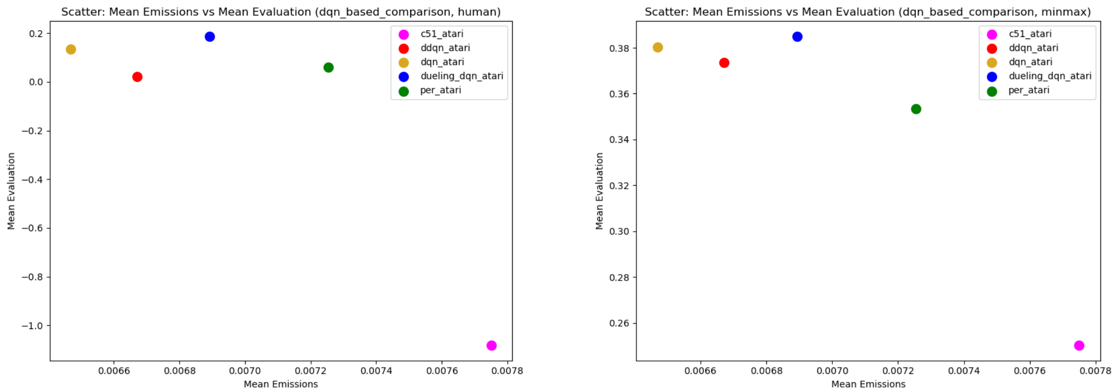
Scatter Plots of Emissions vs. Performance To visually depict the trade-off between carbon footprint and final performance, Figures 33 and 34 show scatter plots of `Mean Emissions` on the x-axis against (*i*) the `IQMean` or (*ii*) the `Mean` of final evaluation on the y-axis. We plot both the human-normalized and min–max normalized variants:



(a) Human Norm: Mean Emissions vs. IQMean Evaluation

(b) Min–Max: Mean Emissions vs. IQMean Evaluation

Figure 33: Scatter: Mean Emissions vs. Interquartile Mean (IQM) final evaluation, for DQN-based algorithms. C51 (magenta) appears far to the right (highest emissions) and near the bottom in human norm, though it's closer in min–max. DQN and Double DQN cluster with relatively low emissions.



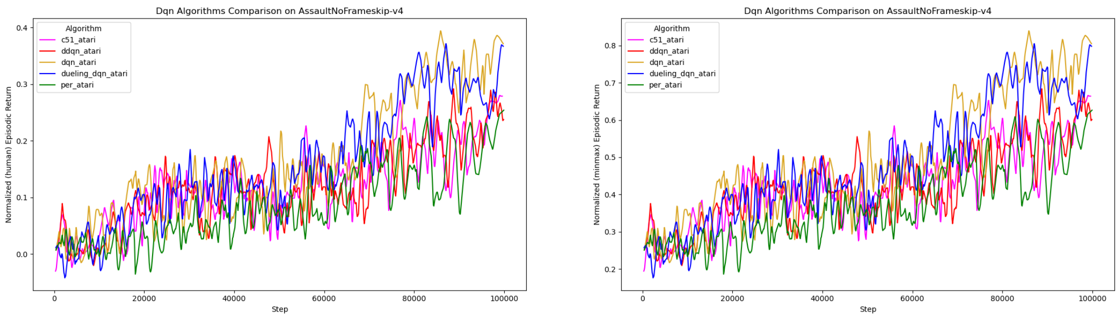
(a) Human Norm: Mean Emissions vs. Mean Evaluation

(b) Min–Max: Mean Emissions vs. Mean Evaluation

Figure 34: Scatter: Mean Emissions vs. Mean final evaluation, for DQN-based algorithms. Again, C51 (magenta) is an outlier with higher emissions and negative (human) or low (min–max) returns. Dueling (blue) has moderate emissions and the highest human-norm performance.

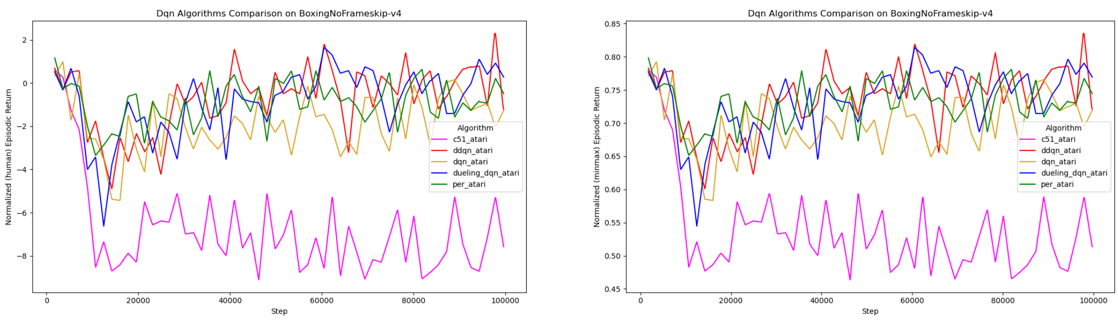
Game-by-Game Observations When looking individually per environment:

- **Boxing** heavily skews human-normalized averages for PER and especially C51, yielding strongly negative means. Dueling or Double DQN often handle *Boxing* more stably.
- **Freeway** is comparatively easier, so all variants converge to near-human or above. PER and Dueling both score well here.
- **Assault** sees moderate or high min–max normalized returns across the board; distributional methods like C51 can do fairly well in some seeds, but not enough to beat DQN or Dueling on average.



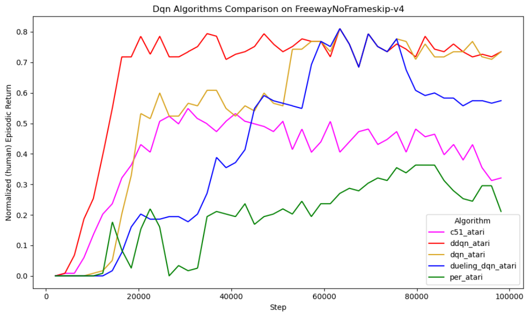
(a) Human Norm: Assault **(b)** Min–Max: Assault

Figure 35: Comparison of DQN-based algorithms on **Assault**, showing human-normalized (**left**) and min–max normalized (**right**) final returns over 100k steps. Distributional methods (e.g. C51) can excel in some seeds but not enough to surpass Dueling/DQN on average.

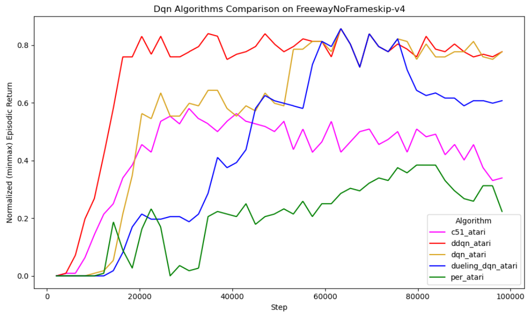


(a) Human Norm: Boxing **(b)** Min–Max: Boxing

Figure 36: Comparison of DQN-based algorithms on **Boxing**. The human-normalized scale (**left**) highlights large negative dips, especially for C51 and PER. Min–max normalized (**right**) shows moderate ranges for most algorithms.



(a) Human Norm: Freeway



(b) Min–Max: Freeway

Figure 37: Comparison of DQN-based algorithms on **Freeway**. All methods converge relatively quickly to near-human or above. PER and Dueling typically perform strongly on this environment.

Emissions and Efficiency As indicated by both Table 22, Table 23, and the scatter plots (Figures 33–34), **C51** has the highest mean emissions (~ 0.00775 kg), while **PER** also exceeds baseline levels at 0.00725 kg. **DQN** remains the lowest (0.00647 kg). In short 100k-step training, the benefits of distributional or prioritized approaches do not fully emerge, whereas their computational overhead (and hence emissions) is quite tangible.

Emissions Barplot Finally, Figure 38 illustrates a direct barplot comparison of the average emissions (with error bars for standard deviation) for the five methods.

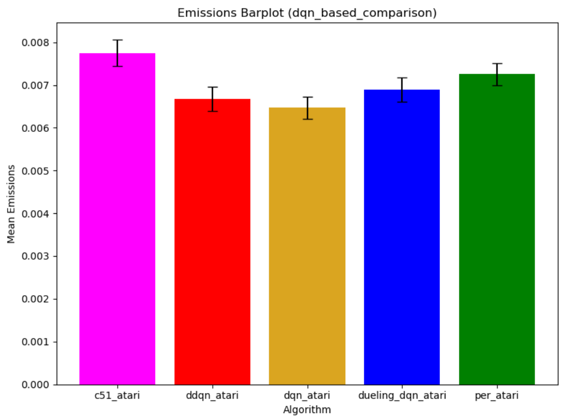


Figure 38: Emissions Barplot (DQN-based Comparison). C51 (magenta) leads with ~ 0.0078 kg, while DQN (gold) is at the low end (0.00647).

Key Takeaways

- **Dueling DQN** shows the best human-norm mean (0.1860), or second-best min–max (0.3849). Its overhead is modest.

- **Double DQN** is cheap in energy and helps curb Q-value inflation, but does not necessarily raise final returns in short runs (0.0226 human, 0.3737 min–max).
- **PER** is also more costly (0.00725 kg) with only slight gains in 100k steps, suggesting PER's advantage might need longer training to appear.
- **C51** is the outlier in both emissions and negative returns (human), though its IQM is far less extreme, indicating that only a few seeds/games are catastrophic.
- **Baseline DQN** remains a viable option at 100k steps, balancing decent performance and the lowest emissions of the five variants.

4.4 Policy-Based Algorithms

This section presents results for the three policy gradient methods.

4.4.1 REINFORCE

- Training execution and stability.
- Performance scores and energy consumption.

Training execution. Stability of policy gradient training. Final scores and emissions.

4.4.2 PPO (Proximal Policy Optimization)

- Effect of clipping on training stability.
- Performance vs. REINFORCE.

Did clipping help stabilize training? Efficiency compared to REINFORCE.

4.4.3 SAC (Soft Actor-Critic)

- Impact of entropy regularization.
- Energy efficiency compared to PPO and REINFORCE.

How did the entropy regularization affect performance? Energy trade-offs (is SAC more expensive to train?).

4.5 Overall Comparison of Policy-Based Algorithms

- Graphs comparing all policy-based methods.
- Which method achieved better stability?
- Summary of trade-offs between energy consumption and performance.

Graphs comparing policy-based methods. Which is more stable? Which is more efficient? Final takeaway: Did one clearly outperform the others in both energy efficiency and reward?

4.6 Cross-Category Comparison: DQN vs. Policy Gradient

- Which family was more energy-efficient?
- Graphs comparing the best-performing models from each category.
- Conclusion: Do policy-based methods require more energy but provide better sample efficiency?

Which family is generally more energy-efficient? Graphs comparing best DQN-based vs. best policy-based model. Final summary of the findings.

5 Implications of the Results

This section interprets the findings from the previous section and discusses their broader implications.

This section is more interpretative and application-focused—it explains why the results matter.

5.1 General Observations

- What trends emerged from the results?
- Were any algorithms unexpectedly energy-efficient?
- Did any algorithm perform significantly worse than expected in terms of energy usage?

What were the most notable trends in the results? Any surprising findings? (e.g., did a simpler method turn out to be more efficient than an advanced one?).

5.2 Energy Efficiency vs. Performance Trade-Off

- Which algorithms achieved the best performance at low energy costs?
- Are high-performance methods necessarily more energy-intensive?
- Is there a clear "sweet spot" balancing energy and reward?
- Are there diminishing returns for performance improvements vs. energy consumption?

Which algorithms performed well at low energy costs? Which algorithms achieved the best performance but at high energy costs? Is there a clear "sweet spot" algorithm?

5.3 Practical Implications for AI Sustainability

- If a company prioritizes high performance, which algorithm should they use?
- If low energy consumption is the main concern, what is the best choice?
- How can reinforcement learning be made more sustainable?

If companies want maximum performance, which algorithm should they choose? If a low-carbon footprint is the priority, what's the best option? What does this study suggest about future AI energy efficiency research? Seconda versione, dopo aver fatto notare della sezione conclusioni:

- Which algorithms should be prioritized in resource-limited environments?
- If reinforcement learning is applied in an industry setting, which method should be chosen for:
  - Maximum efficiency?
  - Best performance at a reasonable cost?
  - Low-carbon AI initiatives?
- How could energy-efficient reinforcement learning impact research and business decisions?

——— ricorda video su "deepseek clone at 30$", dice tipo che quale algo usi sembra non fare differenza, quindi si potrebbe optare a prescindere per il più green, o soft spot tra reinforce e ppo per dire, magari reinforce with baseline e migliori batch.

5.4 Limitations and Future Work

(dopo lo slash la versione aggiornata dopo aver fatto notare che c'è la sezione conclusioni, la versione aggiornata delle direzioni future è semplicemente azzeccata dopo quella originale, nel senso primi due item sono vecchi, altri 3 nuovi)

- **CodeCarbon limitations:** Real-time tracking slowed down experiments. / Real-time tracking was too slow, limiting fine-grained energy measurements.
- **Hardware constraints:** Would stronger GPUs reduce energy costs through faster convergence? / The computational budget influenced the choice of tested methods.
- **Future directions:**
  - Development of energy-efficient RL architectures.
  - Methods for optimizing training without excessive power use.
  - Can reinforcement learning frameworks be modified to prioritize energy-efficient training?
  - How does energy consumption vary with different hardware architectures?
  - Investigating potential trade-offs between batch size, learning rate, and energy efficiency.

CodeCarbon limitations: Not tracking step-by-step emissions limited insights into fine-grained energy consumption patterns. Hardware constraints: Would more powerful GPUs reduce energy costs through faster convergence? Future Research Directions: Could energy-efficient architectures be developed? Should reinforcement learning algorithms be adapted for lower energy consumption?

6 Conclusions

This section summarizes the key findings of the study and reflects on the broader significance of energy-efficient reinforcement learning.

6.1 Summary of Findings

- What were the most important insights gained from the experiments?
- Which algorithms proved to be the best trade-off between energy efficiency and performance?
- Which models consumed excessive energy without substantial performance benefits?

6.2 Final Thoughts on Energy-Efficient Reinforcement Learning

- Can reinforcement learning methods be optimized for sustainability without sacrificing performance?
- How does this study contribute to the ongoing conversation about AI's environmental impact?

6.3 Future Research Directions

- What are the next steps for improving energy efficiency in deep reinforcement learning?
- How can future benchmarks expand upon this study?
- Can policy-driven techniques (e.g., adaptive learning rates) optimize both performance and sustainability?

References

[1] V. Mnih et al., *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[2] V. Mnih et al., "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2016.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[6] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[7] J. Peng and R. J. Williams, "Incremental multi-step q-learning," in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 226–232.

[8] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *arXiv preprint arXiv:1707.06887*, 2017.

[9] M. Fortunato et al., "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[10] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, "Data-efficient reinforcement learning with self-predictive representations," *arXiv preprint arXiv:2007.05929*, 2020.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] T. P. Lillicrap et al., *Continuous control with deep reinforcement learning*, 2019. arXiv: `1509.02971 [cs.LG]`. [Online]. Available: https://arxiv.org/abs/1509.02971.

[14] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1587–1596.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: `1801.01290 [cs.LG]`. [Online]. Available: https://arxiv.org/abs/1801.01290.

[16] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," *arXiv preprint arXiv:2004.13649*, 2020.

[17] L. Kaiser et al., *Model-based reinforcement learning for atari*, 2024. arXiv: `1903.00374 [cs.LG]`. [Online]. Available: https://arxiv.org/abs/1903.00374.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013, ISSN: 1076-9757. DOI: 10.1613/jair.3912. [Online]. Available: http://dx.doi.org/10.1613/jair.3912.

[19] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, *Deep reinforcement learning at the edge of the statistical precipice*, 2022. arXiv: `2108.13264 [cs.LG]`. [Online]. Available: https://arxiv.org/abs/2108.13264.

[20] B. Courty et al., *Mlco2/codecarbon: V2.4.1*, version v2.4.1, May 2024. DOI: 10.5281/zenodo.11171501. [Online]. Available: https://doi.org/10.5281/zenodo.11171501.

[21] S. Huang et al., *Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms*, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html.

[22] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[23] Farama Foundation, *Arcade Learning Environment (ALE) Environments*, Accessed: 2025-02-06, 2025. [Online]. Available: https://ale.farama.org/environments/.