

RLSB

Software Engineering for Artificial Intelligence

# Reinforcement Learning Sustainability Benchmark

Report v1.0

Luca Strefezza\*

February 24, 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

<b>1</b>	<b>Context</b>	<b>4</b>
<b>2</b>	<b>Goals</b>	<b>5</b>
<b>3</b>	<b>Methodological Steps</b>	<b>5</b>
3.1	Algorithms Selection . . . . .	5
3.1.1	Value-Based Methods . . . . .	6
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 . . . . .	16
3.4.3	Interpolation and Aggregation . . . . .	16
3.4.4	Plot Generation and CSV Output . . . . .	17
<b>4</b>	<b>Preliminary Results and Findings</b>	<b>17</b>
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 . . . . .	25
4.2.3	Prioritized Experience Replay . . . . .	32
4.2.4	Dueling DQN . . . . .	38
4.2.5	Categorical DQN (C51) . . . . .	45
4.3	Overall Comparison of DQN-Based Algorithms . . . . .	52
4.4	Policy-Based Algorithms . . . . .	58
4.4.1	REINFORCE . . . . .	59
4.4.2	Proximal Policy Optimization (PPO) . . . . .	63
4.4.3	Soft Actor-Critic (SAC) . . . . .	68
4.5	Overall Comparison of Policy Gradient Algorithms . . . . .	74
4.6	Overall Algorithm Comparison . . . . .	83
4.6.1	Final Evaluation Performance . . . . .	83
4.6.2	Emissions and Runtime . . . . .	84
4.6.3	Samples per Second (SPS) Comparison . . . . .	85
4.6.4	Performance vs. Emissions (Scatter Plots) . . . . .	86
4.6.5	Per-Game Details . . . . .	87

4.6.6	Per-Environment Episodic Returns (All Algorithms) . . . . .	88
4.6.7	Summary . . . . .	92
<b>5</b>	<b>Implications of the Results</b>	<b>93</b>
5.1	General Observations . . . . .	93
5.2	Energy Efficiency vs. Performance Trade-Off . . . . .	93
5.3	Practical Implications for AI Sustainability . . . . .	94
5.4	Limitations and Future Work . . . . .	94
<b>6</b>	<b>Conclusions</b>	<b>95</b>
6.1	Summary of Findings . . . . .	95
6.2	Final Thoughts on Energy-Efficient Reinforcement Learning . . . . .	95
6.3	Future Research Directions . . . . .	96
<b>References</b>		<b>97</b>

## 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 divided 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<sub>2</sub>-equivalents (CO<sub>2</sub>eq) in order to estimate the carbon footprint of their work. CO<sub>2</sub>-eq is a standardized measure used to express the global warming potential of various greenhouse gases: the amount of CO<sub>2</sub> that would have the equivalent global warming impact. For computing, which emits CO<sub>2</sub> via the electricity it is consuming, carbon emissions are measured in kilograms of CO<sub>2</sub>-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 (the agent can play roughly 4 times more games without significantly increasing the runtime).

**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  $84 \times 84$  pixels (`GrayScaleObservation` and `ResizeObservation`).
- a history of the last four frames is stacked (`FrameStack(4)`) to provide temporal context, compensating for the partially observable nature of the environment.

**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.

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

Feature	NoFrameskip-v4 (Our Setup)	ALE v5
Frame Skipping	Explicitly set via MaxAndSkipEnv(skip=4)	Implicit (default 4)
No-op Start	NoopResetEnv(noop_max=30)	NoopResetEnv(noop_max=30)
Episodic Life	EpisodicLifeEnv	EpisodicLifeEnv
Fire Reset	FireResetEnv (if needed)	FireResetEnv (if needed)
Observation Preprocessing	Grayscale + Resize (84x84) + FrameStack(4)	Grayscale + Resize (84x84) + FrameStack(4)
Reward Clipping	ClipRewardEnv (-1, 1)	ClipRewardEnv (-1, 1)
Sticky Actions (repeat_action_probability)	Not Used (Fixed Action Selection)	Enabled (0.25)

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. Table 1 show a comparison between our setup and v5.

### 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. Both parameters and hyperparameters are passed to the script as command-line arguments.

## 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 skewed 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. Among the first to use this approach were the authors of [2], from which we took the random policy and professional human player values used for normalization. The formula for obtaining the normalized value is:  $x_{\text{norm}} = \frac{x - x_{\text{random}}}{x_{\text{human}} - x_{\text{random}}}$ , where  $x$  is the agent score,  $x_{\text{random}}$  is the random policy score, and  $x_{\text{human}}$  is the professional human player score.

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

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 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 Run.** We grouped rows in the CSV by a specific tag (e.g., `charts/episodic_return`, `charts/SPS`) and by run.
2. **Interpolated to a Common Grid.** For each subset, we created a uniformly spaced array of steps (*i.e.*, 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*, *iqmean* 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. 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 inter-algorithm families and across them.

### 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 continuously. 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 by CodeCarbon.

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.

**Table 2:** Key hyperparameters for the Deep Q-Network baseline. Only `env_id` and `seed` change across runs.

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

**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 kept  $\tau$  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  $\vartheta$  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.

**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  $\sim$ 170 steps per second on average, though environment differences introduce some variance.

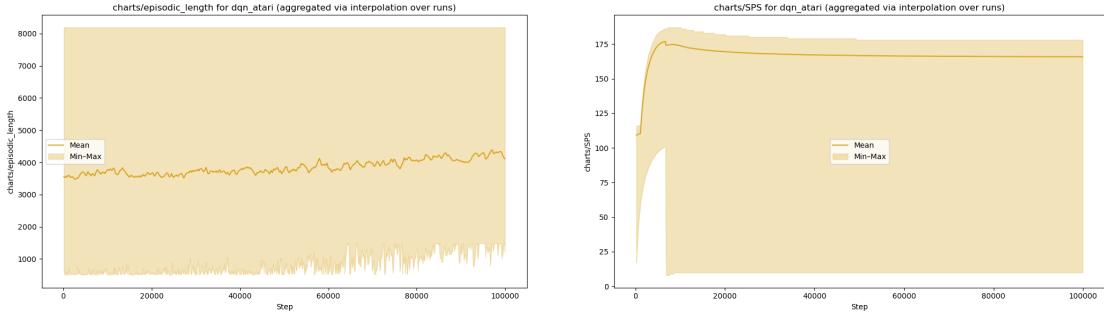
**Q-Values and TD Loss** Figures 1c and 1d show `losses/q_values` and `losses/td_loss`, respectively, across all runs. On average, Q-values increase steadily, suggesting the network’s estimates of future returns keep growing with experience. However, the broad min–max band indicates some seeds or games 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.

**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 16. Figures 2 and 3 aggregate these returns across all 32 runs, while Figures 4 and 5 show per-game curves.

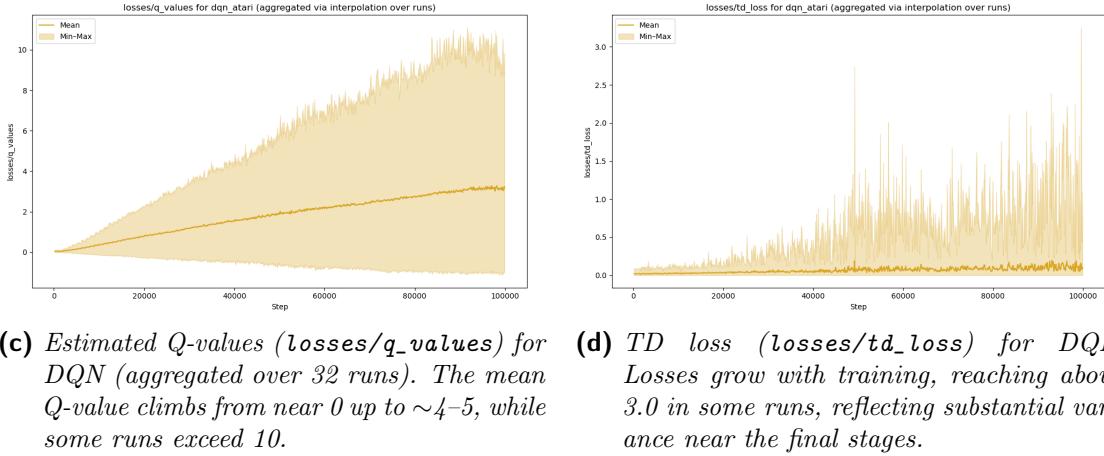
In the human-normalized plot, the mean hovers near zero, occasionally dipping negative due to poor performance on certain games. In the min–max plot, the average climbs from near 0.2 to around 0.4–0.5 by the end, indicating moderate relative progress.

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).

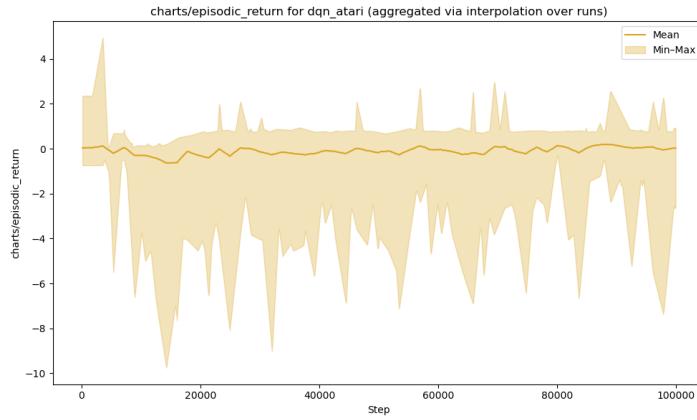
**Emissions** Table 3 presents the aggregated CO<sub>2</sub>-eq for DQN (over all 32 runs). The mean is about **0.00647 kg**, with a minimum of 0.00616 and a maximum near 0.0070.



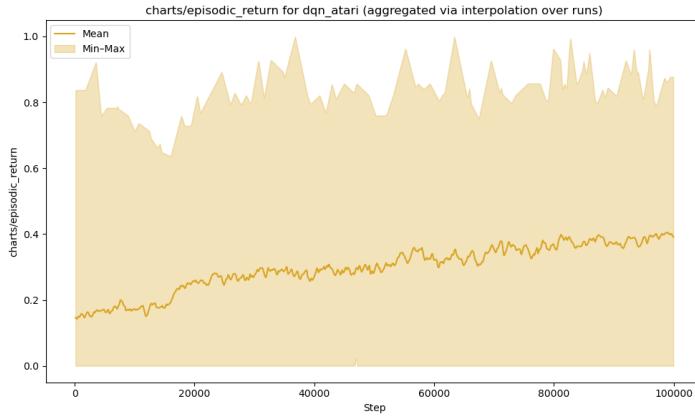
- (a)** Aggregated episodic length for DQN over 100k steps (interpolation across 32 runs). The mean hovers around 3500–4000 steps, while the min–max envelope extends from near 0 to over 8000.
- (b)** Steps per second (SPS) for DQN. After an initial ramp-up, the mean SPS stabilizes around 170–180, with some runs dipping as low as 20 or spiking above 200.



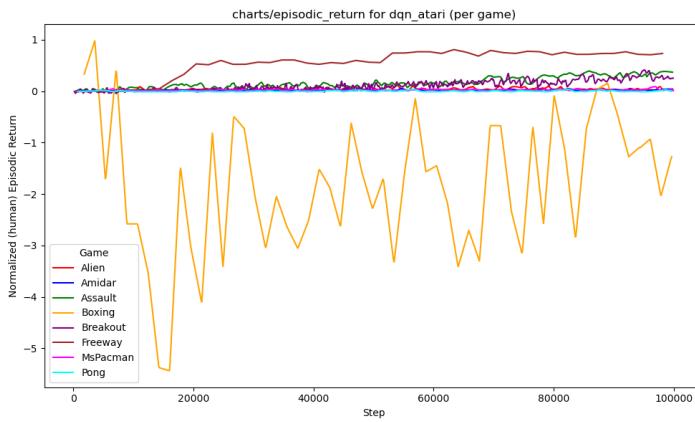
**Figure 1:** Performance metrics for DQN over 100k steps, aggregated across 32 runs.



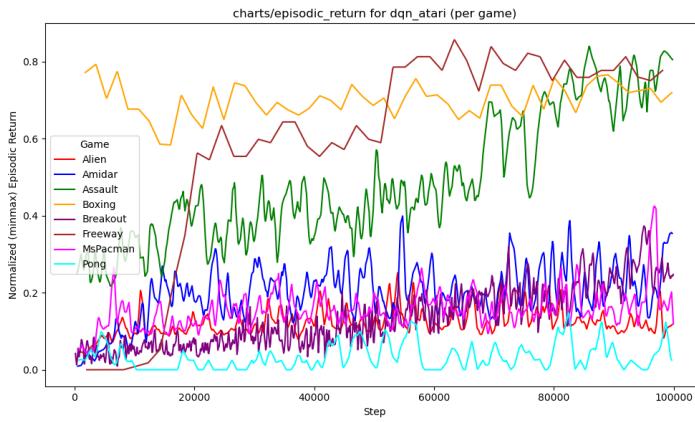
**Figure 2:** Aggregated DQN episodic return (human-normalized) over 100k steps. The shaded region represents min–max variation.



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



**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<sub>2</sub>eq) 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

**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<sub>2</sub>-eq per run. This is unsurprising for a 100k-step setting, 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.

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

### 4.2.2 Double DQN

Double DQN is one of the first and simpler tweaks made to DQN. It's simply the adaptation of the Double Q-learning algorithm, initially introduced in a tabular setting, to the Deep Reinforcement Learning setting. Double Q-Learning solves the problem of the maximization bias that afflicts Q-Learning using two different estimations of the action value function when constructing the TD-target: one to choose the action and another to evaluate it. Double DQN does the same thing, starting from DQN. In this environment the natural candidate for a second estimation of the action value function is the target network, so it uses the Q-Network to select the actions, and the target network to evaluate them. Therefore, the only difference in the implementation of the two algorithms is in the update rule. In the tabular case it goes from:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

where the max operator is the cause of the problem, to:

$$Q_1(S_t, A_t) = Q_1(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q_2(S_{t+1}, \text{argmax}_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right]$$

The roles of  $Q_1$  and  $Q_2$  can be reversed according to various schedules or rules, and similar approaches have been experimented with in the literature regarding DRL as well, but we have stuck with the original proposal that simply makes use of the target network.

**(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  $\times$  four seeds), while the rest remain unchanged. In particular, we again set `buffer_size=10k` and `learning_starts=1000`.

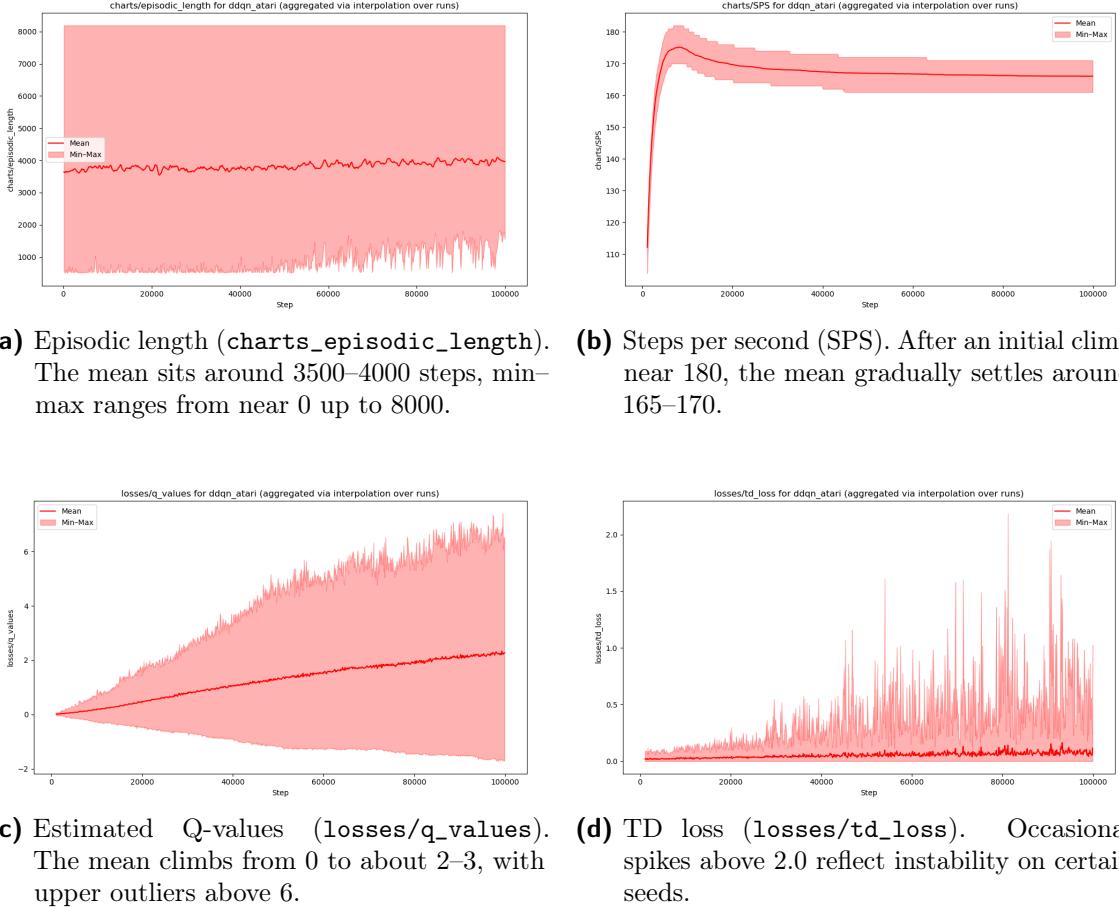
**Hyperparameter Tuning** To isolate the effect of Double DQN, we kept all settings identical to the baseline DQN, simply enabling the Double DQN update scheme. Following [4], we tested higher `target_network_frequency` (in particular the more promising one was 3000, scaled from the 10k–20k range in 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).

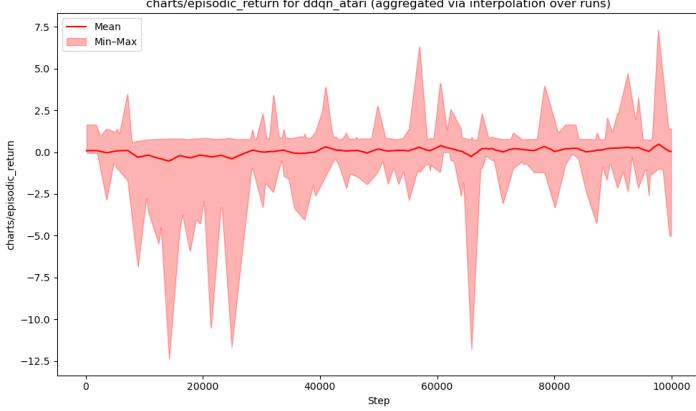
Episodic length and SPS curves are very similar to baseline DQN's (Section 4.2.1). Meanwhile, the mean Q-values grow more modestly than DQN's (which often exceed 4–5 by the end), confirming that Double DQN's approach does mitigate the overestimation (see figure 11 on page 31 for a direct comparison). TD loss remains low overall, though some runs spike above 2.0 near late training.

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

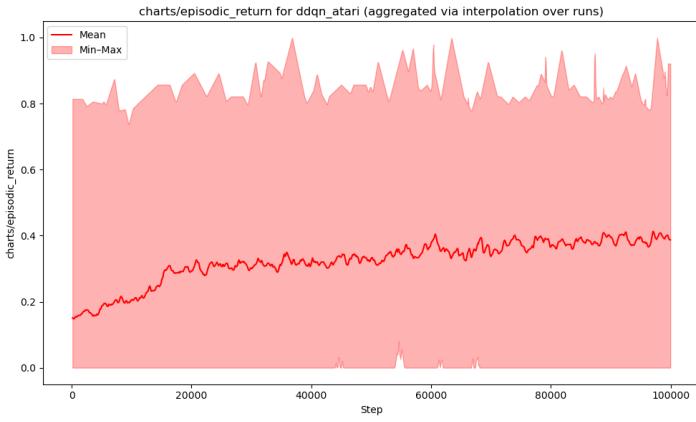
Parameter	Value
<code>exp_name</code>	<code>ddqn_atari</code>
<code>seed</code>	1..4
<code>torch_deterministic</code>	True
<code>cuda</code>	True
<code>track</code>	True
<code>wandb_project_name</code>	<code>rlsb</code>
<code>capture_video</code>	False
<code>save_model</code>	True
<code>upload_model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



**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.

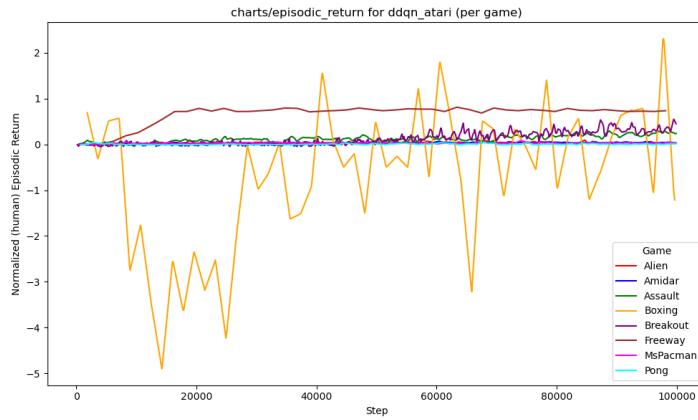
**Episodic Return (Human vs. Min–Max Normalized)** Figures 7 and 8 show Double DQN’s aggregated episodic returns (human- and min–max-normalized, respectively). Figures 9 and 10 break these results down by game.

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  $\sim 0.2$  to  $\sim 0.4$ , similar to DQN’s overall trajectory.

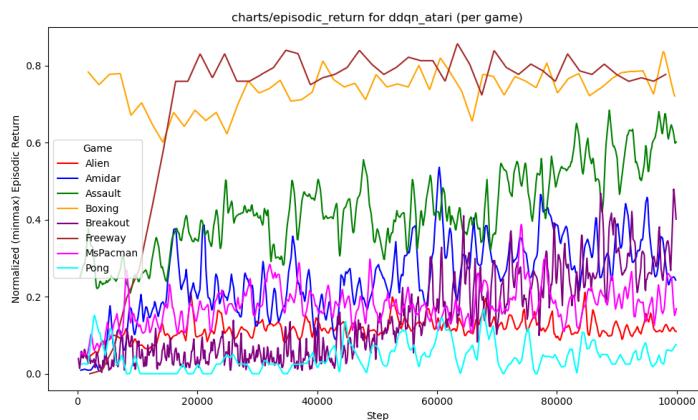
**Emissions** Table 7 summarizes Double DQN’s CO<sub>2</sub>-eq emissions across 32 runs. The mean is about **0.00667 kg**, slightly above DQN’s  $\sim 0.00647$ .

**Table 7:** Carbon emissions (kg CO<sub>2</sub>eq) 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



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

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

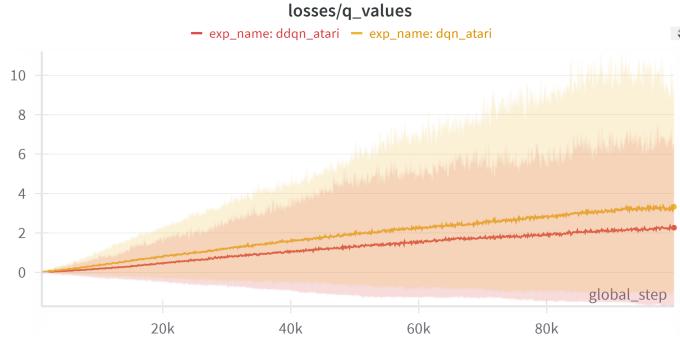
**Table 9:** Per-game final evaluation for Double DQN (human- vs. min–max normalized). Each row aggregates 40 total episodes (10 per seed).

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

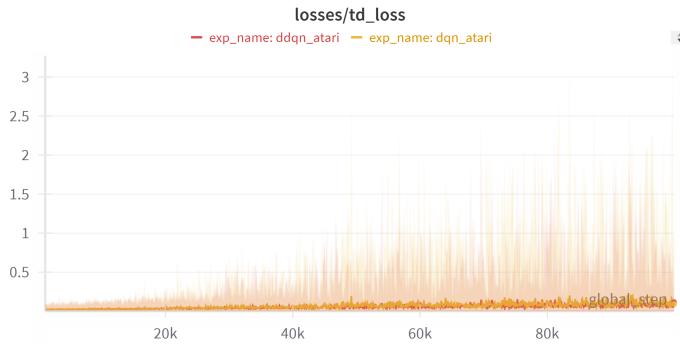
**Evaluation Results** Table 8 compiles final returns (human-/min–max normalization) aggregated over the 32 runs. Compared to DQN’s  $\sim 0.135$  (human) and  $\sim 0.380$  (min–max), Double DQN attains 0.023 (human) and 0.374 (min–max). While the min–max average is comparable, the human-normalized 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



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



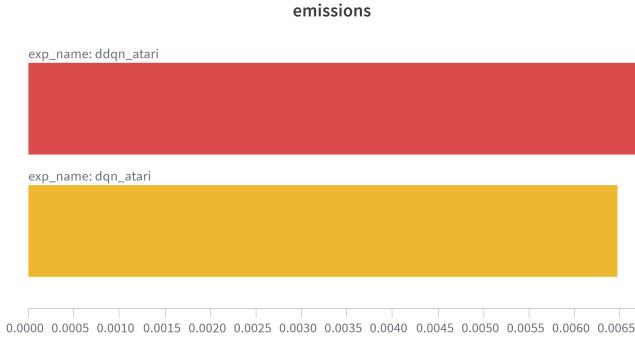
**Figure 12:** Comparison of TD loss for DQN (gold) vs. Double DQN (red). Both remain near 0 for extended periods, though DQN shows slightly higher spikes.

gold; Double DQN’s mean Q-values grow more slowly, suggesting less overestimation. Figure 12 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.
- **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  $\sim 0.00667$  kg CO<sub>2</sub>-eq, slightly higher than DQN’s 0.00647 kg.



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

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

In a standard DQN replay buffer, transitions are sampled *uniformly*, giving equal probability to each experience. Prioritized Experience Replay (PER) [5] seeks to allocate more sampling probability to transitions with higher TD error, on the premise that they contain more learning signal for the agent, thus achieving an effect similar to that of prioritized sweeping in the value iteration algorithm for the tabular case. With this focus on “informative” samples, also inspired by biology, PER can potentially accelerate training and reduce sample complexity, especially in long training horizons.

The original paper proposes two primary methods to implement prioritization:

- *Sum-Tree* approach, which stores priority values  $p_i$  in a binary tree, allowing for efficient sampling of transitions proportional to  $p_i^\alpha$ .
- *Rank-Based* approach, which sorts the buffer by TD error magnitude and samples according to rank orders.

We adopt the *sum-tree* variant in our implementation, mirroring the method described in [5] for improved efficiency over naive priority queues. Apart from this change, the underlying DQN hyperparameters (e.g., network architecture, target frequency) remain the same, and we incorporate importance-sampling corrections as recommended to offset the sampling bias introduced by prioritization.

We also explored two implementations of the PER replay buffer—a `numpy`-based (for consistency with the replay buffer used for the other algorithms) and a `torch`-based version. The `numpy` approach proved much slower, so we used the `torch` one in the final runs. It should be noted that even though the other DQN variants use a replay buffer `numpy`-based, it is the one implemented in Stable Baselines, so it employs

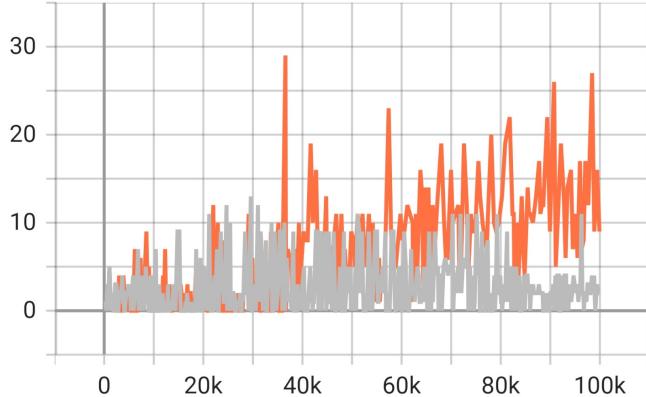
**Table 10:** Key hyperparameters for Prioritized Experience Replay (PER). Only `env_id` and `seed` vary across runs.

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

optimizations that our version lacked. For this reason, we assume that using the torch-based implementation does not invalidate direct comparisons.

**(Hyper)Parameters** Table 10 summarizes the key hyperparameters used in our PER implementation. Only `env_id` and `seed` vary across runs (again, eight Atari games  $\times$  four seeds = 32 runs), while other settings match those of the baseline DQN (Section 4.2.1) to isolate PER’s effects.

**Hyperparameter Tuning** We retained the same settings as baseline DQN (Section 4.2.1) to highlight PER’s impact alone, an approach mostly followed by the original authors too. Although [5] suggests lowering the learning rate by a factor of four, our tests at 100k steps with it and other values showed again that  $1 \times 10^{-4}$  worked better. Figure 14 compares these rates on *Breakout*, indicating faster convergence at  $1 \times 10^{-4}$ .



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

**Table 11:** Carbon emissions (kg CO<sub>2</sub>eq) for PER across 32 runs.

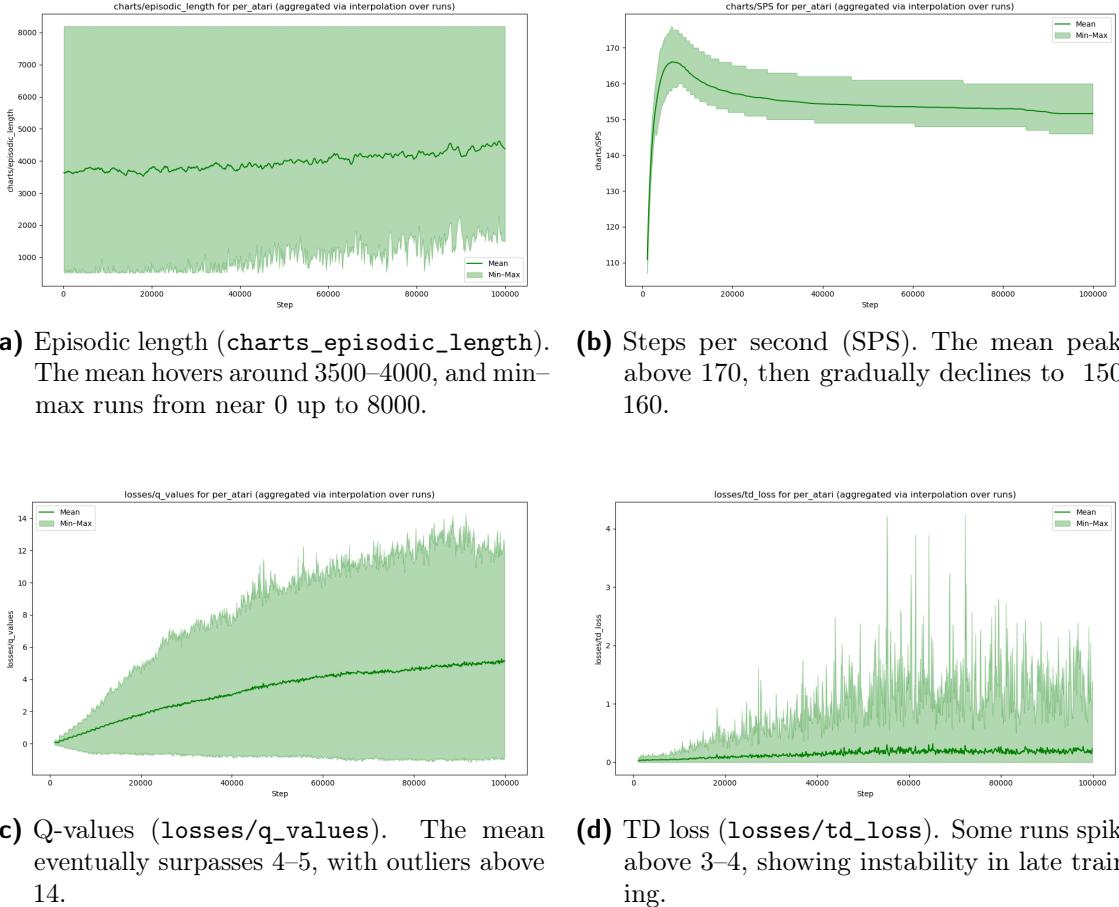
Algorithm	mean	std	median	q25	q75	min	max	iqmean
PER	0.007254	0.000263	0.007146	0.007074	0.007354	0.006935	0.007819	0.007157

**Training Dynamics** Figure 15 on the next page 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.

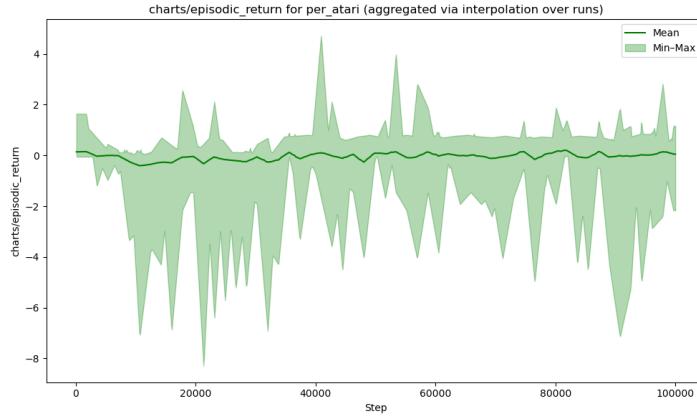
**Episodic Return** Figures 16 (human-normalized) and 17 (min–max) show PER’s aggregated episodic returns. The mean in human-normalized scale oscillates around zero, occasionally dipping below –2 or –3 in some seeds.

**Per-Game Returns** Figures 18 and 19 on page 36 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.

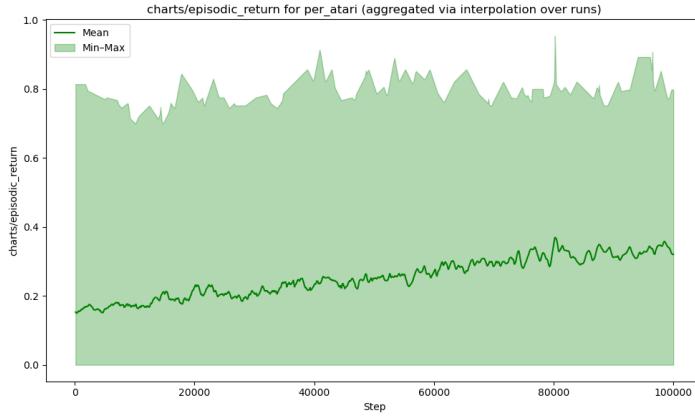
**Emissions.** Table 11 shows the statistics of the emissions of PER, while Figure 20 on page 37 shows a barplot comparing PER’s mean emissions ( $\approx 0.00725$  kg) to DQN’s ( $\approx 0.00647$  kg). The overhead of prioritized sampling may partly account for this higher footprint.



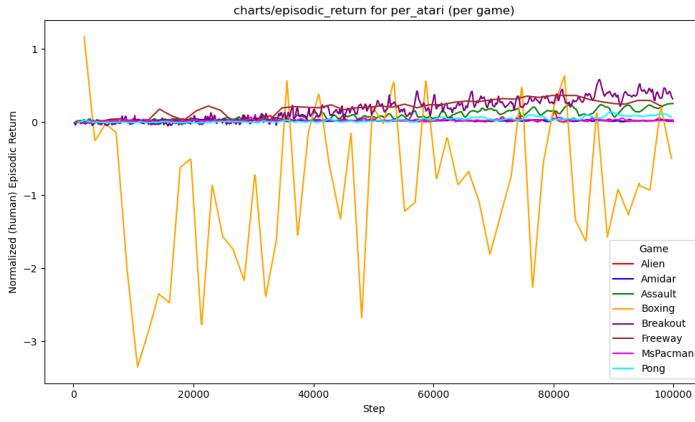
**Figure 15:** PER training metrics over 100k steps, interpolated across 32 runs.



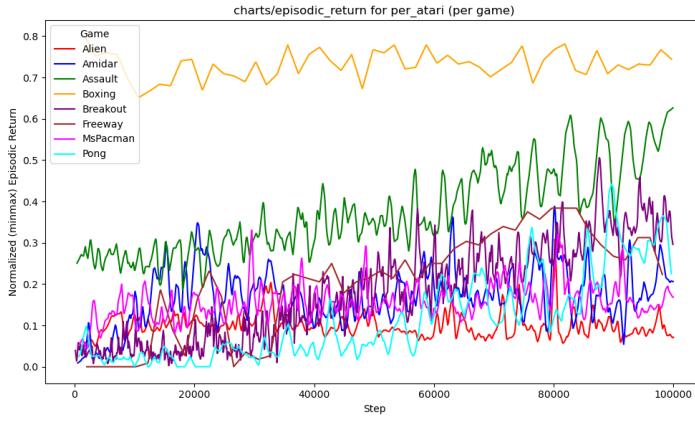
**Figure 16:** PER episodic return (human-normalized), aggregated over 32 runs. Negative outliers appear for certain seeds/environments.



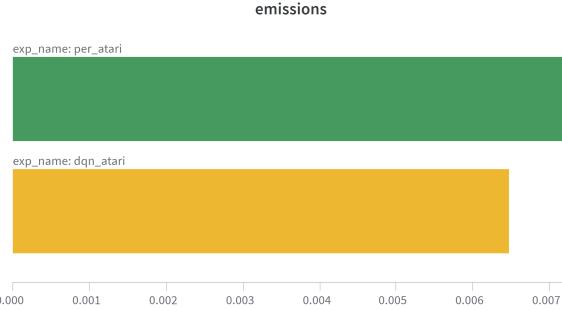
**Figure 17:** PER episodic return (min–max normalized), aggregated over 32 runs.



**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.



**Figure 20:** Mean emissions comparison: PER (green) at  $\sim 0.00725$  kg vs. DQN (gold) at  $\sim 0.00647$  kg.

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

Normalization	mean	std	median	q25	q75	min	max	iqmean
<b>Human</b>	0.0607	1.0170	0.0539	0.0175	0.2541	-10.2619	6.8809	0.0813
<b>Min–Max</b>	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  $\times$  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  $\sim 0.0008$  kg CO<sub>2</sub>-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  $1 \times 10^{-4}$  (Figure 14 on page 34).
- **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<sub>2</sub>-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

The *Dueling DQN* [6] architecture modifies the final layers of the Q-network to separate the estimation of the state-value function  $V(s)$  from that of the advantage function  $A(s, a)$ , defined as  $A(s, a) = Q(s, a) - V(s)$ . These two *heads*, termed *streams* in the paper, are then combined to yield

$$Q(s, a) = V(s) + (A(s, a) - \mathcal{B})$$

where  $\mathcal{B}$  is a sort of "baseline" (see [6] for the details on why this is necessary). The authors of the paper proposed either  $\mathcal{B} = \max_{a' \in A(s)} A(s, a')$  or the *average*  $\mathcal{B} = \frac{1}{|\mathcal{A}|} \sum_{a' \in A(s)} A(s, a')$ , but mainly experimented with the latter, so we did the same, as did most of the subsequent literature.

This approach allows the network to learn which states are valuable (*regardless* of action) and which actions are relatively better than others (*given* a state), thereby stabilizing learning in states where multiple actions have a close Q-value, and improving sample efficiency in many environments.

**(Hyper)Parameters** Adopting the same base configuration as our DQN baseline (Section 4.2.1), we set `buffer_size`=10 000 and `learning_starts`=1000 to accommodate the 100k-step regime. Table 14 summarizes the key hyperparameters used in our Dueling DQN implementation. As with the baseline DQN, the `env_id` and `seed` parameters were

varied across runs to ensure statistically significant results across different games and random initializations.

Because the dueling architecture introduces separate streams for  $V(s)$  and  $A(s, a)$ , the model ends up with slightly more parameters than the baseline and other DQN variants. We considered reducing the size of the last shared layer to keep the overall parameter count equal to the baseline, but being the difference only about 512 weights, we opted to retain the original layer size for consistency with the other tested DQN variants.

The original Dueling DQN work introduced gradient clipping to maintain training stability, since there were now the gradients from two different stream that merged in the common layers. However, our tests indicated no significant difference with or without gradient clipping, suggesting sufficient inherent stability in our setup. Therefore, we chose to omit it for consistency across all algorithms and to avoid introducing unnecessary modifications.

Finally, to maintain a controlled comparison, we did not combine dueling with Double DQN or PER (although that was explored in [6]), as our primary goal is to evaluate the individual contribution of this single architectural tweak to performance and emissions, in line with our methodology for all DQN variants.

**Training Dynamics** Figure 21 displays key training metrics aggregated over 32 runs (8 games  $\times$  4 seeds).

As shown in Figure 21a, the episodic length generally hovers between 3500 and 4000 steps, exhibiting variability with minimums near 0 and maximums reaching up to 8000. The steps-per-second (SPS) metric, depicted in Figure 21b, shows an initial ramp-up phase before stabilizing around 155-160, slightly below the baseline DQN’s average of approximately 165-170.

Q-values (Figure 21c) generally ascend beyond 4-5, with some outliers reaching 8-10, a trend consistent with standard DQN but with slightly lower peak values than Double DQN or PER.

The TD loss (Figure 21d) remains relatively low, generally staying below 1.5-2.0, with some runs exhibiting increased variability later in training.

**Episodic Return (Aggregated)** Figures 22 and 23 show the average episodic returns over 100k steps, in *human-normalized* and *min-max normalized* scales, respectively.

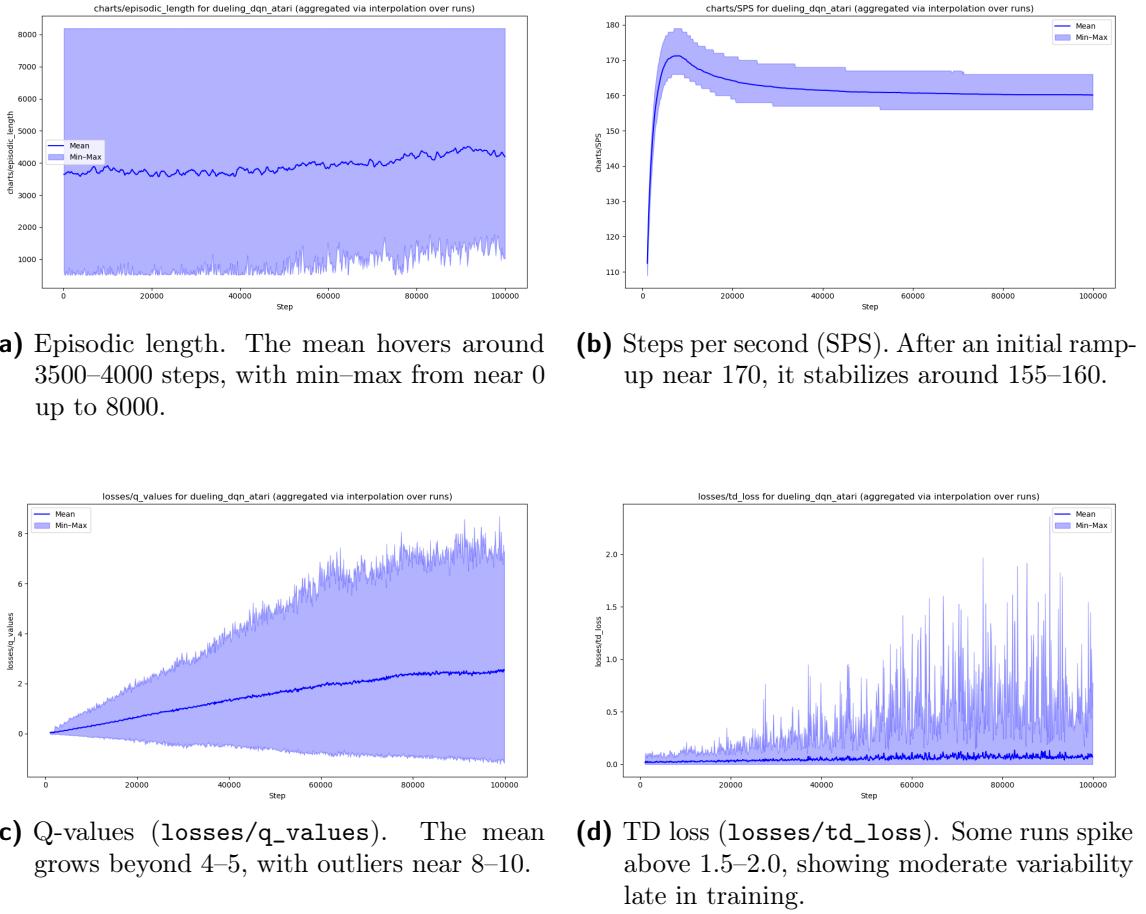
As seen in Figure 22, the human-normalized returns exhibit significant variance, with occasional dips below -8. The mean trend hovers around 0.1-0.2, slightly surpassing the baseline DQN’s typical value, but is influenced by the volatility of results in environments like *Boxing*.

Min-Max Normalized curve, in (Fig. 23) scales near 0.1-0.2 to 0.35-0.45, which equals to what see for the baseline implementation.

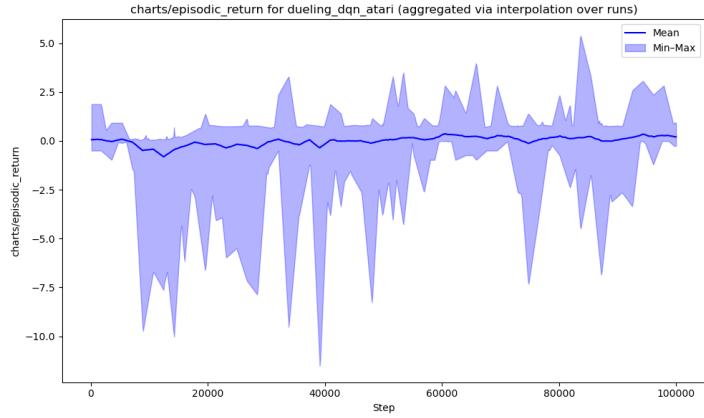
**Per-Game Returns** Figures 24 and 25 provide a detailed breakdown of returns for each of the 8 Atari games: On Figure 24, with the human normalized metrics, you can see that *Boxing* (orange) can swing from  $-1.93$  up to  $+3.55$ , while *Freeway* and *Assault*

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

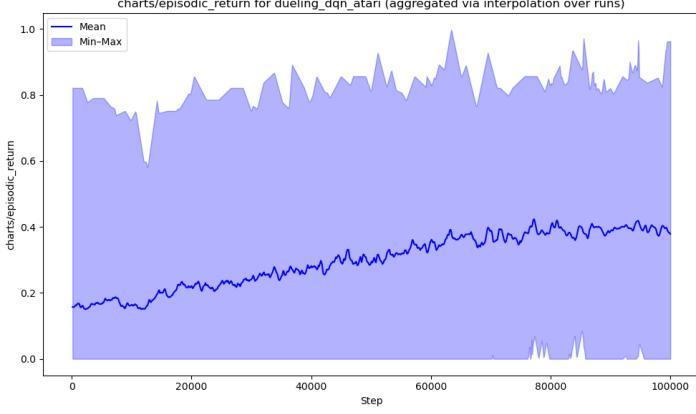
Parameter	Value
<code>exp_name</code>	<code>dueling_dqn_atari</code>
<code>seed</code>	1..4
<code>torch_deterministic</code>	True
<code>cuda</code>	True
<code>track</code>	True
<code>wandb_project_name</code>	<code>rlsb</code>
<code>capture_video</code>	False
<code>save_model</code>	True
<code>upload_model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



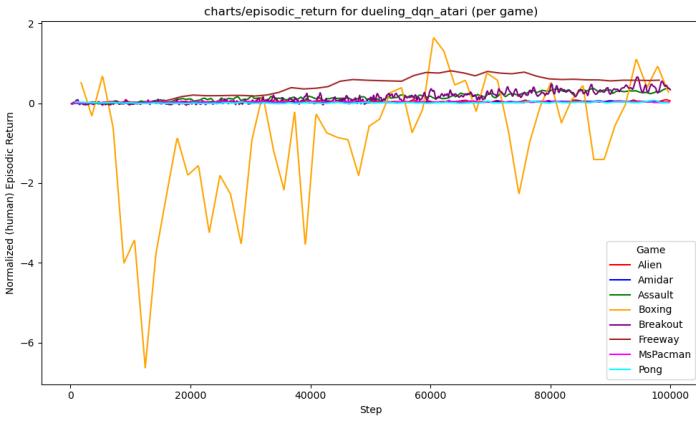
**Figure 21:** Dueling DQN training metrics over 100k steps, aggregated over 32 runs.



**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.



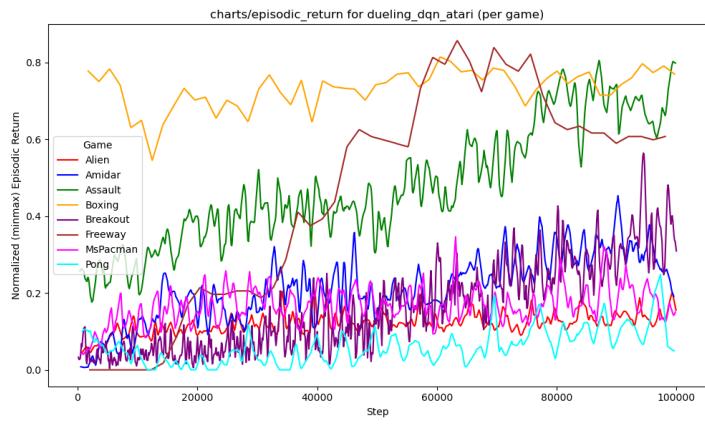
**Figure 24:** Dueling DQN returns per game (human-normalized). *Boxing* (orange) experiences deep negative dips, while *Freeway* and *Assault* can reach higher normalized scores.

can reach higher normalized scores. Figure 25 demonstrates that, considering the min max normalization, *Freeway* often pushes above 0.7, *Assault* similarly reaches 0.7–0.8.

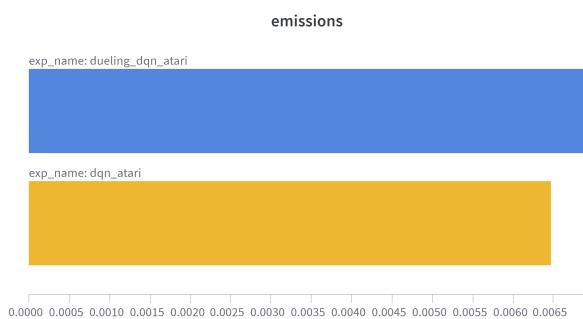
**Emissions** Table 15 presents the aggregated carbon footprint: a mean of 0.006893 kg CO<sub>2</sub>-eq, with a standard deviation of 0.00029. This is higher than the baseline DQN’s ~ 0.00647 kg but still below some others (e.g., PER’s ~ 0.00725 kg). Figure 26 illustrates a direct comparison between Dueling DQN and baseline DQN.

**Table 15:** Carbon emissions (kg CO<sub>2</sub>eq) for Dueling DQN across 32 runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmean
Dueling DQN	0.006893	0.0002901	0.006742	0.006672	0.007002	0.006617	0.007478	0.006779



**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  $\sim 0.00689$  kg vs. DQN (gold) at  $\sim 0.00647$  kg.

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

Normalization	mean	std	median	q25	q75	min	max	iqmean
<b>Human</b>	0.1860	0.5258	0.0402	0.0148	0.3530	-1.9286	3.5476	0.1020
<b>Min–Max</b>	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

**Evaluation Results** Table 16 reports the overall final evaluation (10 episodes each) for Dueling DQN across all 32 runs, while Table 17 provides a game-by-game breakdown. Analyzing Table 16, one can see that in human normalized version we obtain some numbers which, given what we found for dqn or double dqn, seem high (0.5 in freeway), but we also see the value for boxing pulling down the other score. The numbers from min max are in line with similar algorithms, with the freeway ones again on the lead.

**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  $\sim 0.00689$  kg CO<sub>2</sub>-eq are higher than DQN’s  $\sim 0.00647$  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  $\sim 0.00689 \text{ kg CO}_2\text{-eq}$ , more than DQN’s 0.00647 but less than PER’s 0.00725.

In terms of performance, Dueling DQN demonstrates a small improvement in human-normalized returns compared to the baseline DQN, while maintaining a similar emissions profile, as shown by the evaluation results.

Overall, **Dueling DQN** yields a small performance boost compared to baseline DQN within 100k steps, at a marginally higher carbon cost, consistent with the idea that decoupling  $V(s)$  from  $A(s, a)$  can help focus updates in states where actions yield similar Q-values.

### 4.2.5 Categorical DQN (C51)

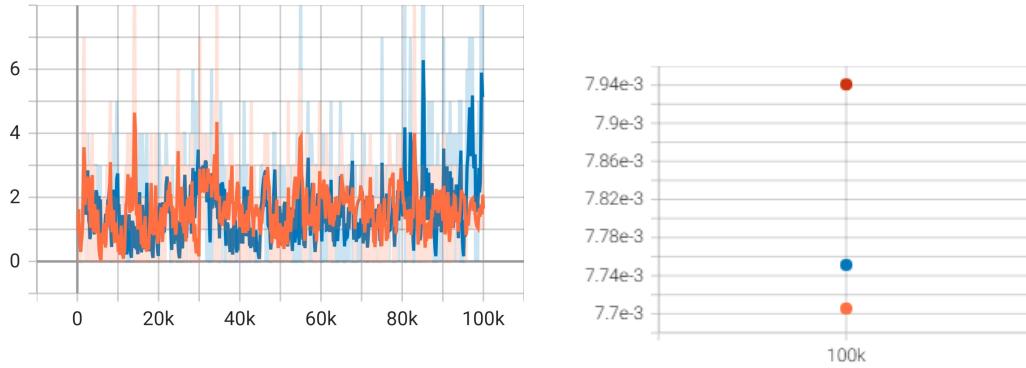
Categorical DQN (C51) [8] extends the traditional DQN framework by modeling the action-value function  $Q(s, a)$  as a full (discrete) probability distribution over possible returns rather than just a single expected value. This distributional approach enables the agent to capture not only the mean reward but also the variability and uncertainty associated with future returns. C51 achieves this by discretizing the range of potential returns, spanning from  $v_{\min}$  to  $v_{\max}$ , into a fixed set of support points (atoms).

During the learning process, the algorithm projects the Bellman update onto this discrete support and uses a cross-entropy loss to align the predicted distribution with the target distribution. Such a rich representation of returns is particularly beneficial in risk-sensitive or high-variance environments, as it provides more informative learning signals that can lead to improved stability and performance over longer training periods. Although this enhanced representation introduces additional computational overhead, as reflected by the increased emissions compared to the baseline DQN, it holds promise for more nuanced decision-making in complex tasks.

**(Hyper)Parameters** The key hyperparameters used for the C51 implementation are summarized in Table 18. In this configuration, most settings remain constant across runs, with only the environment identifier (`env_id`) and the random seed (`seed`) varying. Notably, C51 employs a discrete return distribution with 51 atoms spanning the interval  $[-10, 10]$ , and uses a learning rate of 0.00025. Other parameters such as the total timesteps (100 000), batch size (32), and target network update frequency (1000) have been kept consistent with the other tested algorithm.

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

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



**(a)** C51 raw episodic return on Breakout with a `target_network_frequency` of 1000 (blue) and 10 000 (orange)  
**(b)** Emissions at 100k steps, `target_network_frequency` = 1000 (blue), 5000 (red) 10 000 (orange)

**Figure 27:** Tuning of `target_network_frequency`.

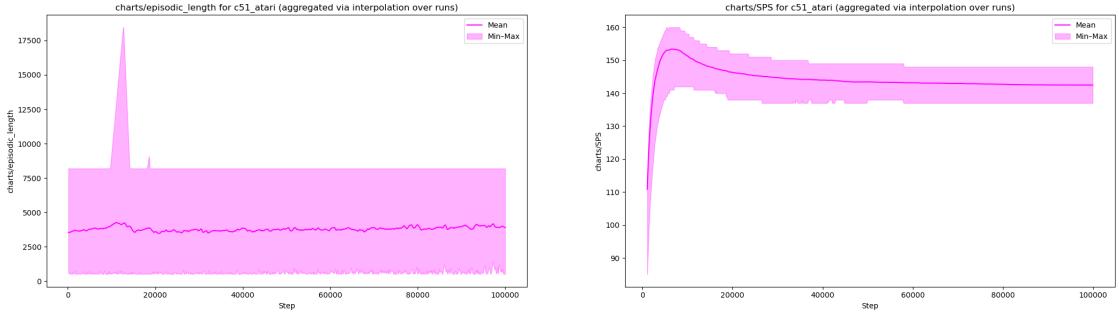
**Hyperparameter Tuning** We adopted CleanRL’s `c51_atari` configuration with 51 atoms,  $v_{\min} = -10$ ,  $v_{\max} = 10$ , and a learning rate of  $2.5 \times 10^{-4}$ . While we tested alternative rates, this default proved effective over only 100k steps.

Following the original work, we also experimented with various `target_network_frequency` values, between them 1000, 5000, and 10 000 (see figure 27a), the last one being the cleanrl default. Ultimately, once again 1k proved to be the best value, providing a good balance of stable returns and moderate emissions (figure 27b), aligning with other DQN-based variants.

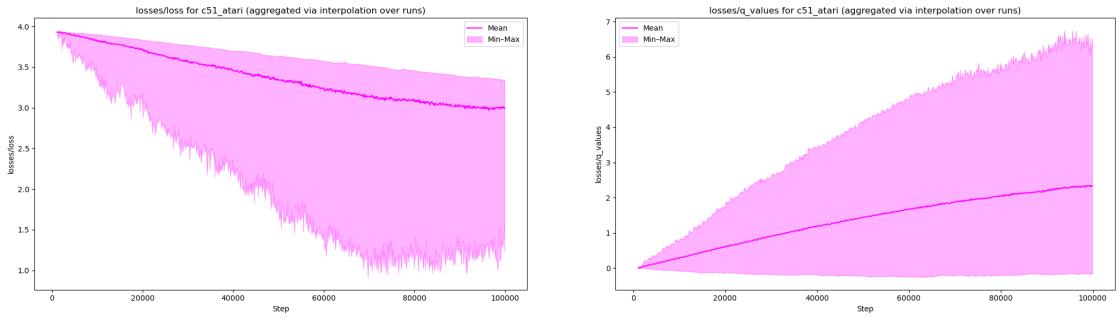
**Training Dynamics** Figure 28 on the next page shows that certain runs produce extremely long episodes (Figure 28a), while the overall SPS curve (Figure 28b) hovers around 140–150 later in training, slightly lower than baseline DQN’s  $\sim 160$ . The training loss (Figure 28c) descends from near 4.0 to 2.0, while distributional `q_values` (Figure 28d) broaden significantly.

**Episodic Return (Aggregated)** Due to extreme negative performance in certain games, such as *Boxing*, the human-normalized return (Figure 29 on the following page) often falls below -1. In min–max (Figure 30 on page 49), the mean approaches  $\sim 0.25$ –0.30 by 100k steps.

**Per-Game Returns** Figure 31 presents the human-normalized episodic returns for C51 on a per-game basis. Notably, the game *Boxing* exhibits extremely negative returns—with values plunging below -12, significantly impacting the overall human-normalized performance. In contrast, games such as *Alien* and *Amidar* show relatively stable and modest returns. Similarly, Figure 32 displays the min–max normalized returns, where *Assault* reaches values above 0.5, while *Pong* remains consistently near zero. These per-game

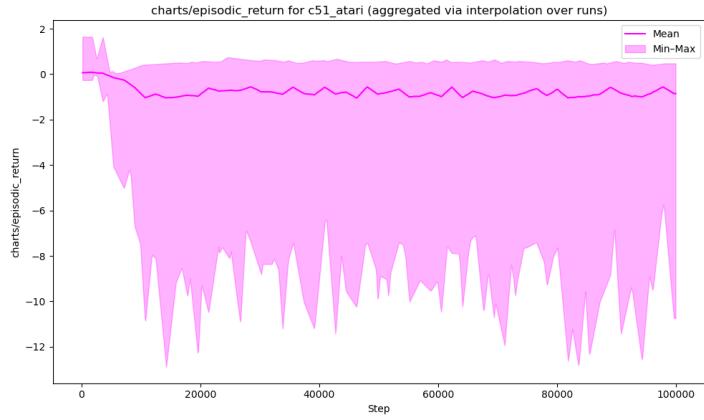


- (a)** Episodic length (`charts/episodic_length`). The mean is  $\sim 3500\text{--}4000$ , while min–max occasionally spikes above 8000 or even 17500 in some runs.
- (b)** Steps per second (SPS). Mean peaks around 160–170, then slowly converges near 140–150.

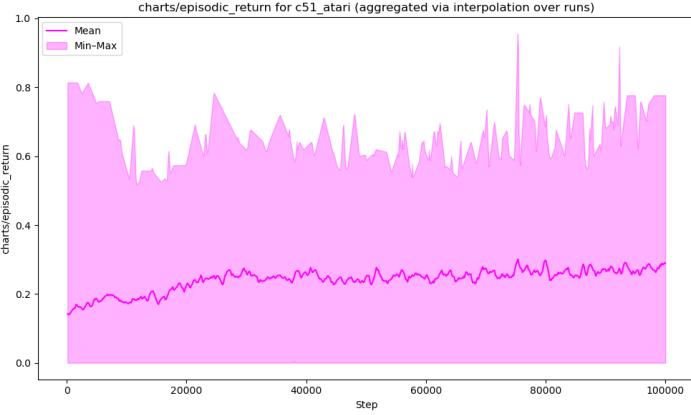


- (c)** Overall loss (`losses/loss`). The mean steadily declines from about 4.0 toward near 2.0 by the end of training.
- (d)** Q-values (`losses/q_values`). The mean climbs from near 0 to  $\sim 4\text{--}5$ , with outliers reaching 6–7.

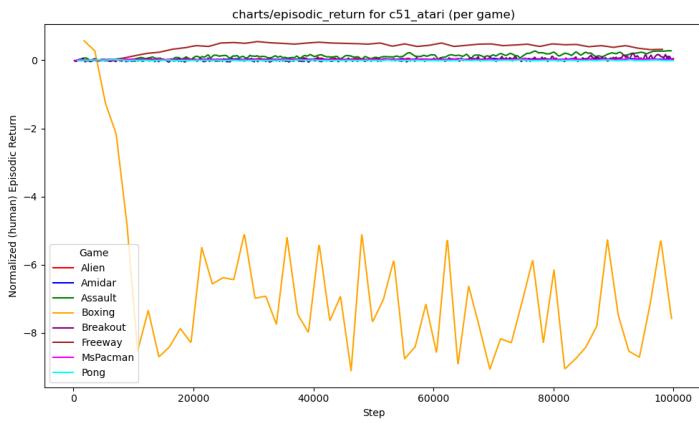
**Figure 28:** C51 training metrics over 100k steps, aggregated (via interpolation) across 32 runs.



**Figure 29:** C51 episodic return (human-normalized), aggregated over 32 runs. Negative scores dominate, especially due to *Boxing*'s steep dips below -10.



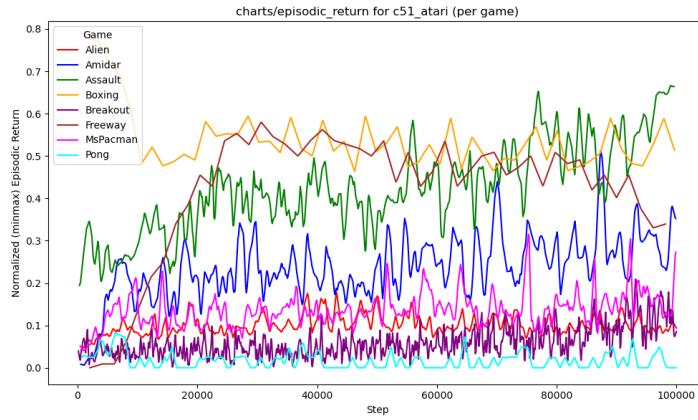
**Figure 30:** C51 episodic return (min–max normalized). The mean grows toward 0.25–0.30, indicating moderate performance relative to each game’s min–max range.



**Figure 31:** C51 returns per game (human-normalized). *Boxing* (orange) can plunge below -12, dwarfing improvements in other games.

analyses underline the variability of C51’s performance across different environments, emphasizing the importance of detailed game-specific evaluations.

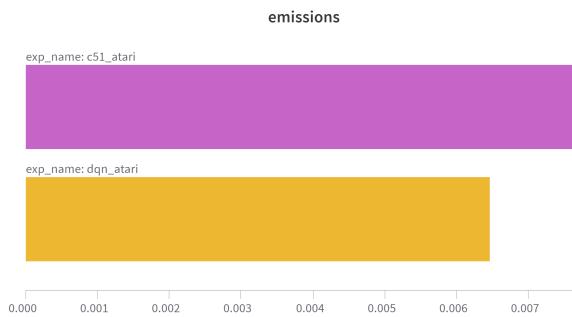
**Emissions** Table 19 details the carbon emissions recorded for C51 across the 32 runs, showing a mean emission of approximately 0.00775 kg CO<sub>2</sub>eq. This value exceeds that of the baseline DQN, which recorded mean emissions of 0.00647 kg CO<sub>2</sub>eq. The additional computational overhead in C51 arises from the need to compute and update a 51-atom probability distribution for the return at each training step, a process that likely increases GPU utilization. Although the absolute emissions remain relatively modest, the consistent increase across runs highlights an important trade-off: while the distributional approach of C51 may offer richer representations of return variability, it does so at the cost of higher energy consumption. This observation is particularly relevant in contexts where energy efficiency and environmental impact are critical considerations.



**Figure 32:** 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<sub>2</sub>eq) 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 33:** Mean emissions: C51 (magenta) at  $\sim 0.00775$  kg vs. DQN (gold) at  $\sim 0.00647$  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	$5.27 \times 10^{-18}$	-0.01	-0.01
	Min–Max	0.0	0.0	0.0	0.0

**Evaluation Results** The overall final evaluation of C51, conducted over 10 episodes per run and aggregated across 32 runs, is summarized in Table 20. On the human-normalized scale, C51 achieves a mean return of -1.0811 with a high standard deviation of 3.2862 and a range from -12.8810 to 0.7770, reflecting significant variability driven by outlier performances (notably in *Boxing*). In contrast, min–max normalization yields a mean of 0.2503 with a smaller standard deviation of 0.2568 and a range from 0.0 to 0.8270, indicating a more balanced distribution of performance scores. Furthermore, Table 21 provides a per-game breakdown, where for example *Boxing* registers a human-normalized mean of -9.3869, while other games such as *Alien* and *Amidar* exhibit near-neutral scores. These results offer a nuanced perspective on C51’s performance across different games and normalization methods.

**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 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 ( $\sim 0.00775$  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  $\sim 0.00775$  kg CO<sub>2</sub>-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.

**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<sub>2</sub>-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.

Algorithm	Final Episodic Return (Mean)		Emissions
	Human Norm	Min–Max	(kg CO <sub>2</sub> -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

**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 <sub>2</sub> )	
	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

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

## 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.

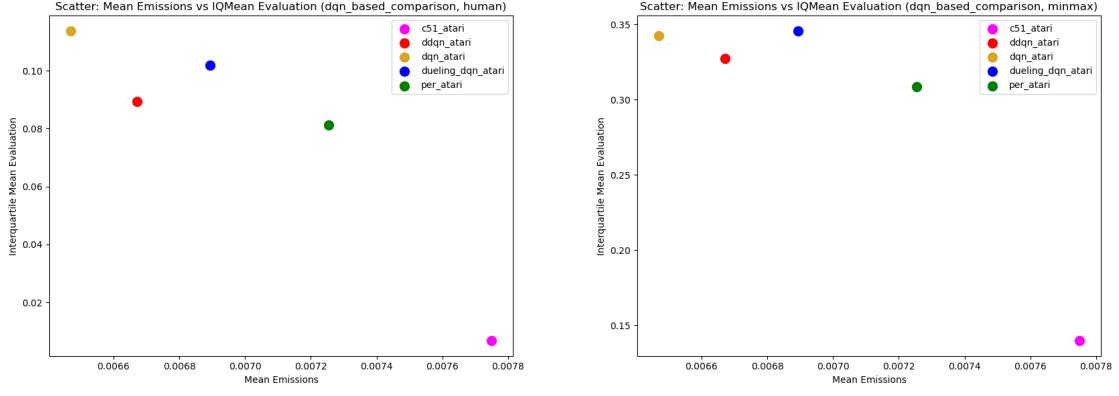
- **PER** does not strongly outperform DQN in short (100k-step) training, scoring 0.0607 human-norm and 0.3533 min–max, with slightly higher emissions (0.00725 kg). Prioritizing TD errors may show more benefit in longer runs.
- **C51** exhibits the largest negative dip on human-norm ( $-1.0811$ ), largely 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).

## Discussion.

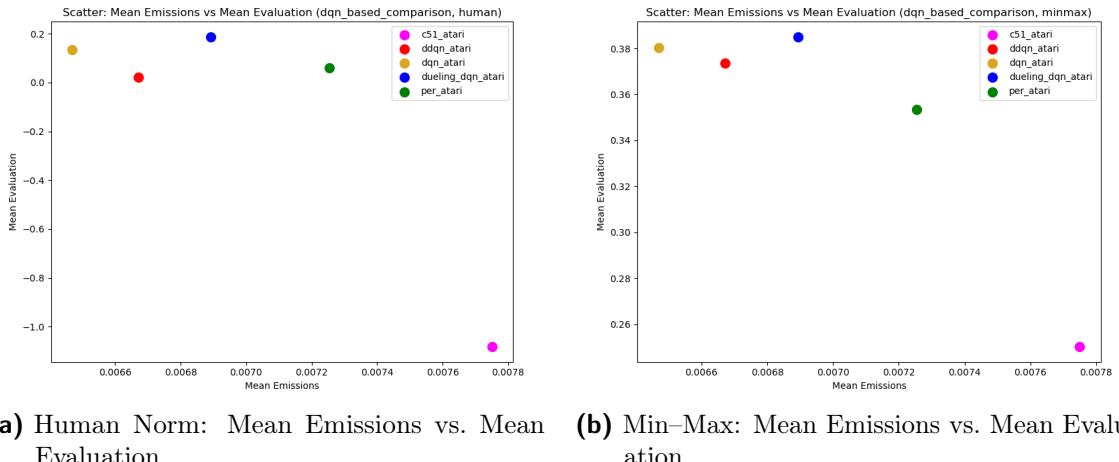
- **Human-Norm vs. IQM.** Some methods (e.g., Double DQN, C51) exhibit a large discrepancy between the mean and IQM in human-normalized returns, indicating a handful of extreme outliers (often due to specific games like *Boxing*).
- **Dueling DQN.** Its mean is highest in human-norm (0.1860), but the 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 only a few seeds/games are catastrophic 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.

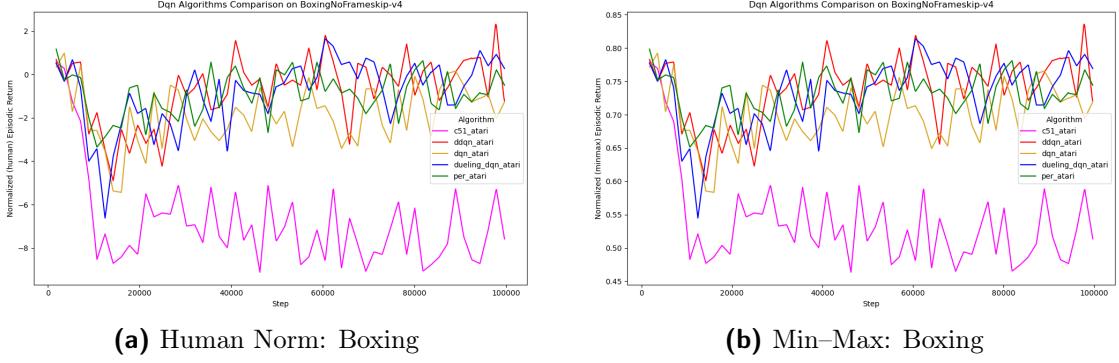
**Scatter Plots of Emissions vs. Performance** To visually depict the trade-off between carbon footprint and final performance, Figures 34 and 35 show scatter plots of Mean Emissions on the x-axis against (i) the IQMean or (ii) the Mean of final evaluation on the y-axis. We plot both the human-normalized and min–max normalized variants.



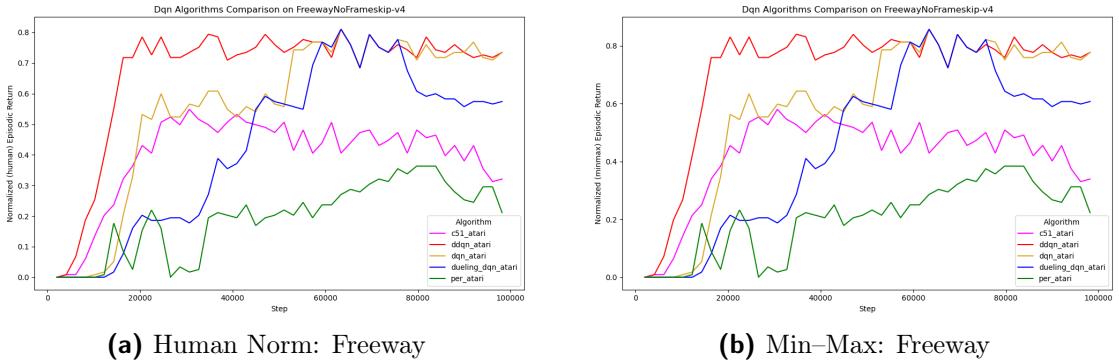
**Figure 34:** Scatter: Mean Emissions vs. Interquartile Mean (IQM) final evaluation, for DQN-based algorithms. C51 (magenta) appears far to the right (highest emissions) and near the bottom in human norm, though it's closer in min–max. DQN and Double DQN cluster with relatively low emissions.



**Figure 35:** 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.



**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.

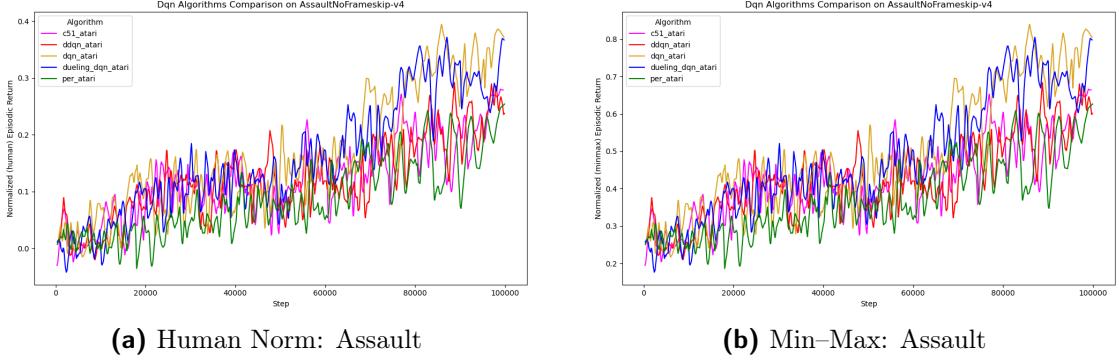


**Figure 37:** Comparison of DQN-based algorithms on *Freeway*. All methods converge relatively quickly to near-human or above. PER and Dueling typically perform strongly on this environment.

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

- **Boxing** (figure 36) heavily skews human-normalized averages for PER and especially C51, yielding strongly negative means. Dueling or Double DQN often handle *Boxing* more stably.
- **Freeway** (figure 37) is comparatively easier, so all variants converge to near-human or above. PER and Dueling both score well here.
- **Assault** (figure 38 on the following page) 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.

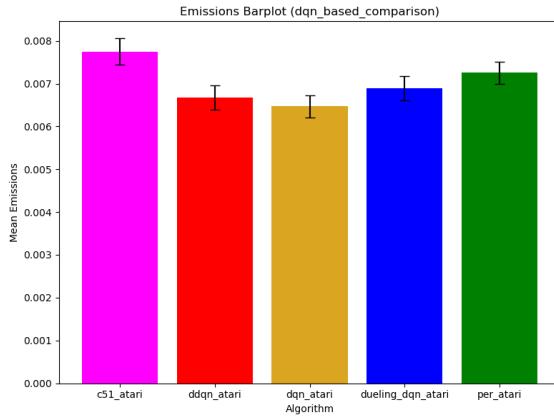
**Emissions and Efficiency** As indicated by both Table 22, Table 23, and the scatter plots (Figures 34–35), **C51** has the highest mean emissions ( $\sim 0.00775$  kg), while **PER**



**Figure 38:** Comparison of DQN-based algorithms on *Assault*, showing human-normalized (*left*) and min–max normalized (*right*) final returns over 100k steps. Distributional methods (C51) can excel in some seeds but not enough to surpass Dueling/DQN on average.

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.

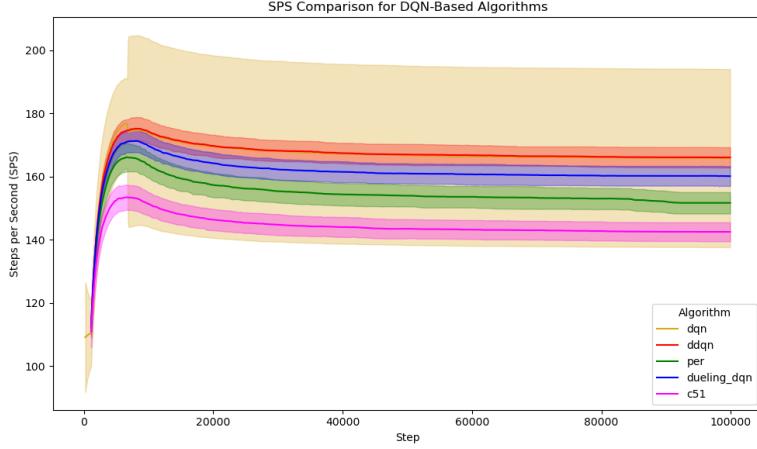
**Emissions Barplot** Finally, Figure 39 illustrates a direct barplot comparison of the average emissions (with error bars for standard deviation) for the five methods.



**Figure 39:** Emissions Barplot (DQN-based Comparison). C51 (magenta) leads with  $\sim 0.0078$  kg, while DQN (gold) is at the low end (0.00647).

**Steps Per Second (SPS) Comparison** Figure 40 displays the aggregated SPS for each of the five DQN-based algorithms. DQN (gold) and Double DQN (red) generally achieve the highest throughput, stabilizing around 160–170 steps per second. PER (green) and Dueling (blue) see moderate slowdowns, often converging in the 140–160 range. C51 (magenta), with its added distributional overhead, tends to maintain the lowest SPS,

sometimes falling below 140. These throughput differences align with the emissions data (Tables 22 and 23 on page 53), reinforcing that more computationally demanding methods can incur higher carbon footprints, even within a relatively short 100k-step training horizon.



**Figure 40:** Comparison of Steps Per Second (SPS) over 100k steps for the five DQN-based algorithms. Baseline DQN and Double DQN generally run fastest, while C51 exhibits the slowest throughput.

## 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.

**Table 24:** Key hyperparameters for the REINFORCE algorithm. Only `env_id` and `seed` vary across runs.

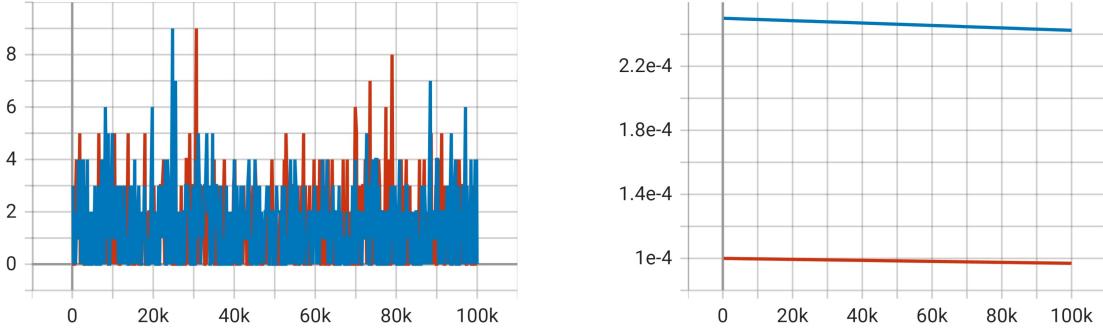
Parameter	Value
<code>exp_name</code>	reinforce_atari
<code>seed</code>	1..4
<code>torch_deterministic</code>	True
<code>cuda</code>	True
<code>track</code>	True
<code>wandb_project_name</code>	rlsb
<code>capture_video</code>	False
<code>save_model</code>	True
<code>env_id</code>	e.g. AmidarNoFrameskip-v4
<code>total_timesteps</code>	100000
<code>learning_rate</code>	0.00025
<code>num_envs</code>	1
<code>gamma</code>	0.99

#### 4.4.1 REINFORCE

REINFORCE is a classical policy gradient algorithm, discussed in Chapter 13 of [11]. In its pure form, the algorithm updates the policy parameters using the full return observed in an episode, without any baseline or actor–critic modifications, thus adhering closely to the theoretical formulation. Although it enjoys strong theoretical convergence guarantees in the limit of infinite interactions, in practice (especially over a short 100k-step training horizon) REINFORCE exhibits high variance, leading to unstable learning. Nonetheless, in our implementation we follow the standard formulation as closely as possible, with only minor modifications such as a slight learning rate annealing.

**(Hyper)Parameters** We adopt a standard configuration for REINFORCE similar to prior work on policy gradient methods. Table 24 summarizes the key hyperparameters used in our experiments. In our setup, only the environment identifier (`env_id`) and the random seed (`seed`) vary across runs. Notably, the learning rate is set to 0.00025 and the discount factor ( $\gamma$ ) to 0.99.

**Hyperparameter Tuning** We tested two primary learning rates ( $1 \times 10^{-4}$  vs.  $2.5 \times 10^{-4}$ ), each with a slight annealing schedule over the 100k steps. Ultimately,  $2.5 \times 10^{-4}$  converged more reliably in short-run experiments, as illustrated in Figure 41. However, even the better (and higher) learning rate sees limited improvements in just 100k steps, reflecting REINFORCE’s inherent high variance. In our experiments, we also tested the impact of (removing) reward clipping; however, probably because we already normalize returns as a best practice to stabilize learning, its presence or absence had no measurable effect, so we kept it for consistency with the other methods tested.



(a) Episodic Return (Raw) for Two LRs

(b) Learning Rate Schedules

**Figure 41: REINFORCE LR Tuning.** Two learning rates ( $1 \times 10^{-4}$  in red,  $2.5 \times 10^{-4}$  in blue) both with minimal decay. The higher LR eventually performed better, although volatility remained high.

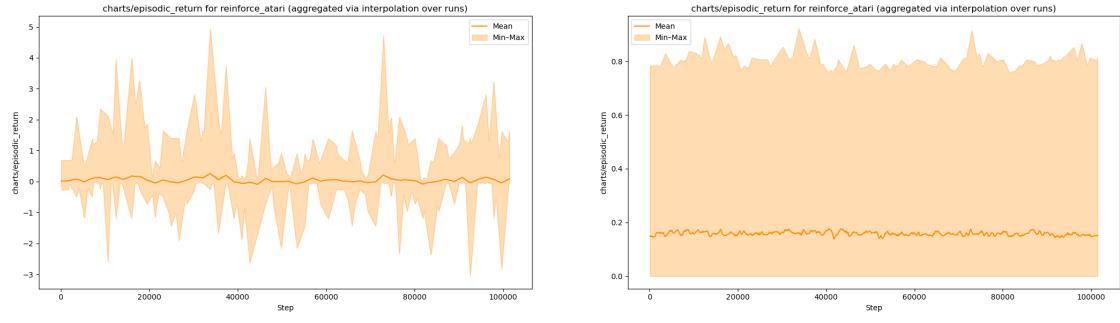
**Aggregated Returns** Figures 42a and 42b show the aggregated episodic returns (mean  $\pm$  min–max envelope) for *human-normalized* and *min–max normalized* scales:

- **Human norm** typically hovers around zero—some seeds spike up to +4 or +5, others dip to -2 or -3.
- **Min–max norm** plateaus around 0.15–0.20, but occasionally climbs near 0.8 in certain runs.

**Per-Game Breakdown** Figure 43 displays the same returns (human & min–max) but *separated by environment*:

- **Boxing** shows extreme positive and negative values in human-norm.
- **Freeway** remains nearly flat (close to zero).
- **Assault** in min–max normalization reaches approximately 0.4 in some seeds, outpacing many other tasks.

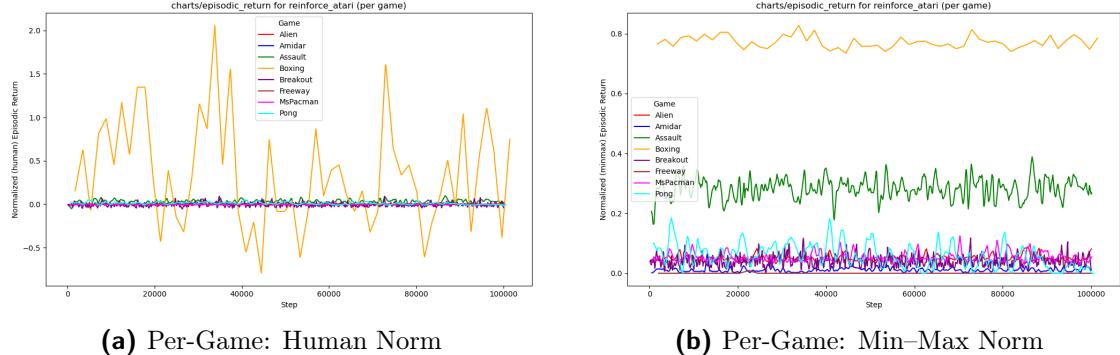
**Training Metrics** Additional training metrics are shown in Figures 44 and 45. The episode length stabilizes near 3800–4000 steps (Figure 44a), while the steps per second (SPS) plateaus between 150 and 170 (Figure 44b). The policy loss (raw) fluctuates significantly due to high-variance returns, and the learning rate drifts minimally from  $2.5 \times 10^{-4}$  to approximately  $2.4 \times 10^{-4}$  over 100k steps.



(a) Human-Normalized Returns (Aggregated)

(b) Min-Max Returns (Aggregated)

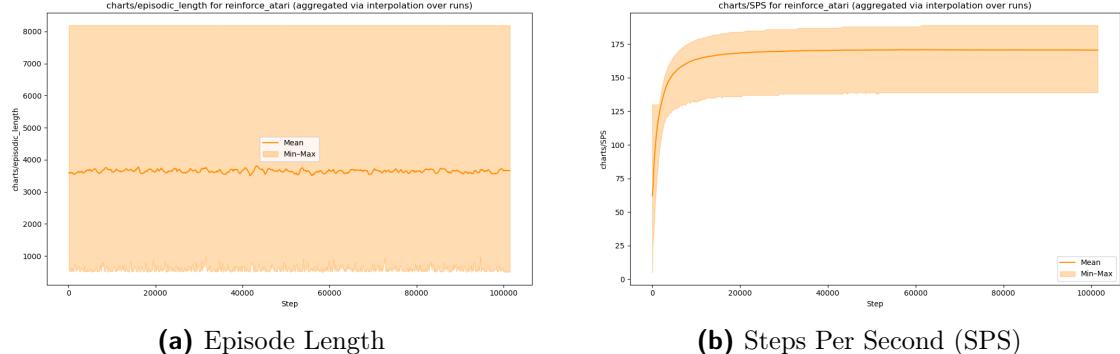
**Figure 42: Aggregated Episodic Returns.** Strong variance across seeds/games yields a wide min-max band in both normalizations. Mean in human norm often oscillates near zero, while min-max hovers around 0.15–0.20.



(a) Per-Game: Human Norm

(b) Per-Game: Min-Max Norm

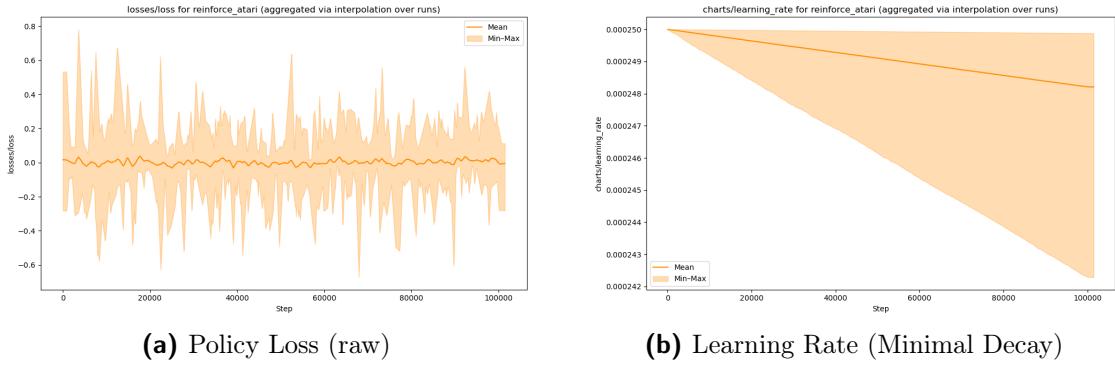
**Figure 43: Returns by Environment.** Variability is evident across different Atari games, with Boxing and Assault exhibiting the highest fluctuations.



(a) Episode Length

(b) Steps Per Second (SPS)

**Figure 44: Episode Length and Throughput.** The average length stabilizes near 3800–4000 steps; min–max can exceed 7000–8000 in some seeds. SPS plateaus around 150–170.



**Figure 45:** *Loss and Learning-Rate Curves.* The loss shows large fluctuations, while the LR decays minimally over the training period.

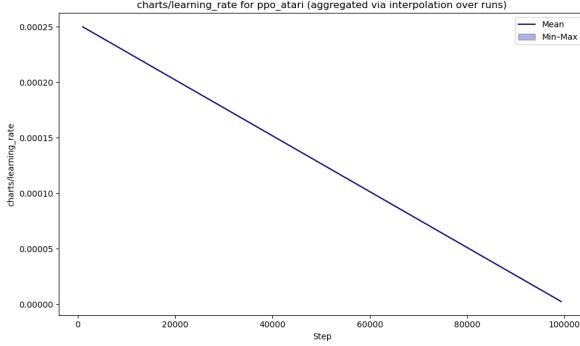
**Table 25:** REINFORCE: Final Evaluation and Emissions (Means and Ranges). High variance is evident from the negative minimum and positive maximum values.

	Human Norm	Min–Max	Emissions
Mean	0.0259	0.1544	0.00676
Std	0.3549	0.2515	0.00056
Min	-2.4048	0.0	0.00614
Max	2.8333	0.8527	0.00760

**Evaluation and Emissions** Table 25 summarizes the final evaluation performance and emissions across 32 runs (8 Atari games  $\times$  4 seeds). The mean human-normalized return is approximately 0.026, with a min–max normalized mean of about 0.154. The average energy consumption is 0.006 76 kg CO<sub>2</sub>eq, slightly above the baseline DQN’s 0.006 47 kg CO<sub>2</sub>eq.

## Discussion

- **High Variance:** REINFORCE, without a baseline, relies solely on full-episode returns, which leads to high variance in updates and unstable learning in a 100k-step regime.
- **Learning Rate Impact:** The higher learning rate of  $2.5 \times 10^{-4}$  performs better than  $1 \times 10^{-4}$ , though improvements remain limited due to the inherent variance.
- **Reward Clipping:** The absence of any effect from reward clipping confirms that return normalization is an effective stabilization measure.
- **Energy Consumption:** With emissions at 0.006 76 kg CO<sub>2</sub>eq on average, REINFORCE has a slightly higher energy cost than baseline DQN, reflecting its computational overhead despite a simpler forward pass.



**Figure 46: PPO: Learning Rate Decay.** The learning rate anneals linearly from  $2.5 \times 10^{-4}$  down to near 0 over 100 000 steps, reducing update magnitudes as training progresses.

- **Convergence Horizon:** Although REINFORCE has strong theoretical convergence guarantees, our experiments indicate that even one million steps may not be sufficient to achieve significantly better performance, underscoring the challenges of using pure policy gradient methods in practice. Practical implementations often incorporate baselines or actor–critic structures to mitigate these issues and accelerate learning in fewer steps.

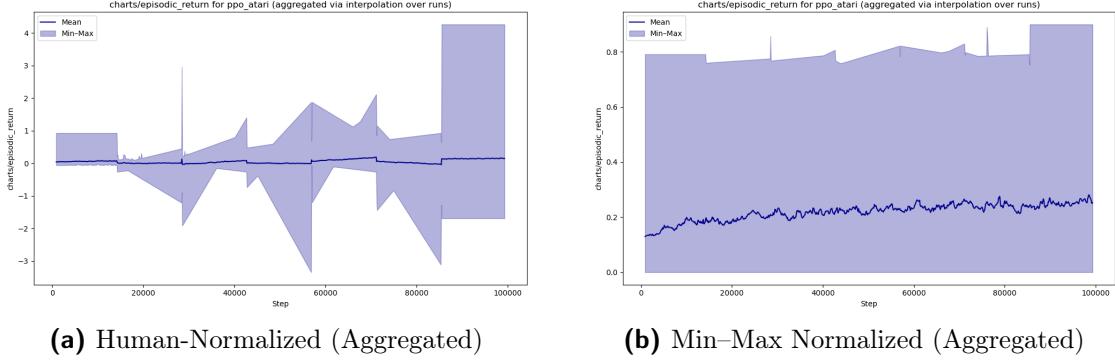
#### 4.4.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy actor–critic method that has gained popularity due to its simplicity (conceptually, the implementation presents many hardships) and robust performance. Unlike pure policy gradient methods such as REINFORCE, PPO employs a clipped surrogate objective that limits the magnitude of policy updates, thus reducing the risk of destructive parameter changes. It further leverages Generalized Advantage Estimation (GAE) [24] to compute more stable advantage estimates. These innovations allow PPO to achieve more stable and efficient learning even in high-dimensional environments, while its use of multiple parallel environments helps to improve sample efficiency.

**Hyperparameter Tuning and Settings** We follow the default *CleanRL* [21] PPO hyperparameters with a few notable modifications. Table 26 summarizes the key hyperparameters. In our setup, only `env_id` and `seed` vary across runs, while `num_envs=8` enables collecting experience from 8 parallel environments per rollout. Each iteration processes  $8 \times 128 = 1024$  steps, leading to approximately 97 updates over 100k total timesteps. Note that the parameters `batch_size`, `minibatch_size`, and `num_iterations` are computed dynamically at runtime based on the rollout configuration (i.e.,  $\text{num\_envs} \times \text{num\_steps}$ ), following the CleanRL implementation. We set `clip_coef = 0.1` (a bit lower than the common 0.2) and employ standard GAE with `gae_lambda = 0.95` for advantage estimation. Importantly, `anneal_lr=True` triggers a substantial linear decay of the learning rate from  $2.5 \times 10^{-4}$  down toward zero by the end of training, as illustrated in Figure 46.

**Table 26:** Key hyperparameters for PPO. Only `env_id` and `seed` vary across runs. Note that `num_envs=8` collects experience from 8 parallel environments each rollout.

Parameter	Value
<code>exp_name</code>	<code>ppo_atari</code>
<code>seed</code>	1..4
<code>torch_deterministic</code>	True
<code>cuda</code>	True
<code>track</code>	True
<code>wandb_project_name</code>	<code>rlsb</code>
<code>capture_video</code>	False
<code>save_model</code>	True
<code>env_id</code>	e.g. AlienNoFrameskip-v4
<code>total_timesteps</code>	100000
<code>learning_rate</code>	0.00025
<code>num_envs</code>	8
<code>num_steps</code>	128
<code>anneal_lr</code>	True
<code>gamma</code>	0.99
<code>gae_lambda</code>	0.95
<code>num_minibatches</code>	4
<code>update_epochs</code>	4
<code>norm_adv</code>	True
<code>clip_coef</code>	0.1
<code>clip_vloss</code>	True
<code>ent_coef</code>	0.01
<code>vf_coef</code>	0.5
<code>max_grad_norm</code>	0.5
<code>target_kl</code>	None
<code>batch_size</code>	1024
<code>minibatch_size</code>	256
<code>num_iterations</code>	97



**Figure 47: PPO: Aggregated Episodic Returns.** The broad shaded area shows high variance among runs/games. The mean remains around zero in human norm but reaches 0.2–0.3 in min–max.

**Aggregated Returns** Figures 47 depict the episodic returns (mean  $\pm$  min–max) for both human and min–max normalization across 32 total runs (8 games, 4 seeds each):

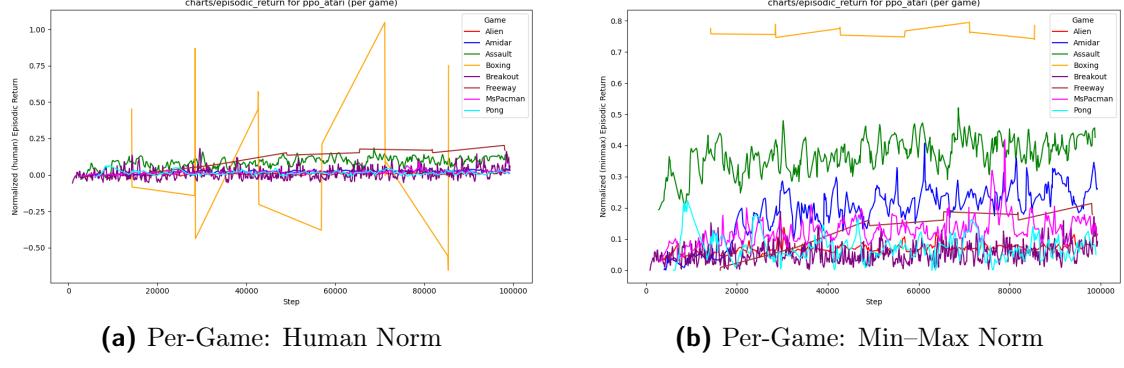
- **Human norm** (Fig. 47a): The average meanders near zero, with large negative dips (below  $-2$ ) and a few positive spikes (above  $+4$ ).
- **Min–max norm** (Fig. 47b): The mean gradually hovers around 0.2–0.3, with some seeds spiking up to 0.8 or higher.

**Per-Game Returns** Figure 48 shows the returns for each environment. They clearly illustrates that the **Boxing** environment exhibits particularly high variability in human-normalized returns, with some seeds showing extreme positive spikes and others deep negative dips. In contrast, environments such as **Assault** tend to display more consistent performance in the min–max scale. In particular:

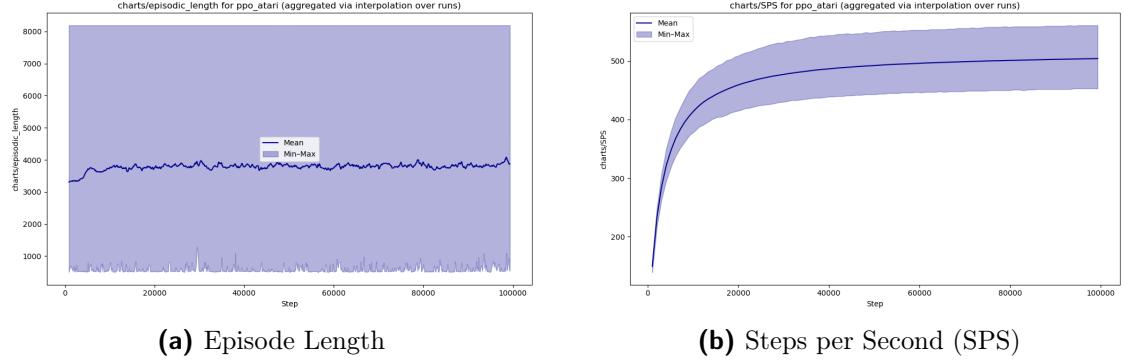
- Boxing (orange) occasionally leaps above  $+1$  or below  $-1$  in human norm, with big shifts in min–max as well.
- Assault (green) can exceed 0.4 in min–max.
- Some runs in Breakout or Freeway remain near zero.

**Additional Training Metrics** Alongside returns, we log episode length, throughput (SPS), learning rate, KL divergences, and other diagnostic losses:

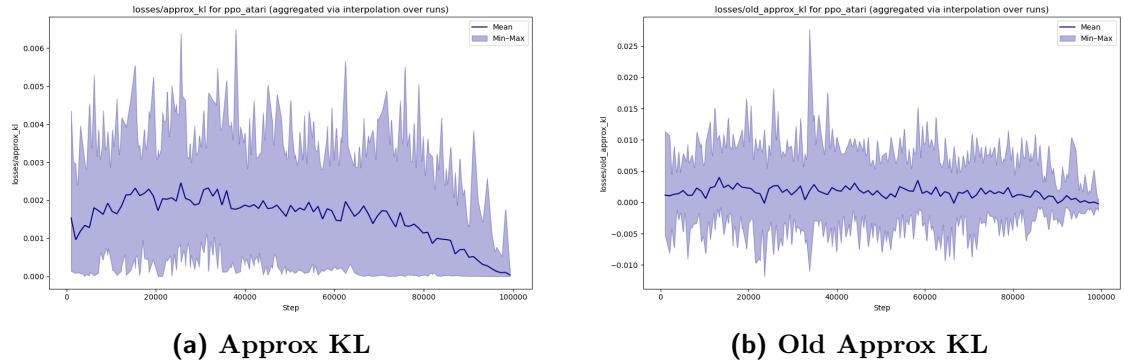
**Evaluation and Emissions** Table 27 summarizes PPO’s final returns (human/min–max) and mean CO<sub>2</sub>eq emissions. Notably, PPO emits only  $\sim 0.00288$  kg CO<sub>2</sub> on average—significantly lower than the 0.006–0.007 range observed in DQN-based methods. This efficiency likely stems from parallel sampling and faster per-environment updates.



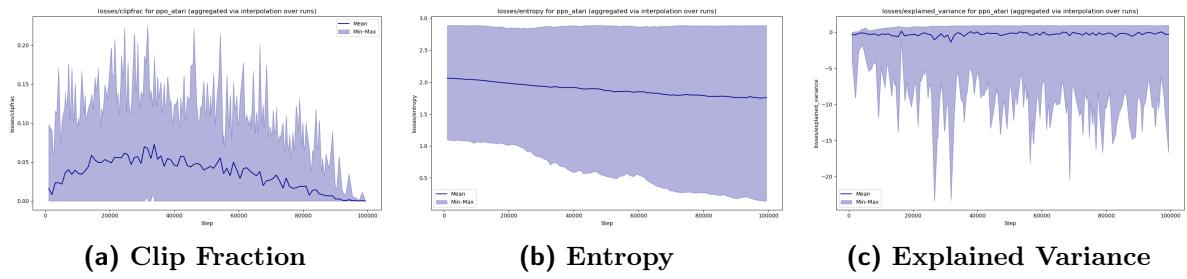
**Figure 48: PPO: Returns by Environment.** Notable extreme spikes/dips appear in *Boxing* (orange) and occasionally *Assault* or *Breakout*.



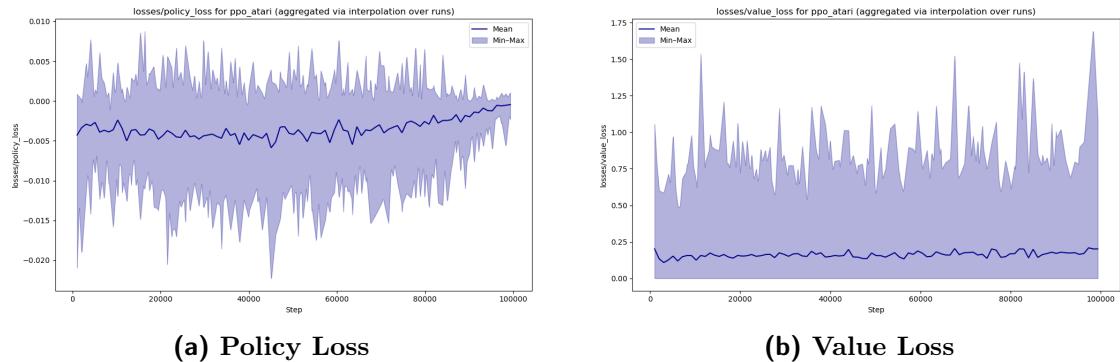
**Figure 49: PPO: Episode Length & Throughput.** The mean ep-length hovers near 3500–4000 steps, with some runs dropping below 1000 or spiking above 7000. Meanwhile, parallel envs let SPS exceed 400 or 500.



**Figure 50: PPO: KL Divergences vs. Training Steps.** Both metrics hover near 0.001–0.003 for most of training, though spikes appear. The old KL is centered around 0 but can jump above +0.02 or dip below -0.01.



**Figure 51: PPO: Additional Diagnostics.** (a) Clip fraction often spikes above 0.1 early, then drops under 0.01 by 100k steps. (b) Entropy declines from about 2.0 to below 1.5 in some runs, indicating the policy becomes more deterministic. (c) Explained variance remains near or below 0 for many runs, suggesting challenges in value function estimation in some seeds.



**Figure 52: PPO: Policy and Value Losses.** (a) The policy loss fluctuates around slightly negative values (often between  $-0.01$  and  $0.0$ ), with occasional positive spikes up to  $\sim 0.005$ . (b) The value loss remains around  $0.1$ – $0.3$  on average but can exceed  $1.0$  in some runs (max reaching over  $1.5$ ). This large min–max spread indicates variability across seeds and environments, while the mean stays comparatively stable.

**Table 27: PPO: Final Evaluation (Mean) & Emissions.** Negative dips mainly from *Boxing* or *Breakout*.

Metric	Mean	Std	Min / Max
Human-Norm Return	0.077	0.563	(-5.02 / 3.31)
Min–Max Return	0.248	0.271	(0.0 / 0.9643)
Emissions (kg CO <sub>2</sub> )	0.00288	0.00039	(0.00244 / 0.00369)

## Discussion

- **Performance Variability:** Some runs stay near zero, while others occasionally spike above +1 or +2 in human norm. The min–max scale averages around 0.25, but can exceed 0.8 in certain seeds or games.
- **Low Emissions:** PPO’s parallel rollout approach (8 envs) plus relatively fast updates yield the lowest carbon footprint among tested algorithms so far.
- **KL Divergence and LR Decay:** As the learning rate decays linearly to near zero, both the approximate KL divergence (Fig. 50) and clip fraction (Fig. 51(a)) decrease, suggesting slower policy updates toward the end of training.
- **Policy/Value Losses:** The policy loss tends to remain slightly negative (Fig. 52a), while the value loss exhibits moderate averages with occasional large spikes, implying some seeds or games exhibit abrupt divergences in state-value estimation.
- **Entropy and Value Modeling:** The entropy decline (Fig. 51b) indicates increasing determinism in the policy, and low or negative explained variance (Fig. 51c) suggests challenges in modeling returns accurately, perhaps due to limited training steps or high variability.

Overall, PPO in a 100k-step Atari benchmark exhibits high variability in returns but achieves notably lower emissions than DQN-based methods. The method’s innovations—such as the clipped surrogate objective and GAE—provide more stable learning than pure policy gradients, though performance improvements remain modest over this short training horizon.

### 4.4.3 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an off-policy actor–critic algorithm that incorporates entropy regularization to encourage exploration and robust learning. SAC leverages two Q-functions to mitigate overestimation bias and employs a target network to stabilize critic updates. A distinctive feature of SAC is the autotuning of the temperature parameter  $\alpha$ , which dynamically balances reward maximization against policy entropy by aiming for a target entropy of approximately 0.89. This combination of techniques makes SAC particularly effective in continuous control tasks, though its performance can be sensitive to hyperparameter settings when applied in discrete environments such as Atari.

**Hyperparameter Tuning and Settings** We base our SAC implementation on the `sac_atari` variant from CleanRL, adapted to a 100k-step training horizon. Table 28 summarizes the key hyperparameters. In our setup, only `env_id` and `seed` vary across runs. Notably, we use a replay buffer of 20k and set `learning_starts=1000` to ensure sufficient initial experience. Although the default SAC configuration is designed for 5 million interactions (with, for example, a `buffer_size` roughly 1/5 of that and `target_network_frequency` proportionally lower), we found that proportionally scaling

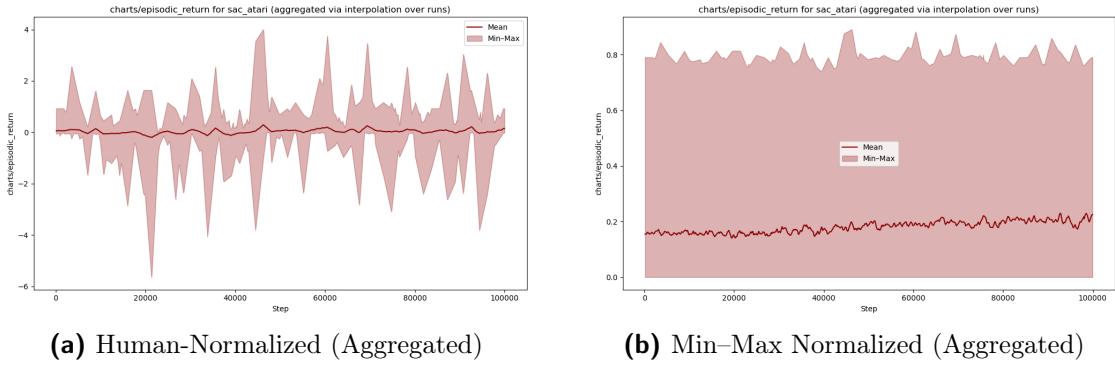
**Table 28:** Key hyperparameters for SAC. Only `env_id` and `seed` vary across runs.

Parameter	Value
<code>exp_name</code>	sac_atari
<code>seed</code>	1..4
<code>torch_deterministic</code>	True
<code>cuda</code>	True
<code>track</code>	True
<code>wandb_project_name</code>	rlsb
<code>capture_video</code>	False
<code>save_model</code>	True
<code>env_id</code>	e.g. AlienNoFrameskip-v4
<code>total_timesteps</code>	100000
<code>buffer_size</code>	20000
<code>gamma</code>	0.99
<code>tau</code>	1.0
<code>batch_size</code>	64
<code>learning_starts</code>	1000
<code>policy_lr</code>	0.0003
<code>q_lr</code>	0.0003
<code>update_frequency</code>	4
<code>target_network_frequency</code>	1000
<code>alpha</code>	0.2
<code>autotune</code>	True
<code>target_entropy_scale</code>	0.89

these values leads to instability or completely unrealistic values. Therefore, we chose a `target_network_frequency` of 1000 (similar to our DQN-based methods) to maintain stable target updates. Autotuning is enabled for  $\alpha$  so that the entropy coefficient adjusts dynamically based on a target entropy of  $\sim 0.89$ . Additionally, both `policy_lr` and `q_lr` are set to 0.0003. These modifications on which we settled after a coarse grid search represent a compromise between the original CleanRL defaults and the practical demands of training under a limited 100k-step regime.

**Aggregated Returns** Figures 53 illustrate the aggregated episodic returns (mean  $\pm$  min–max) in both human- and min–max-normalized scales. Over a 100k-step horizon, considerable variance is observed, with some runs dropping below -4 on the human-normalized scale while min–max values mostly remain under 0.8.

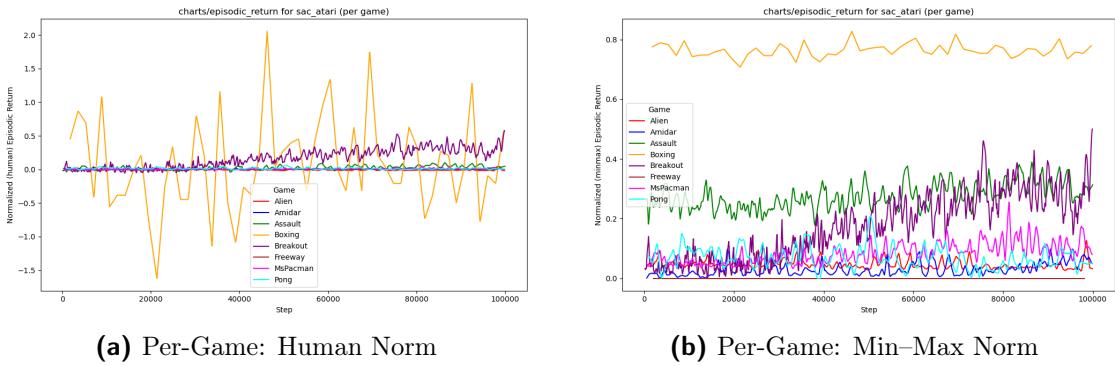
**Per-Game Returns** Figure 54 breaks down performance by environment. Notably, *Boxing* significantly skews the human-norm average downward, while *Breakout* and *Assault* achieve moderate positive scores.



(a) Human-Normalized (Aggregated)

(b) Min-Max Normalized (Aggregated)

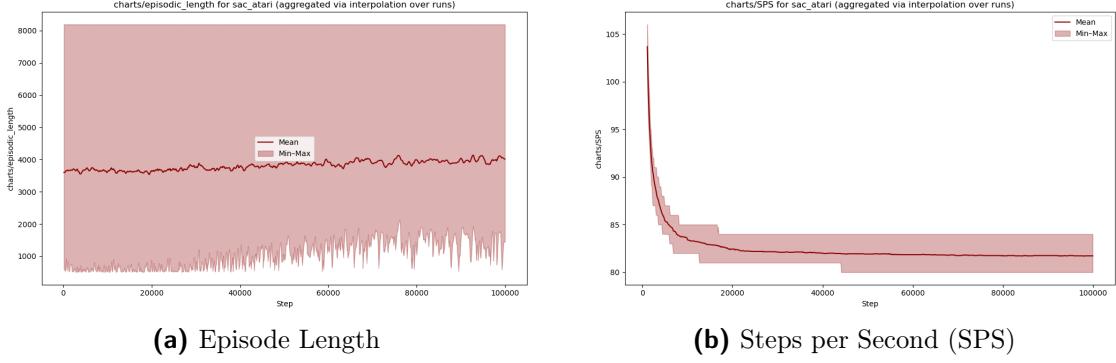
**Figure 53: SAC: Aggregated Episodic Returns.** Notice the swings below -4 in the human norm, while min–max remains mostly under 0.8.



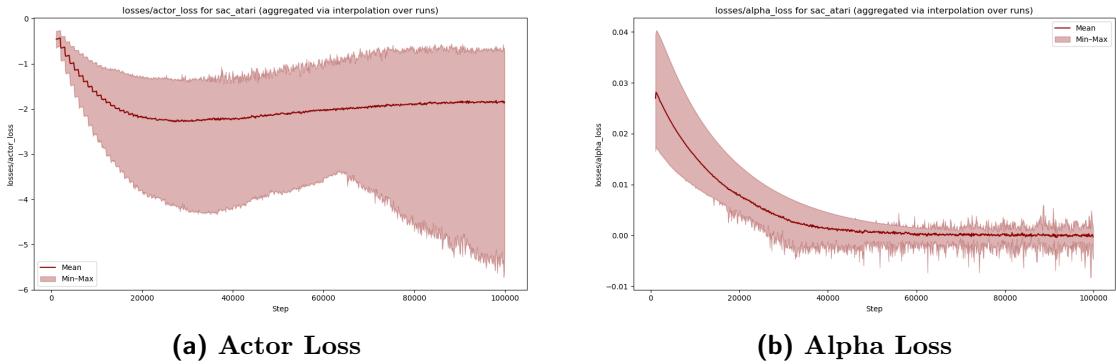
(a) Per-Game: Human Norm

(b) Per-Game: Min-Max Norm

**Figure 54: SAC: Returns by Environment.** *Boxing* can dip below -20 in some runs, while *Pong* remains near -0.01 (human norm).



**Figure 55: SAC: Episode Length & Throughput.** Min–max ranges from under 1000 to over 7000 steps, while SPS stabilizes around 80 after initial decay.

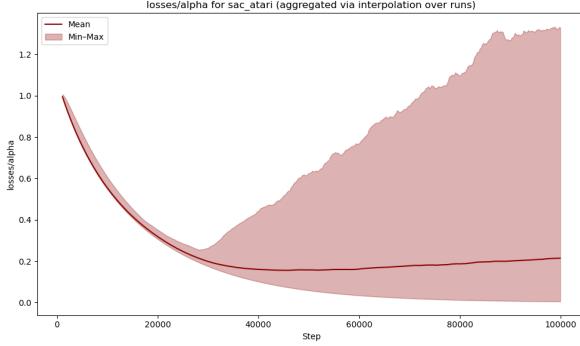


**Figure 56: SAC: Actor and Alpha Losses.** The actor loss bottoms out near -3 by  $\sim 10$ k steps, while the alpha loss decays from 0.03 to nearly 0.

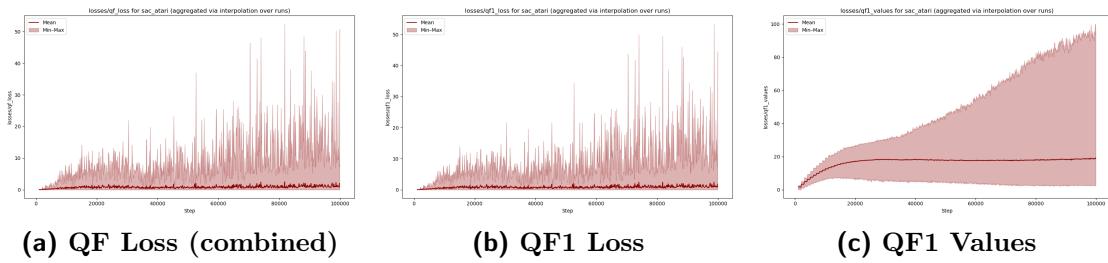
**Episode Length and SPS** Figure 55 shows the episode length and steps-per-second (SPS). The average episode length hovers near 3800–4000 steps, while SPS drops from over 100 at the beginning to approximately 80 by 20k steps, stabilizing thereafter.

**Actor Loss and Temperature ( $\alpha$ )** Figures 56 and 57 detail the evolution of the actor loss and the entropy temperature:

- **Actor Loss:** Initially near -1.0, it quickly drops to about -3 by 10k steps, then recovers to around -2.
- **Alpha Loss:** Falls below 0.01 around 30k steps, indicating fewer gradient corrections to  $\alpha$ .
- **$\alpha$  Parameter:** Exhibits a U-shaped curve, dipping to approximately 0.1 before rising above 0.3; some runs exceed 1.0 near the end, suggesting increased exploration.



**Figure 57: SAC: Learned  $\alpha$  Over Time.** After dipping to around 0.1 near 30k steps, some runs see  $\alpha$  climb above 1.0, with the mean reaching approximately 0.3 at 100k steps.



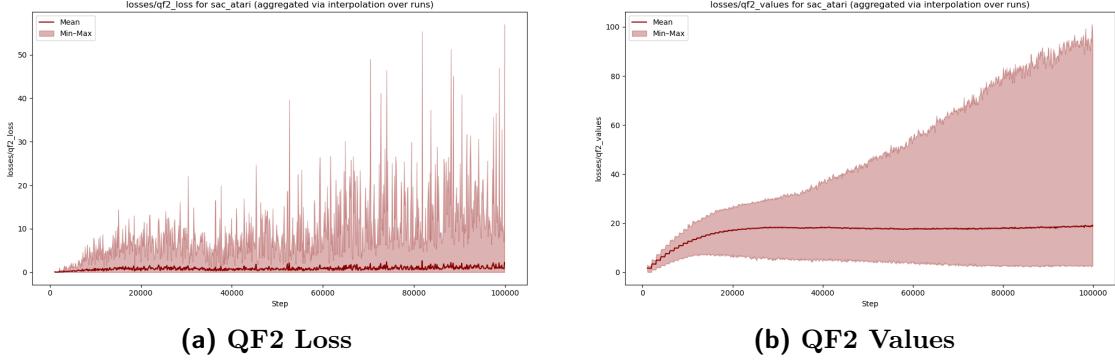
**Figure 58: SAC: QF1 Metrics.** Mean QF1 losses remain below 3, although min–max spikes can exceed 50 after 40k steps. QF1 value estimates rise from near 0 to about 20 in the mean, with some outliers above 80.

**Q-Function Losses and Values** SAC utilizes two Q-functions (QF1 and QF2) to mitigate overestimation and stabilize learning. Figures 58 and 59 showcase their losses and value estimates:

Both QF1 and QF2 display moderate mean losses (generally below 3), but their min–max band ranges reveal occasional instability, particularly after  $\sim 40$ k steps. Q-value estimates similarly show high variability, with some seeds surpassing 80–100 near the end, which can indicate overestimation or genuinely high return states.

**Evaluation and Emissions** Table 29 reports final performance and average CO<sub>2</sub> emissions. Due to extreme negative results in *Boxing*, the mean human-normalized return is  $-1.10$ , whereas the min–max normalized mean is around 0.227. *Breakout* and *Assault* partially offset this. Emissions average approximately 0.01545 kg CO<sub>2</sub>, which is higher than PPO’s but in line with other off-policy methods that require more frequent updates.

**Evaluation Procedure** While SAC is naturally a stochastic policy method (and many works in continuous control evaluate it by sampling actions), in our discrete Atari setting we opted for a deterministic evaluation: selecting the action with the highest probability at each decision step. This choice not only aligns with the evaluation procedures of other



**Figure 59: SAC: QF2 Metrics.** QF2 losses also exhibit large spikes (up to 50+ in some runs), while QF2 values increase from below 10 to around 20 on average, reaching up to 100 in certain seeds.

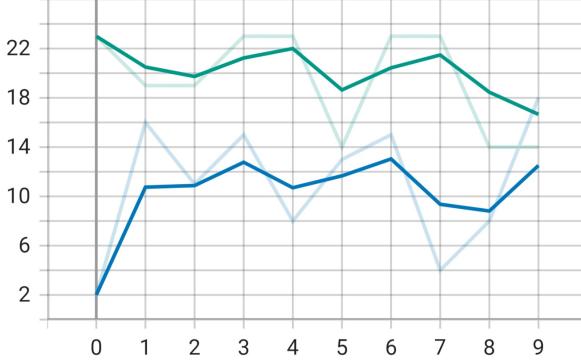
**Table 29:** SAC: Final Evaluation (Mean) & Emissions.

Metric	Mean	Std	Min	Max
Human-Norm Return	-1.10	4.43	-23.36	0.93
Min-Max Return	0.227	0.254	0.00	0.83
Emissions (kg CO <sub>2</sub> )	0.01545	0.00033	0.01495	0.01610

discrete-action methods such as DQN but also tends to yield more stable performance metrics (and, obviously, higher returns, see figure 60). Although many continuous-action SAC studies employ stochastic evaluation, our approach is consistent with several adaptations of SAC to discrete domains, where deterministic evaluation is preferred.

## Discussion

- **High Q-Function Variance:** Both QF1 and QF2 exhibit substantial spikes in losses and inflated value estimates, suggesting overestimation or instability in certain seeds—a common challenge in off-policy algorithms with limited training interactions.
- **$\alpha$  Behavior:** The learned entropy coefficient ( $\alpha$ ) displays a U-shaped trajectory, initially decreasing to promote exploitation, then increasing in some runs (even exceeding 1.0), which indicates renewed exploration later in training.
- **Performance Skew:** Extremely poor performance in *Boxing* dominates the human-norm average, although environments like *Breakout* and *Assault* achieve moderate positive scores, illustrating the algorithm’s uneven game-to-game performance in only 100k steps.
- **Emissions:** The relatively high number of gradient updates (update frequency = 4), coupled with maintaining two critics/Q-functions, results in higher carbon



**Figure 60:** The two possible SAC evaluation (deterministic in green and stochastic in blue) compared on Breakout. The plots show the 10 raw evaluation returns.

**Table 30:** Overall human-normalized returns (aggregated) for policy gradient algorithms.

Algorithm	Mean	Std	Min	Max	IQM	Median
PPO	0.077	0.563	-5.02	3.31	0.0173	0.0163
REINFORCE	0.026	0.355	-2.40	2.83	-0.0039	-0.0029
SAC	-1.100	4.428	-23.36	0.93	0.0085	0.0045

emissions ( $\sim 0.01545$  kg CO<sub>2</sub>) compared to simpler on-policy methods, although this is expected given the off-policy nature of SAC.

Overall, SAC shows promise in some Atari tasks under 100k interactions, yet it suffers from high variance in Q-function estimates and a complex  $\alpha$  schedule. Longer training or adjustments to the critic updates (maybe more conservative ones) may be required to achieve more stable performance across seeds.

## 4.5 Overall Comparison of Policy Gradient Algorithms

In this section, we compare the three policy gradient-based algorithms tested in our benchmark: **PPO** (on-policy with a clipped objective), **REINFORCE** (a basic Monte Carlo policy gradient), and **SAC** (an off-policy method with automatic entropy tuning). Each algorithm was trained for 100 000 steps on the same 8 Atari games, with 4 random seeds per game, producing 32 runs per algorithm. We examine both their *final performance* (human-normalized and min–max normalized returns) and *carbon emissions* over the course of training.

**Final Evaluation Performance: Human-Normalized.** Table 30 summarizes the aggregated human-normalized returns over all 8 environments for each algorithm (mean, std, etc.).

**Table 31:** Overall min–max normalized returns (aggregated) for policy gradient algorithms.

Algorithm	Mean	Std	Min	Max	IQM	Median
PPO	0.248	0.271	0.00	0.964	0.1523	0.1204
REINFORCE	0.154	0.252	0.00	0.853	0.0291	0.0393
SAC	0.227	0.254	0.00	0.826	0.1477	0.1105

**Table 32:** Average carbon emissions (kg CO<sub>2</sub>) for policy gradient algorithms over 100k steps.

Algorithm	Mean	Std	Min	Max
PPO	0.00288	0.00039	0.00244	0.00369
REINFORCE	0.00676	0.00056	0.00614	0.00760
SAC	0.01545	0.00033	0.01495	0.01610

By this metric, **PPO** shows the highest mean value (0.077), albeit with large variance (0.563). **REINFORCE** follows at 0.026, while **SAC** has a negative average (−1.10), strongly influenced by its very poor performance on *Boxing* (see Section ?? for details). Notably, the interquartile mean (IQM) for all three algorithms is near zero or slightly negative, reflecting that the short 100k-step horizon yields limited gains in some games. SAC’s extreme negative outliers in *Boxing* pull its mean well below zero, even though it attains moderate success in *Assault* and *Breakout*.

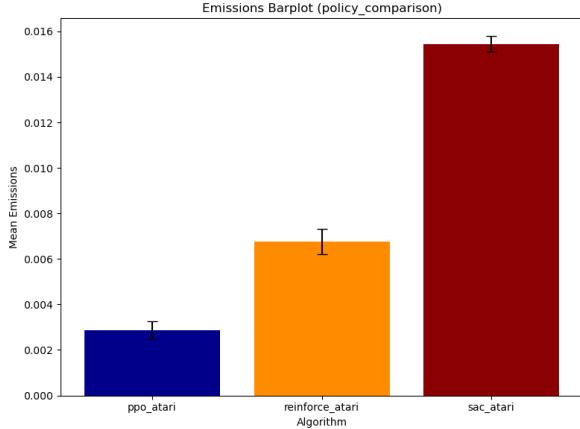
**Final Evaluation Performance: Min–Max Normalized.** A similar pattern emerges when we switch to min–max normalization, as shown in Table 31.

Here, **PPO** leads with a mean of 0.248, slightly outperforming **SAC** (0.227). **REINFORCE** remains lower on average (0.154). Interestingly, SAC’s min–max maximum (0.826) is in line with PPO’s (0.964), suggesting some of its runs achieve decent returns in certain games, offset by very low or zero returns in others.

**Emissions and Energy Consumption.** In addition to performance, our benchmark tracks **carbon emissions** (kg CO<sub>2</sub>) during training (Fig. 61). Table 32 shows the mean emissions for each algorithm aggregated over all 32 runs:

**PPO** exhibits the lowest emissions, averaging ∼0.0029 kg CO<sub>2</sub> per run, whereas **SAC** is by far the highest (∼0.0154 kg). **REINFORCE** lands in the middle at ∼0.0068 kg. These differences likely arise from:

- *REINFORCE*: Simpler architecture, but it replays from scratch each episode (Monte Carlo), incurring moderate overhead.
- *PPO*: On-policy sampling plus parallel environments yield a relatively fast throughput, reducing total compute time.
- *SAC*: Off-policy approach with frequent gradient updates, two Q-networks, and autotuning overhead results in higher GPU usage.



**Figure 61: Policy Algorithms: Mean Emissions (kg CO<sub>2</sub>).** Error bars show the standard deviation across 32 runs.

**Table 33:** Total wall-clock time for 32 runs (8 games × 4 seeds) per algorithm.

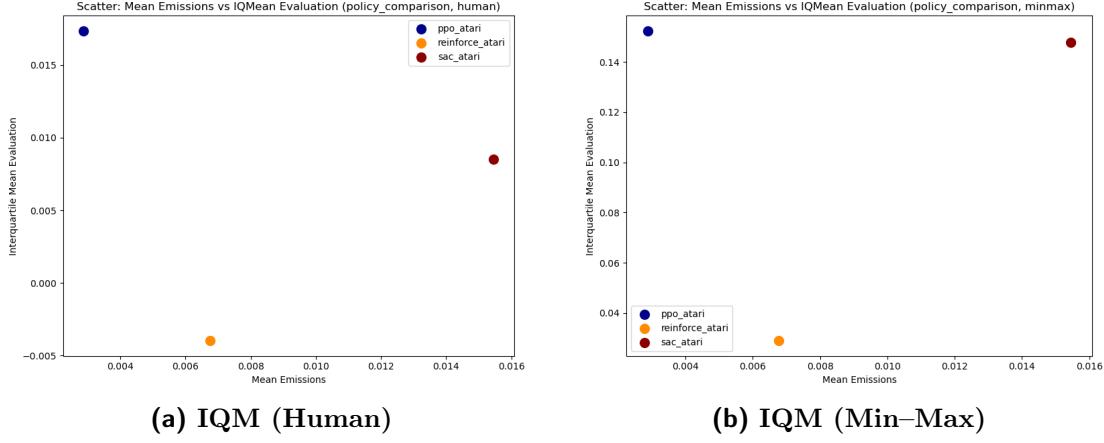
Algorithm	Total Training Time
PPO	2 h 25 m 08 s
REINFORCE	5 h 53 m 23 s
SAC	11 h 28 m 23 s

**Performance vs. Emissions** Beyond the aggregated returns and raw carbon footprints, it is also informative to visualize the **“trade-off”** between *mean emissions* (kg CO<sub>2</sub>) and *performance* (mean or IQM). Figures 62 and 63 combine these metrics under both human-normalized and min–max normalization.

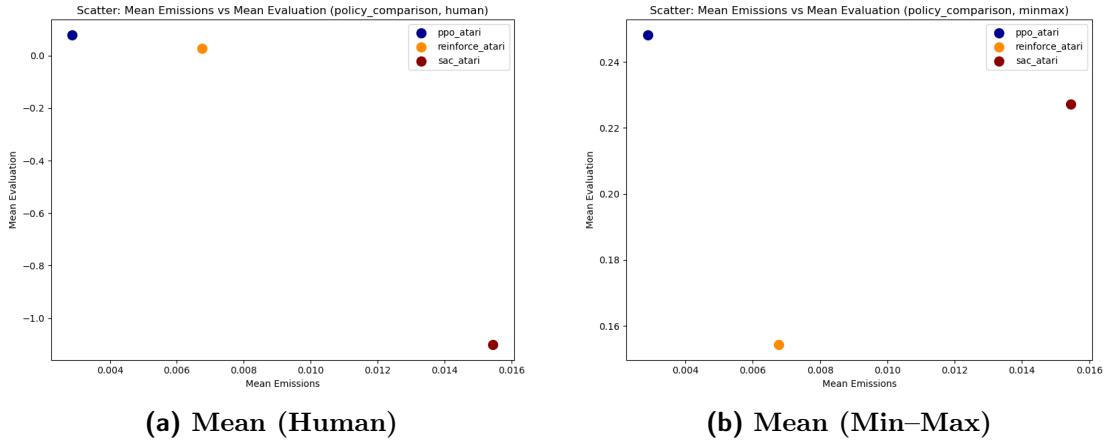
*Observation.* - In all four scatter plots, **PPO** (blue) occupies the *lowest emissions* regime ( $\approx 0.003$  kg CO<sub>2</sub>) yet achieves higher or comparable returns than the others. - **SAC** (red) clearly emits more ( $\approx 0.015$  kg), placing it on the far right of each scatter, while its performance can be strong (especially in mean min–max) or moderate (IQM), depending on the metric. - **REINFORCE** (orange) sits in between for emissions ( $\approx 0.007$  kg) but remains near zero or negative in human-normalized returns. Overall, these visualizations underscore a **“trade-off”**: although SAC can sometimes compete in raw performance, its significantly higher carbon footprint may not be ideal for short 100k-step benchmarks.

**Training Throughput and Runtime** A key factor driving these energy differences is **training speed**, often measured in *Samples per Second* (SPS). Figure 64 compares the SPS curves for PPO, SAC, and REINFORCE across the 100k interactions. Meanwhile, Table 33 lists the total wall-clock time for all 32 runs (8 games × 4 seeds).

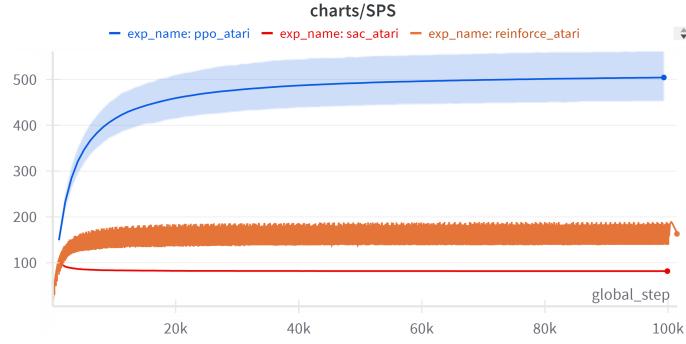
*Commentary.* - **PPO** easily attains the highest SPS ( $\sim 500$ ), completing all runs in just over 2.4 hours. This efficiency helps keep its emissions the lowest. - **SAC**’s off-policy updates and dual Q-networks produce a low SPS ( $\sim 100$ ), causing an 11.5-hour total



**Figure 62: Mean Emissions vs. Interquartile Mean (IQM) Return.** Each point corresponds to one algorithm’s aggregated final performance (IQM) against its mean carbon emissions. (a) uses human-normalized returns, while (b) is min–max normalized.



**Figure 63: Mean Emissions vs. Mean Return.** Similar to Fig. 62, but plotting the average (mean) final performance in human (a) vs. min–max (b) normalization.



**Figure 64: Combined SPS Curves for PPO, SAC, and REINFORCE.** PPO rapidly climbs above 400–500 SPS, while SAC settles around 100–120, and REINFORCE around 150–200. Shaded areas (if visible) indicate min–max across seeds.

runtime and the highest carbon footprint. - **REINFORCE**, despite simpler logic, often hits  $\sim 150$  SPS, finishing around 5.9 hours. Its overhead partly stems from lower data efficiency (Monte Carlo returns) and non-parallel sampling.

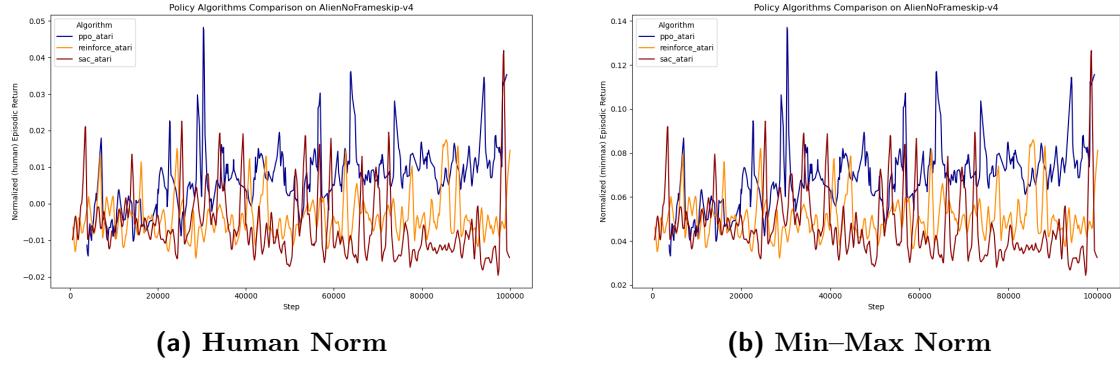
**Synthesis** Putting it all together:

- **PPO** emerges as the most *energy-efficient* approach, yielding good (and sometimes top) performance while taking the shortest total runtime ( $\approx 2.4\text{h}$ ). It also leads in many environments, particularly *Amidar*, *Assault*, *Freeway*, and *MsPacman*.
- **SAC** can match or exceed PPO in certain tasks (*Breakout*, *Alien*), but suffers from a long training time and high emissions due to frequent gradient updates and dual Q-networks. Its negative outliers in *Boxing* also drag down the overall human-norm mean.
- **REINFORCE** sits in the middle for carbon usage ( $\approx 0.007 \text{ kg CO}_2$ ), but lags behind significantly in final returns, highlighting the difficulty of pure Monte Carlo methods within 100k steps.

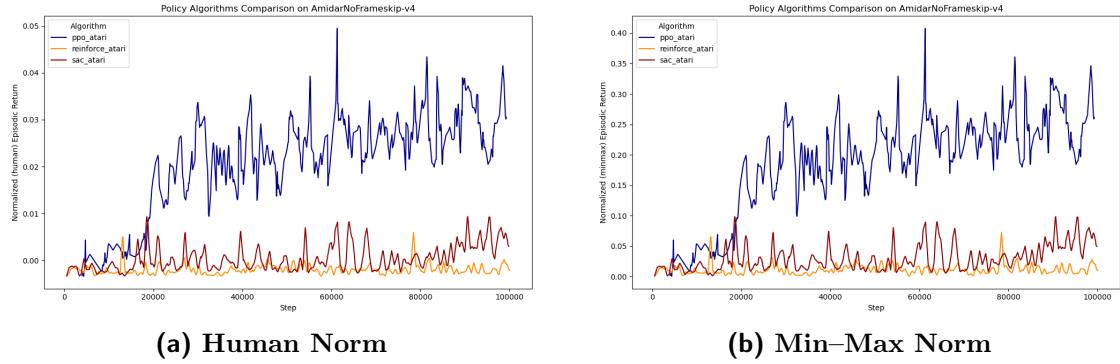
Hence, for *short-horizon* Atari training, **PPO** stands out as the best balance of performance and sustainability, while **SAC** demands substantially more compute resources for often modest gains—except in specialized games like *Breakout*, where it excels.

**Per-Game Observations** To illustrate performance trends for **PPO**, **REINFORCE**, and **SAC**, each of the eight Atari games is now presented with a paired set of plots. One subfigure shows the human-normalized returns over 100 000 steps, while the other displays the corresponding returns under **min–max normalization**, which rescales each environment’s returns based on its observed minimum and maximum. This dual presentation clarifies differences that might be obscured by large raw score ranges.

**Alien.**



**Figure 65: Alien.** (a) Under human normalization, PPO (blue) and SAC (red) periodically reach 0.04–0.05, while REINFORCE (orange) lingers near 0.0. (b) Min–max scaling places PPO’s peaks around 0.12–0.14, with SAC following closely by 100k steps, and REINFORCE near 0.04–0.08.



**Figure 66: Amidar.** (a) PPO (blue) rises above 0.03, while SAC (red) and REINFORCE (orange) stay under 0.01. (b) Min–max scaling reveals PPO crossing 0.30, whereas SAC/REINFORCE remain below 0.1.

*Commentary.* Figure 65 shows that the numeric range under min–max is larger (up to 0.14) than in human norm (up to 0.05), yet the relative ordering (PPO  $\approx$  SAC  $>$  REINFORCE) remains similar.

### Amidar.

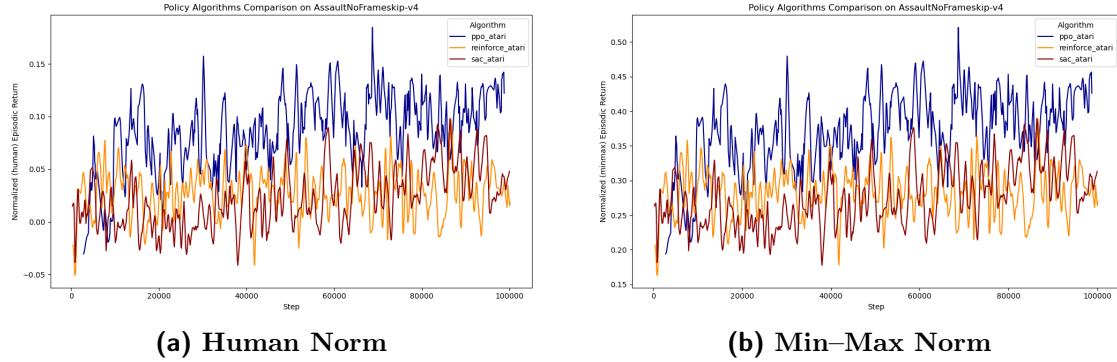
*Commentary.* In Figure 66, the difference is more dramatic in min–max space, where PPO nears 0.35 vs.  $\approx$  0.05 in human norm.

### Assault.

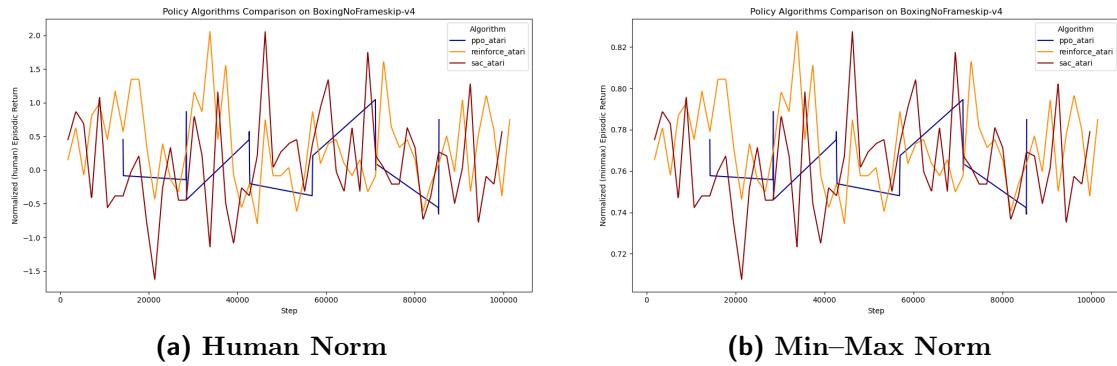
*Commentary.* PPO leads in both scales, with a clearer margin under min–max (0.50 vs. 0.35).

### Boxing.

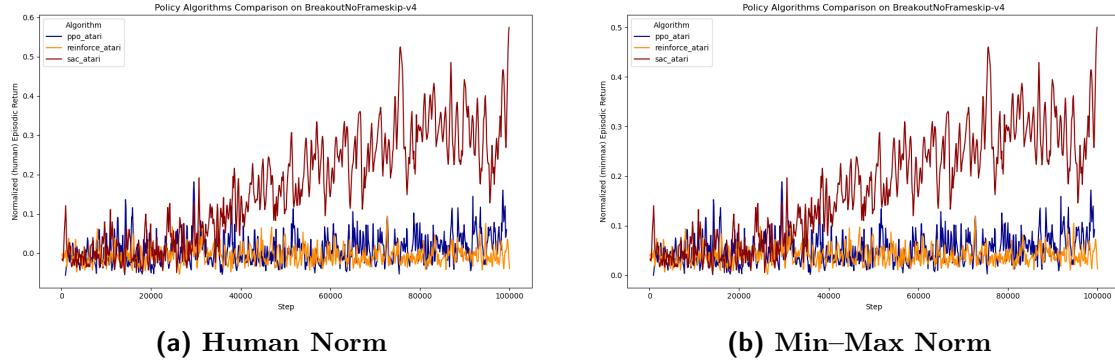
*Commentary.* Figure 68 highlights how an environment’s large or small absolute score range can drastically alter the min–max scale.



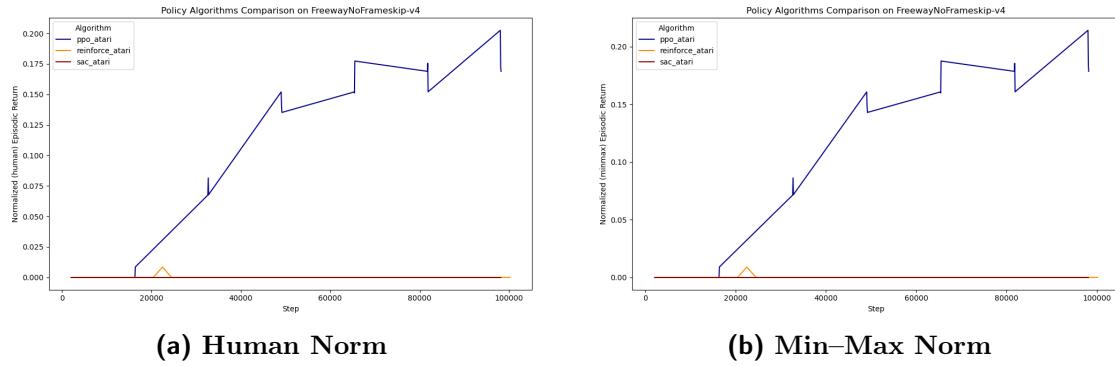
**Figure 67: Assault.** (a) PPO (blue) approaches 0.15–0.18, while SAC (red) remains near 0.05–0.10. (b) Min–max scaling finds PPO around 0.30–0.50, SAC 0.20–0.35, and REINFORCE (orange) near 0.20–0.30.



**Figure 68: Boxing.** (a) All three exhibit wild swings in human norm (e.g., REINFORCE from -1.5 to +2). (b) Min–max compresses those swings to  $\sim 0.72$ –0.82.



**Figure 69: Breakout.** (a) SAC (red) surpasses 0.4–0.5, well above PPO (blue) or REINFORCE (orange). (b) Min–max confirms SAC reaching above 0.5, while PPO stays near 0.2.



**Figure 70: Freeway.** (a) PPO (blue) climbs to  $\sim 0.18\text{--}0.20$ , while SAC (red) and REINFORCE (orange) remain near 0.0. (b) Min–max also shows PPO at  $\sim 0.20$ , with the others near 0.0.

### Breakout.

*Commentary.* In Breakout, **SAC** is the clear winner under both normalizations, though the gap appears even larger in min–max form.

### Freeway.

*Commentary.* Figure 70 illustrates how SAC and REINFORCE barely register on either scale, as PPO’s partial success reveals a bigger raw score gap.

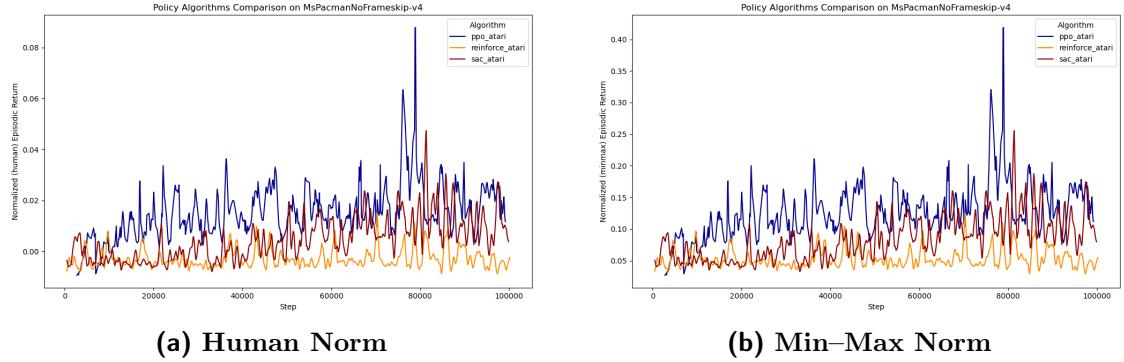
### MsPacman.

*Commentary.* Figure 71 underscores **PPO**’s lead in MsPacman, while SAC remains behind, and REINFORCE barely improves.

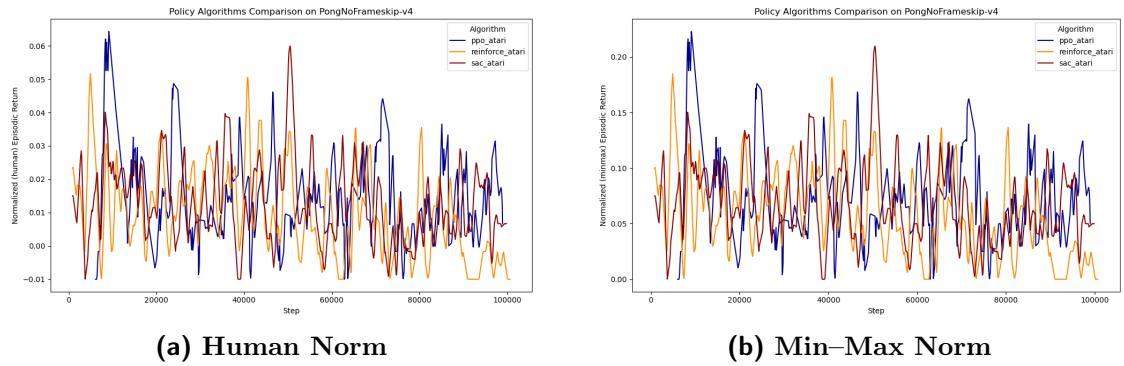
### Pong.

*Commentary.* In *Pong*, Figure 72 reveals no substantial improvement by PPO, SAC, or REINFORCE, regardless of the normalization scheme.

**Overall Combined Takeaways.** Our analysis across both human-normalized and min–max normalized returns reveals consistent trends among the three algorithms. Across all



**Figure 71: MsPacman.** (a) PPO (blue) peaks near 0.08 by 80k steps, SAC (red) stays in 0.03–0.05, REINFORCE (orange) below 0.02. (b) Min–max scales these results, yielding PPO at  $\sim 0.35$ –0.40 vs. SAC  $\sim 0.15$ –0.20, and REINFORCE  $< 0.10$ .



**Figure 72: Pong.** (a) All three methods fluctuate around 0.0–0.06 in human norm. (b) Min–max normalizes that to about 0.0–0.2.

**Table 34:** Overall final returns (human-normalized) for all algorithms.

Algorithm	Mean	Std	Min	Max	IQM	Median
C51	-1.0811	3.2862	-12.88	0.7770	0.0068	0.0000
DDQN	0.0226	1.0083	-8.5952	2.5952	0.0894	0.0527
DQN	0.1353	0.7541	-5.0238	4.7381	0.1137	0.0338
DUELING_DQN	0.1860	0.5258	-1.9286	3.5476	0.1020	0.0402
PER	0.0607	1.0170	-10.2619	6.8809	0.0813	0.0539
PPO	0.0775	0.5632	-5.0238	3.3095	0.0173	0.0163
REINFORCE	0.0259	0.3549	-2.4048	2.8333	-0.0039	-0.0029
SAC	-1.1000	4.4278	-23.3571	0.9286	0.0085	0.0045

eight Atari games, **PPO** generally achieves the highest or near-highest returns—especially in *Amidar*, *Assault*, *Freeway*, and *MsPacman*—demonstrating robust performance over the 100k-step benchmark. In contrast, while **SAC** excels notably in *Breakout* (and sometimes in *Alien*), it exhibits greater variability, struggling in environments such as *Freeway* and *Pong* and showing high volatility in *Boxing*. **REINFORCE** consistently underperforms, with only occasional spikes (as seen in *Boxing*), which highlights the challenges of employing pure Monte Carlo policy gradients within this limited interaction horizon.

Moreover, while the two normalization schemes yield different numerical scales—reflecting, for example, that large raw score ranges in *Boxing* are compressed under min–max—the relative ranking among the algorithms remains largely similar. Overall, these findings underscore **PPO** as the most consistently successful policy gradient method in our study, with **SAC** showing pockets of promise amid higher variance, and **REINFORCE** trailing significantly in most tasks.

## 4.6 Overall Algorithm Comparison

We now bring together all eight algorithms from our benchmark—five value-based methods (*DQN*, *DDQN*, *DuelingDQN*, *PER*, *C51*) plus three policy-gradient methods (*REINFORCE*, *PPO*, *SAC*)—to provide a unified comparison of their final performance, carbon emissions, training runtime, and throughput (SPS).

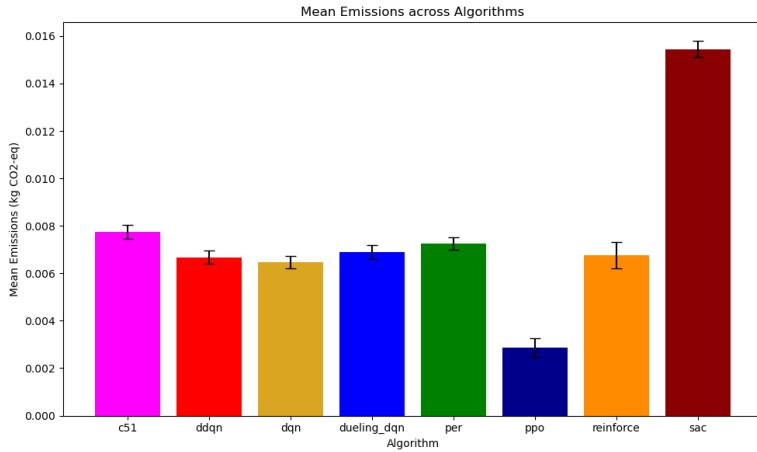
### 4.6.1 Final Evaluation Performance

Tables 34 and 35 summarize the aggregated performance (mean, std, min, max, median, IQM) for each algorithm under both **human** and **min–max** normalization, respectively. These results extend the single-family comparisons made earlier to the full set of eight algorithms.

*Observation.* From these tables, we see that among the *value-based* algorithms, DuelingDQN, DQN, and DDQN often exhibit the highest mean or IQM returns, while *PER* and *C51* sometimes trail behind or show higher variance (especially in human-norm with

**Table 35:** Overall final returns (min–max normalized) for all algorithms.

Algorithm	Mean	Std	Min	Max	IQM	Median
C51	0.2503	0.2568	0.0	0.8270	0.1400	0.2005
DDQN	0.3737	0.2854	0.0	1.0000	0.3272	0.2887
DQN	0.3802	0.3099	0.0	0.9881	0.3426	0.2899
DUELING_DQN	0.3849	0.3056	0.0	0.9523	0.3454	0.2632
PER	0.3533	0.2695	0.0	0.9845	0.3087	0.2583
PPO	0.2481	0.2712	0.0	0.9643	0.1523	0.1204
REINFORCE	0.1544	0.2515	0.0	0.8527	0.0291	0.0393
SAC	0.2272	0.2536	0.0	0.8258	0.1477	0.1105



**Figure 73: Mean Emissions for All 8 Algorithms**, with standard deviation bars. SAC stands out at nearly 0.015 kg CO<sub>2</sub> on average, while PPO is notably lower than any of the DQN-based methods.

large negative outliers). For the *policy* group, SAC yields strong results in some tasks but is heavily penalized in others (e.g., *Boxing*), resulting in negative or near-zero means in human norm. PPO remains moderate, whereas REINFORCE lags in final average.

#### 4.6.2 Emissions and Runtime

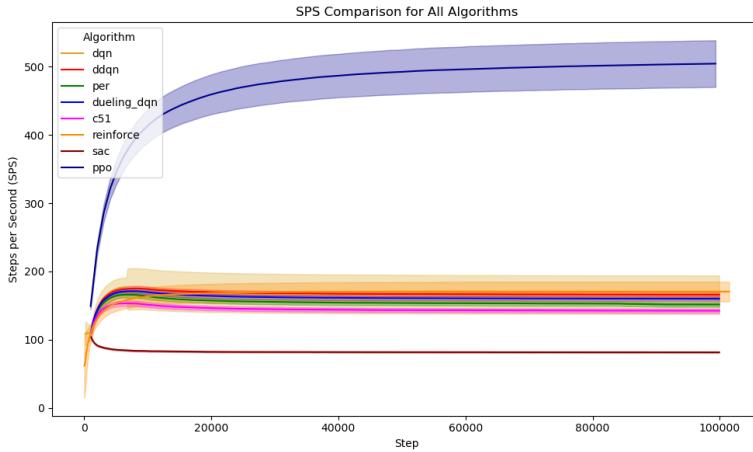
Figure 73 compares the average emissions (kg CO<sub>2</sub>) for all 8 algorithms, along with standard deviation error bars. Table 36 then lists each method’s total wall-clock time to complete 8 games × 4 seeds = 32 runs.

*Commentary.*

- **SAC** demands the longest runtime (over 11.4 hours) and, as seen in Figure 73, produces the highest average emissions.
- **PPO** completes all runs in just 2.4 hours, with the lowest ~ 0.003 kg CO<sub>2</sub>.

**Table 36:** Total runtime (hh:mm:ss) over 32 runs per algorithm.

Algorithm	Total Time
DQN	5h 46m 54s
DDQN	5h 55m 17s
PER	6h 26m 58s
DUELING_DQN	6h 08m 02s
C51	6h 51m 23s
REINFORCE	5h 53m 23s
PPO	2h 25m 08s
SAC	11h 28m 23s



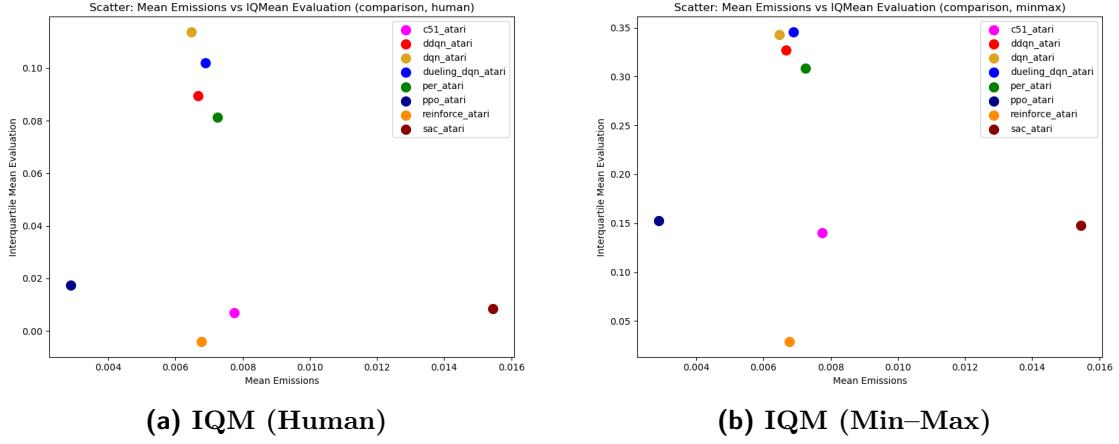
**Figure 74: SPS Comparison for All Algorithms.** PPO (blue line/shade) surpasses 500 SPS (mainly due to the parallel environments), while DQN variants group around 150–200, REINFORCE near 150, and SAC remains under 100. Shaded regions represent min–max or std across seeds.

- **Most DQN variants** cluster around 5–7 hours total, with emissions  $\sim 0.006\text{--}0.008 \text{ kg CO}_2$ , well above PPO but significantly below SAC.

#### 4.6.3 Samples per Second (SPS) Comparison

Another measure of algorithmic efficiency is how many environment interactions each method can process per second. Figure 74 plots the aggregated SPS curves for all eight algorithms over 100k steps.

*Analysis.* - **PPO** quickly ramps up to  $\sim 500$  SPS, dwarfing the  $\sim 100\text{--}200$  range of the DQN algorithms plus REINFORCE. - **SAC**, consistent with its high runtime/emissions, lingers around 80–100 SPS. - Minor differences exist among the DQN variants (e.g., PER may have a slightly heavier overhead due to priority calculations, etc.).



**Figure 75: Mean Emissions vs. IQM Evaluation for All 8 Algorithms.** (a) Human-normalized IQM vs. mean emissions. (b) Min-max-normalized IQM vs. mean emissions. Points are labeled by algorithm (c51, ddqn, dqn, dueling\_dqn, per, ppo, reinforce, sac).

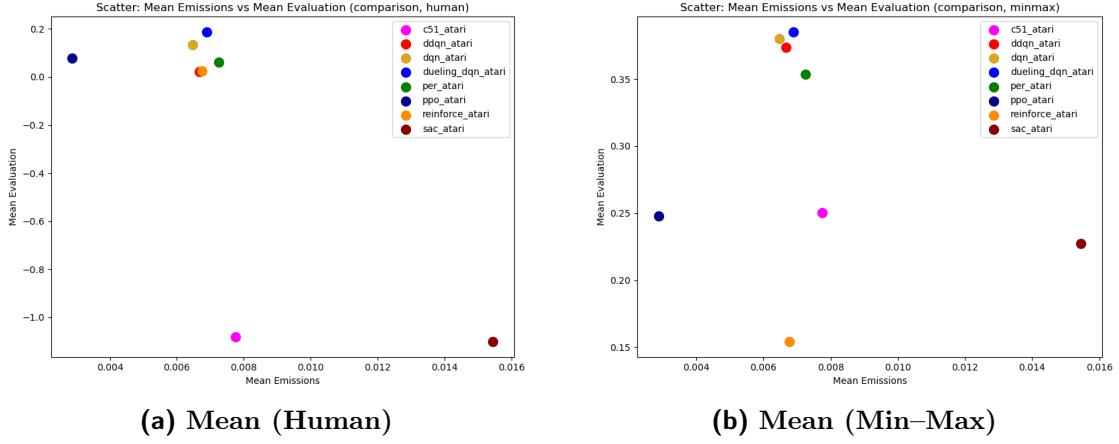
#### 4.6.4 Performance vs. Emissions (Scatter Plots)

To visualize how each algorithm trades off *emissions* and *performance*, we plot the **mean carbon footprint** (*x*-axis) against final evaluation (*y*-axis). Figures 75 and 76 each contain two subplots, one for **human-normalized** and one for **min–max normalized** performance. We show both **IQM** (i.e. interquartile mean) and **mean** returns in separate figures.

*Observation.* Across these four plots:

- **PPO** (blue point) sits at the far left ( $\approx 0.003 \text{ kg CO}_2$ ), with moderate returns in both IQM and mean. It is the most *energy-efficient*.
- **SAC** (red point) is the rightmost outlier ( $\approx 0.015\text{--}0.016 \text{ kg CO}_2$ ), with widely varying performance depending on the metric.
- **DQN-based variants** (C51 in pink, DDQN in red, DQN in gold, Dueling\_DQN in dark blue, PER in green) cluster in the  $\sim 0.006\text{--}0.008$  range of mean emissions. Some, e.g. *DQN* or *DuelingDQN*, achieve higher IQM or mean than others.
- **REINFORCE** (orange) lies around  $\sim 0.007$  emissions but yields relatively low returns under both human-norm and min–max.

Overall, these scatter plots underscore that **\*\*PPO\*\*** provides a strong balance of moderate/high returns with minimal carbon cost, while **\*\*SAC\*\*** can yield decent scores but at a high energy expense. The DQN-family methods vary: some (like *DuelingDQN*) approach or exceed PPO’s performance, but typically with  $2\text{--}3\times$  the emissions.



**Figure 76: Mean Emissions vs. Mean Evaluation for All 8 Algorithms.** (a) Human-normalized mean returns vs. emissions. (b) Min–max normalized mean returns vs. emissions.

**Table 37:** Example snippet of per-game returns (human norm) and emissions for selected environments. Full details for all games and normalizations appear in Appendix ??.

Env	Algorithm	Human Return (Mean)	CO <sub>2</sub> (Mean)	IQM
AlienNoFrameskip-v4	C51	-0.0149	0.00779	-0.0113
AlienNoFrameskip-v4	PPO	0.0103	0.00292	0.0084

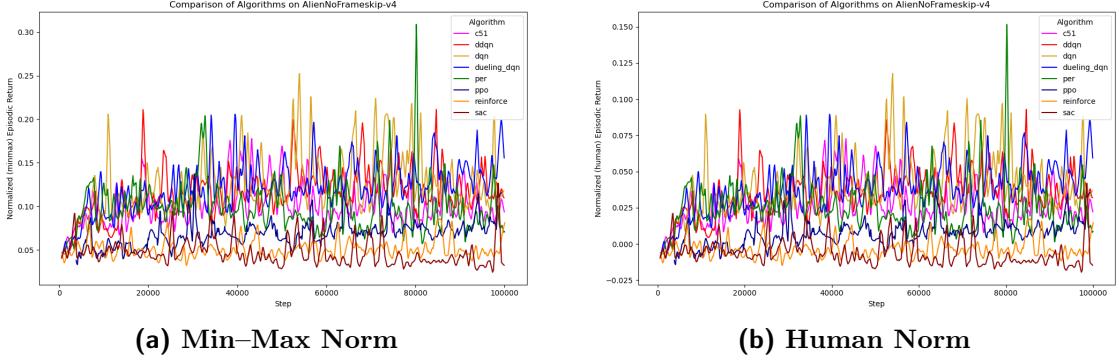
#### 4.6.5 Per-Game Details

To better understand how each algorithm performs on individual Atari environments, we collected \*\*episode-level data\*\* for each run, providing per-game final returns (human or min–max normalized) and per-game mean emissions. Table 37 below shows a \*\*small extract\*\* of these environment-level results for reference:

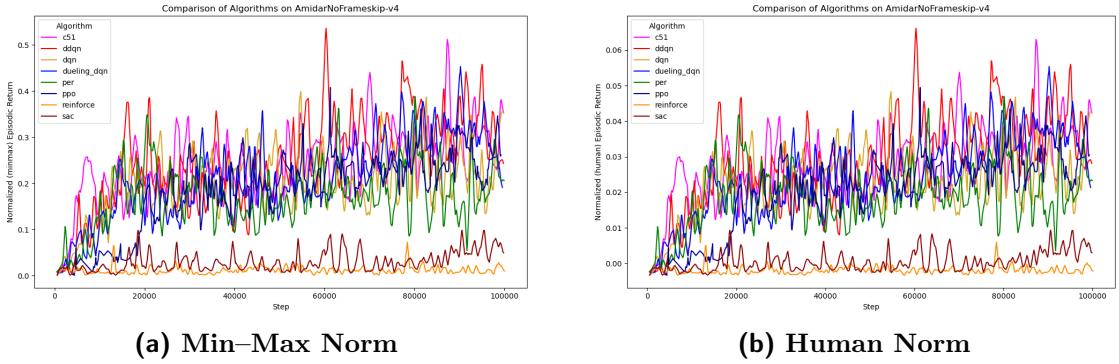
A full breakdown for every environment (Alien, Amidar, Assault, . . . , Pong) and each algorithm is included in **Appendix ??**, where we present the entire CSV data you provided (see the attached code block). We also provide environment-specific line plots of episodic returns (Figures Y1–Y8) for readers who wish to explore how training evolves in detail.

**Interesting Outliers** Here we highlight two outliers:

- **Boxing:** Some algorithms (SAC, REINFORCE) show large negative outliers in the human norm, though min–max compresses them.
- **Freeway:** PPO significantly outperforms DQN and other policy methods at just 100k steps, showing a near-monotonic climb.



**Figure 77: Alien:** Episodic returns for C51 (pink), DDQN (red), DQN (gold), DUELING\_DQN (dark blue), PER (green), PPO (navy blue), REINFORCE (orange), and SAC (dark red).



**Figure 78: Amidar:** Episodic returns across 100k steps.

#### 4.6.6 Per-Environment Episodic Returns (All Algorithms)

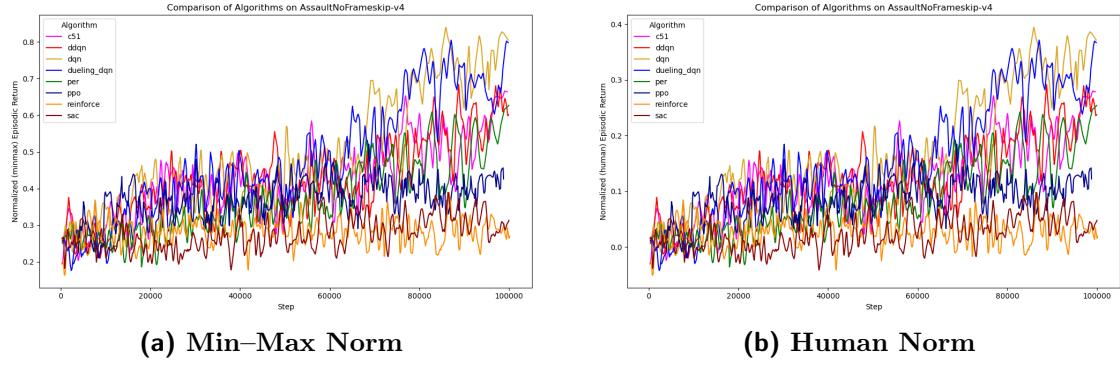
In this section, we display the over-time training curves (up to 100k steps) for *all eight* algorithms on each of the eight Atari environments under study, using both **min–max** and **human** normalization for the episodic returns. We discuss notable patterns in each environment below.

##### AlienNoFrameskip-v4

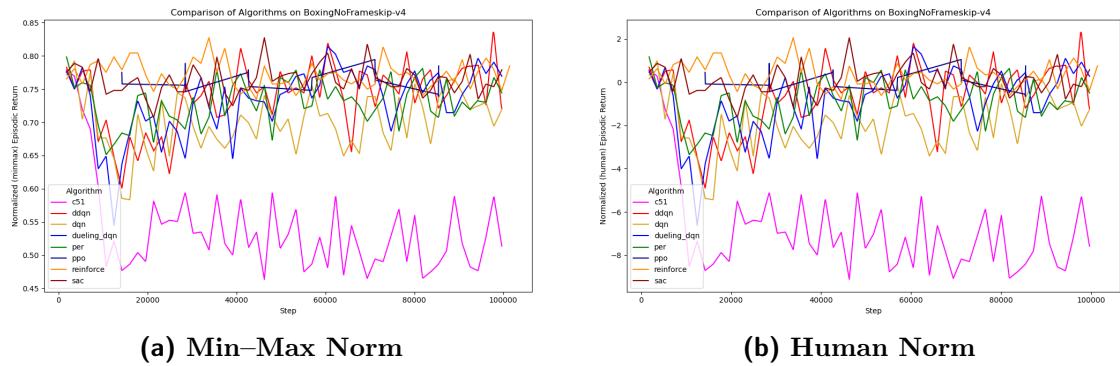
*Commentary.* DDQN (red) and DQN (gold) often spike above 0.15–0.20 in min–max, whereas PPO (blue) remains near 0.10–0.15. SAC (dark red) lingers lower but sometimes climbs late, REINFORCE (orange) generally stays near the bottom, and C51 (pink) shows moderate oscillations. Under human norm, the range compresses to near 0.0–0.1 for many runs.

##### AmidarNoFrameskip-v4

*Commentary.* DDQN (red) and Dueling\_DQN (dark blue) surpass 0.4–0.5 near 80k steps in the min–max figure, while C51 (pink) occasionally peaks around 0.5. PPO (blue) is more steady, around 0.2–0.3. SAC (dark red) and REINFORCE (orange) stay below 0.1. Under human norm, the overall scale is 0.0–0.06 for most algorithms, revealing smaller



**Figure 79: Assault:** Episodic returns across 100k steps.



**Figure 80: Boxing:** Episodic returns across 100k steps.

raw rewards in Amidar.

#### AssaultNoFrameskip-v4

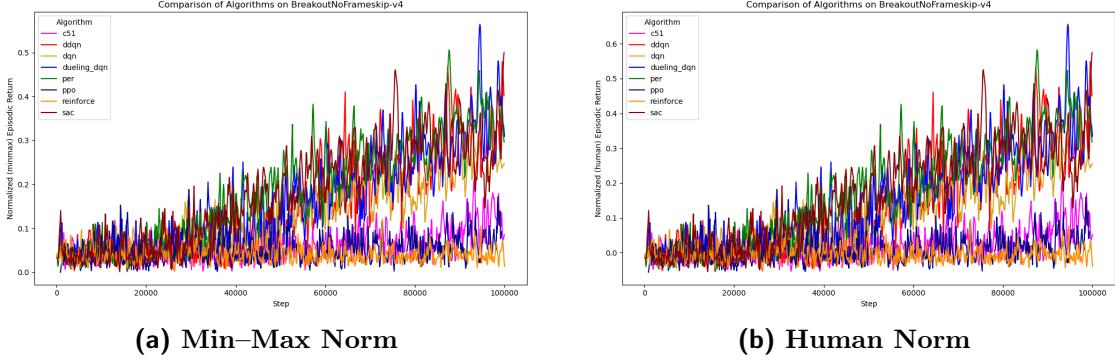
*Commentary.* Dueling\_DQN (dark blue) and DQN (gold) climb toward 0.8–0.85 late in training (min–max). PPO (blue) approaches 0.7, while reinforce (orange) remains under 0.4. C51 (pink) gradually ascends but ends around 0.6. Under human norm, the spread condenses to 0.0–0.4, consistent with higher raw scores in Assault.

#### BoxingNoFrameskip-v4

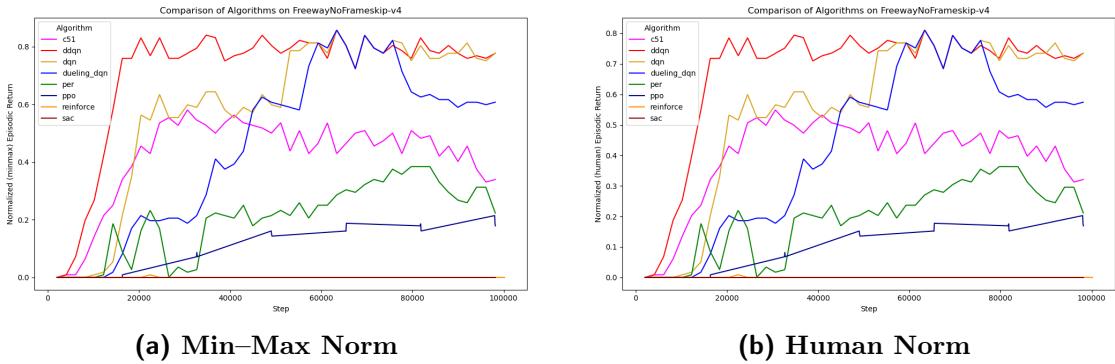
*Commentary.* Min–max values bunch around 0.5–0.8, with DDQN (red) sometimes near 0.8. C51 (pink) hovers 0.55–0.65. Meanwhile, in human norm, C51 and Reinforce see large negative dips below -5.0, showing how Boxing’s limited raw score range drastically affects the human scale. PPO (blue) and Dueling\_DQN (dark blue) stay near 0.0–1.0 in that scale.

#### BreakoutNoFrameskip-v4

*Commentary.* SAC (dark red) excels around 60k–100k steps, surpassing 0.5 in min–max and 0.4–0.5 in human norm. PPO (blue) lags near 0.2, and DQN (gold) settles around 0.2–



**Figure 81: Breakout:** Episodic returns across 100k steps.



**Figure 82: Freeway:** Episodic returns across 100k steps.

0.3. **Reinforce** (orange) stays near or below 0.1. Breakout highlights **SAC**’s potential for strong late-game performance, albeit at a higher computational cost (§4.6).

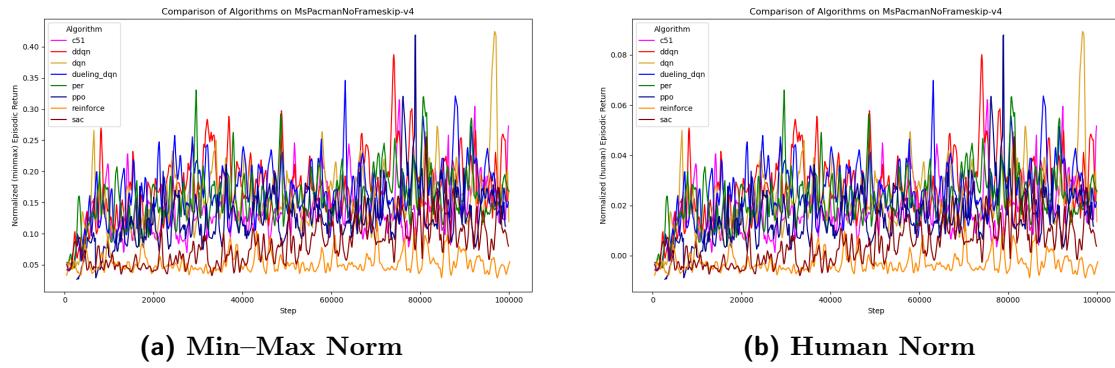
### FreewayNoFrameskip-v4

*Commentary.* Here, DDQN (red) rapidly climbs to 0.85+ in min–max by 15k steps, with C51 (pink) and DQN (gold) following suit. PPO (blue) only reaches 0.6, PER (green) hits 0.3, while Reinforce (orange) and SAC (dark red) remain near 0.0. In human norm, these large raw scores translate to 0.3–0.8 for the top algorithms, reflecting that Freeway has a high reward potential even early in training.

### MsPacmanNoFrameskip-v4

*Commentary.* No single algorithm dominates strongly, but dqn (gold) and ddqn (red) occasionally spike above 0.25 in min–max, while reinforce (orange) remains near 0.0–0.1. PPO (blue) hovers around 0.15–0.20, with c51 (pink) also in that range. Under human norm, the entire scale is fairly tight ( 0.0–0.08) due to MsPacman’s moderate raw reward potential within 100k steps.

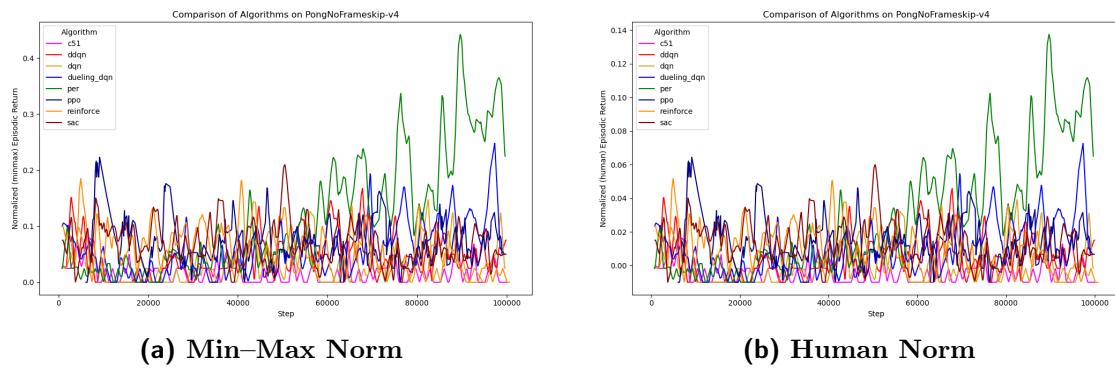
### PongNoFrameskip-v4



**(a) Min-Max Norm**

**(b) Human Norm**

**Figure 83: MsPacman:** Episodic returns across 100k steps.



**(a) Min-Max Norm**

**(b) Human Norm**

**Figure 84: Pong:** Episodic returns across 100k steps.

*Commentary.* PER (green) occasionally spikes near 0.3–0.4 in min–max, DQN (gold) has mid-range fluctuations, PPO (blue) and Reinforce (orange) remain under 0.2 for most runs. SAC (dark red) never climbs above 0.1. Because Pong’s raw scoring can be small or negative, the human norm figure is mostly 0.0–0.1. None of the algorithms achieve the large positive returns that DQN-based methods have historically reached with much longer training.

**Summary of Environment-Level Comparisons** These per-environment plots reinforce the observations from our aggregate metrics (Section 4.6). Notable highlights:

- **Freeway:** Some DQN variants (DDQN, DQN) rapidly approach near-maximum scores, overshadowing PPO and SAC.
- **Breakout:** SAC outperforms others late in training.
- **Boxing:** Min–max compression vs. large negative swings in human norm accentuates how the raw score range shapes these normalizations.
- **Amidar, Assault, Alien:** DuelingDQN and DQN often do well, while PPO is typically moderate, and SAC or Reinforce can lag behind.

Overall, each game’s reward structure significantly influences how algorithms evolve over 100k steps, corroborating the broader performance–emissions findings in Section 4.6.

#### 4.6.7 Summary

Bringing all eight algorithms together reveals distinct trade-offs among performance, emissions, and runtime:

- **PPO** remains the overall most energy-efficient method, finishing runs  $\sim 2.4\text{h}$  total. Its final returns are decent, though certain DQN variants can match or exceed it in specific tasks.
- **SAC** can excel (e.g. *Breakout*) but has the highest emissions and slowest SPS, taking over 11h to finish.
- **DQN-based variants** occupy a middle ground in both carbon footprint ( $\sim 0.006\text{--}0.008 \text{ kg CO}_2$ ) and total runtime (5–7h). Some, like DUELING\_DQN, achieve strong mean returns.
- **REINFORCE** similarly runs about 6h total, but yields lower performance than DQN methods or PPO in most tasks.

Hence, if one’s priority is *high performance* but with minimal compute cost, **PPO** or select DQN variants might be ideal. **SAC** offers potential but requires significantly more energy for short 100k-step training. These findings highlight a clear *trade-off* between

performance and sustainability. **PPO** is the most energy-efficient overall, while certain DQN variants can overtake it in raw performance yet emit  $2\text{--}3\times$  more CO<sub>2</sub>. **SAC** incurs the greatest overhead, offset by its strengths in a few tasks like *Breakout*. In Section 5, we discuss the broader implications of these results for real-world deployments and future research directions.

## 5 Implications of the Results

In this section, we interpret the findings from Section 4 in light of practical concerns such as deployment costs, carbon footprints, and sustainability requirements. We also consider how short-horizon benchmarks (100k steps) might shape our broader understanding of deep RL performance.

### 5.1 General Observations

Overall, our results show that:

- **Value-based** (DQN-family) methods (DQN, DoubleDQN, PER, DuelingDQN, C51) typically converge to moderate or high returns on several environments (e.g., *Freeway*, *Boxing*), but face stability challenges in some tasks (e.g., *Pong*, *MsPacMan*).
- **Policy-based** algorithms produce more varied performance: PPO reliably achieves competitive returns with lower emissions, whereas SAC can outperform others in a few environments (*Breakout*) but runs more slowly and emits significantly more CO<sub>2</sub>.
- **Short 100k-step horizon** constrains the potential for improvement, so many advanced techniques (e.g., Rainbow combinations, PER, etc.) do not show clear benefits over simpler methods within this limited training budget.

As a consequence, while these results confirm certain known trends—e.g., *DoubleDQN* mitigates Q-value overestimation, *PER* can accelerate training if allowed enough steps—the practical impacts at 100k steps are muted.

### 5.2 Energy Efficiency vs. Performance Trade-Off

A central theme of this work is the \*\*trade-off\*\* between achieving higher returns and incurring greater computational cost and carbon emissions. Our measurements reveal that:

- **SAC** consistently has the highest carbon footprint (on average  $\sim 0.015 \text{ kg CO}_2$ ), in part due to its off-policy updates and dual Q-network overhead, even though it sometimes outperforms other algorithms in late-stage learning.

- **PPO** exhibits the *lowest* emissions (around 0.0029 kg CO<sub>2</sub>) and shortest runtime (~ 2.4 hours total for 32 runs) but generally places only mid-to-high in final returns, depending on the environment.
- Most **DQN-based methods** lie in the middle (~ 0.006–0.008 kg CO<sub>2</sub> on average), with total runtimes of around 5–7 hours for 32 runs.

Hence, the classic adage of “no free lunch” holds: **SAC** can deliver strong scores on certain games but at a large computational and environmental cost, while **PPO** is impressively lean and still achieves respectable performance. For tasks where top-tier scores are not essential, a lower-emission method like PPO or DQN might suffice.

### 5.3 Practical Implications for AI Sustainability

For organizations aiming to balance *performance* with *sustainability*:

1. **Hardware Selection:** Using GPUs that CodeCarbon or W&B can track precisely (e.g., recent NVIDIA lines) greatly improves the accuracy of energy estimates. CPU usage is more difficult to capture reliably on certain OS/hardware combos.
2. **Short-Horizon Benchmarks:** Although many RL advances were proposed under multi-million-step training, the 100k-step regime can highlight efficiency differences relevant to real-world scenarios where time or resources are limited.
3. **Algorithm Choice:** If a moderate level of performance is acceptable, adopting **PPO** significantly lowers emissions while reducing training time. If the highest possible return is mandatory and the environment’s raw reward range suits it (e.g., *Breakout*), **SAC** might be worth its higher carbon cost.

This interplay of performance vs. overhead suggests that sustainability-conscious applications should carefully weigh the marginal returns gain from more computationally intense algorithms, especially if those gains only appear after 500k or 1 million steps.

### 5.4 Limitations and Future Work

A few constraints shape our interpretation:

**Limited Training Steps (100k).** Many popular DRL algorithms (Rainbow, distributional expansions, multi-step returns, etc.) truly shine beyond the 1 million-step mark. Our 100k-limit test can underestimate these methods’ potential.

**Restricted Environment Selection.** Although we tested 8 Atari games across 4 seeds (32 runs per algorithm), the full ALE suite has 55+ games. A broader set might reveal different rank orders, especially for highly complex tasks.

**Approximate CPU/RAM Tracking.** Due to Windows Intel Power Gadget deprecation and partial fallback modes, our CPU and RAM usage data rely on either TDP approximations or coarse telemetry from W&B. GPU tracking is more accurate, but the total system-level emissions remain an estimate.

**Stochastic Variation.** With only 4 seeds per environment, some especially negative outliers (Boxing’s large negative dips for certain seeds) can skew the aggregated means, though we mitigate this with IQM as recommended in [19].

Future work might extend training to 1–5 million steps for each method to see if advanced techniques eventually surpass simpler baselines in both performance and energy efficiency. Additionally, exploring specialized hardware or *hybrid HPC* could reveal new ways to reduce DRL’s carbon footprint.

## 6 Conclusions

In this final section, we synthesize the findings and propose next steps.

### 6.1 Summary of Findings

- **Baseline DQN** attains moderate performance across most environments, emitting  $\sim 0.0065 \text{ kg CO}_2$  on average over 100k steps. DoubleDQN reduces Q-value overestimation but does not yield a clear improvement in final returns at this short horizon.
- **PER** overhead raises emissions slightly ( $\sim 0.00725 \text{ kg CO}_2$ ), yet the 100k-step limit masks much of its usual advantage in accelerating learning.
- **DuelingDQN** and **C51** each show environment-specific benefits (e.g., *Boxing*, *Freeway*), but their aggregated returns remain similar to or slightly exceeding baseline DQN, with comparable runtime/emissions.
- **PPO** stands out as the most “eco-friendly,” completing runs in only 2.4 hours for all seeds/games combined and emitting a mere  $0.0029 \text{ kg CO}_2$ . It achieves respectable performance across many environments.
- **SAC** occasionally dominates a few tasks (*Breakout*) but exhibits the highest energy cost ( $0.015 \text{ kg CO}_2$ ), slow throughput (80–100 SPS), and large variance in returns.

### 6.2 Final Thoughts on Energy-Efficient Reinforcement Learning

The tension between *advanced techniques* and *computational overhead* emerges as a core challenge in DRL. Under a short 100k-step regime, simpler methods (PPO, baseline DQN) often provide a better ratio of performance to carbon cost, while fancier algorithms—like SAC or distributional DQN variants—struggle to gain a decisive edge before the run ends.

In realistic production or industry settings with limited time/budget for model tuning, prioritizing algorithms that quickly converge to adequate performance can yield significant energy savings. On the other hand, high-level research or competitive RL benchmarks often push beyond millions of steps, letting advanced methods eventually surpass simpler ones in raw returns.

### 6.3 Future Research Directions

Several pathways could extend this project:

**Longer Training Horizons.** Repeating these experiments at 1–5 million steps would clarify how advanced DQN tweaks or policy gradient expansions (Rainbow, TD3, etc.) surpass simpler baselines given more time. One might track the exact moment (in interactions) at which these enhancements repay their higher emissions.

**Wider Environment Coverage.** Our 8 chosen games reasonably sample different Atari mechanics, but including the full 26 Atari 100k or 55+ ALE tasks could reveal whether certain algorithms generalize better to more varied or obscure games.

**Alternative Normalizations.** In addition to “human” and “min–max” returns, some tasks might benefit from scoreboard-based normalization (comparing to public baselines) or reward shaping to unify the scale across tasks more strictly.

**Real-Time Emissions Minimization.** One future direction might involve *dynamic resource scheduling* or *carbon-aware training*, adjusting GPU usage or frequency if real-time energy prices or carbon intensities fluctuate throughout the day. This idea merges RL with HPC resource management for a truly “green AI.”

**Hybrid CPU–GPU Profiling.** Finally, a more robust toolchain for CPU and RAM tracking (e.g., Intel PCM on Linux) would help produce more accurate system-level carbon footprints, especially for on-policy methods that rely heavily on CPU-based data collection.

With these possible directions in mind, we hope that the insights gained from our 100k-step experiments can foster a broader conversation on *energy-efficient deep RL*, balancing performance with real-world sustainability needs.

**Comments on RL Algorithm Choice in Contemporary Applications.** Recent studies and industry practices have highlighted that the choice of reinforcement learning (RL) algorithm may not critically affect final outcomes in settings where human supervision is employed. For instance, early work in RL fine-tuning for large language models (LLMs) often utilized Proximal Policy Optimization (PPO) due to its robust performance and ease of integration with human feedback. However, subsequent investigations found that simpler

methods, such as Vanilla Policy Gradient (VPG), yield comparable results with a lower computational overhead and easier implementation [[raffel:exploring\\_rl\\_for\\_llms](#), [ouyang:ppo\\_vpg\\_comparison](#)]. Similarly, in emerging frameworks like DeepSeek, preliminary reports suggest that the specific choice of RL algorithm may be less decisive than originally assumed—with experimental results showing negligible differences in performance across various RL methods, although further systematic experimentation is required to substantiate these claims [[deepseek:blog](#), [zhang:rl\\_in\\_deepseek](#)]. This trend implies that, for many practical applications, selecting a more energy-efficient or computationally economical algorithm (e.g., VPG over PPO) might be preferable without incurring significant performance penalties.

## 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 \[cs.LG\]](https://arxiv.org/abs/1509.02971). [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 \[cs.LG\]](https://arxiv.org/abs/1801.01290). [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 \[cs.LG\]](https://arxiv.org/abs/1903.00374). [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](https://doi.org/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 \[cs.LG\]](https://arxiv.org/abs/2108.13264). [Online]. Available: <https://arxiv.org/abs/2108.13264>.
- [20] B. Courty et al., *Mlco2/codcarbon: V2.4.1*, version v2.4.1, May 2024. DOI: [10.5281/zenodo.11171501](https://doi.org/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/>.
- [24] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation*, 2018. arXiv: [1506.02438 \[cs.LG\]](https://arxiv.org/abs/1506.02438). [Online]. Available: <https://arxiv.org/abs/1506.02438>.