RLSB

Software Engineering for Artificial Intelligence

Reinforcement Learning Sustainability Benchmark

Report v1.0

Luca Strefezza*

February 8, 2025

*l.strefezza1@studenti.unisa.it

Abstract

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

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

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

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

Contents

1 Context

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

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

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

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

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

2 Goals

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

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

2. compare the performance of these algorithms in terms of their ability to achieve high scores on the given task;

3. analyze the trade-offs between performance and energy consumption to identify the most efficient algorithms;

4. provide a comprehensive report that can guide practitioners in selecting the appropriate DRL algorithms based on specific use-case requirements.

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

3 Methodological Steps

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

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

3.1 Algorithms Selection

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

3.1.1 Value-Based Methods

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

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

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

3.1.2 Policy Gradient Methods

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

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

3.2 Task Selection

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

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

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

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

3.3 Experiment Setup

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

3.3.1 Number of Runs

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

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

median to aggregate performances scores across tasks to summarize the results of the various runs, but this metrics are not the best way to do so because they ignore the statistical uncertainty inherent in performing only a few runs.

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

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

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

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

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

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

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

3.3.2 Data Logging and Storage

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

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

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

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

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

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

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

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

3.3.3 Development and Execution Environment

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

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

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

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

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

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

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

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

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

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

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

3.3.4 Atari Environment Configuration

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

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

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

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

Episodic life and fire reset:

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

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

Observation preprocessing:

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

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

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

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

The primary difference between our setup and the *v5* environments is the exclusion of sticky actions. While sticky actions can enhance generalization in long training regimes by preventing the agent from overfitting to deterministic game mechanics, their benefits are less relevant in our setting, where each run is limited to only 100k interactions. Under such a short training horizon, the additional stochasticity introduced by sticky actions would significantly degrade training stability and learning efficiency, leading to noisier performance estimates thus removing them is not only not problematic, but

almost mandatory. Furthermore, the use of sticky actions is not as ubiquitous in the reinforcement learning literature as other preprocessing steps, making their exclusion a reasonable choice also for comparability with prior work.

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

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

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

3.3.5 (Hyper)Parameter Configurations

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

Parameters

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

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

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

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

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

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

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

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

3.3.6 Evaluation

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

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

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

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

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

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

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

3.4 Data Analysis and Visualization

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

3.4.1 Log Collection and Merging

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

3.4.2 Normalization of Returns

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

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

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

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

3.4.3 Interpolation and Aggregation

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

In our pipeline, we therefore:

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

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

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

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

3.4.4 Plot Generation and CSV Output

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

Overall, this approach provided:

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

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

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

4 Preliminary Results and Findings

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

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

4.1 Experiment Setup Adjustments

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

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

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

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

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

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

4.2 DQN-Based Algorithms

We present results for five different DQN-based algorithms, including baseline DQN and its extensions. Each algorithm is analyzed individually before an overall comparison.

4.2.1 DQN (Baseline)

In our benchmark, *Deep Q-Network (DQN)* serves as the baseline algorithm against which we compare more advanced DQN variants and policy gradient methods. We trained DQN on the selected eight Atari games, running four distinct seeds per game for a total of 32 runs. Table 2 summarizes the main hyperparameters used; the total number of training steps was 100 000 in each run.

Parameter	**Value**
`learning_rate`	1×10^{-4}
`buffer_size`	10 000
`batch_size`	32
`target_network_frequency`	1000
`exploration_fraction`	0.1
`start_e`, `end_e`	$1.0 \rightarrow 0.01$
`learning_starts`	1000
`train_frequency`	4

Table 2: Key hyperparameters for the DQN baseline. All runs used a discount factor $\gamma = 0.99$, one environment (`num_envs`=1), and `cuda` enabled.

Hyperparameters.

Training Dynamics. Figure 1 shows the *episodic length* during training, aggregated over the 32 runs via linear interpolation onto a common step axis. We observe that the *mean* episode length hovers around 3500–4000 steps for most of training, with substantial variability among runs (up to 8000 steps in the most extreme cases). In some runs, episodes became quite short, indicating early terminations and resulting in the large min–max envelope.

Figure 2 presents the *episodic return* curves. The mean performance remains slightly below zero for most of the training, reflecting the difficulty of some games given only 100 000 steps of interaction. In a few runs, we observe short-lived spikes in performance, but overall there is high variance (the min–max band ranges from about -10 to $+4$). These preliminary results suggest that DQN can struggle to achieve consistently positive returns in the chosen set of Atari games within 100k steps.

Loss and Value Estimates. To further diagnose training behavior, Figure 3 shows the evolution of the estimated *Q-values* over training. Notably, the mean Q grows from near zero to around 4 or 5 by step 100 000, but with large min–max variability (some runs estimate Q-values above 10, while others stay near 0). Meanwhile, the *TD loss* (Figure 4) starts near zero and steadily increases in many runs, peaking in some seeds above 1.0 or even 3.0 toward the end of training. These indicators suggest that DQN's learning process under these hyperparameters and time horizons remains quite unstable, highlighting the challenges of sample efficiency in a short 100 000-step regime.

Energy Consumption. Table 3 shows the average carbon emissions for DQN over the eight games (four seeds each). On average, **0.00647 kg** of CO_2 was emitted per run (with a standard deviation of 0.00026).[1] While this raw number is relatively small, it is still a meaningful measurement in comparing algorithms' resource usage—particularly if scaled up to longer training horizons or larger runs.

Algorithm	mean	std	median	q25	q75	min	max	iqmean
DQN	0.00647	0.00026	0.00634	0.00630	0.00658	0.00616	0.00699	0.00637

Table 3: Carbon emissions for DQN (*kg CO_2 eq.*) aggregated across 32 runs.

Observations Overall, DQN shows moderate performance with high variance and relatively low returns within the 100 000-step constraint. The Q-values and TD loss reveal ongoing instability, suggesting that more advanced extensions or longer training might be necessary to achieve robust performance on these Atari tasks. In terms of energy footprint, DQN remains relatively lightweight in absolute terms, though differences may become more pronounced when compared with other algorithms (Section 4.2.2 and beyond) or when scaled to larger training budgets.

[1] Exact units can vary depending on CodeCarbon's tracking settings; here we report total kg CO_2 equivalent for each run.

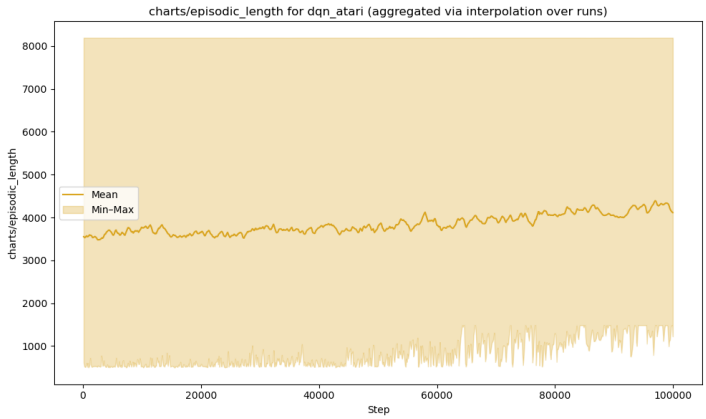

Figure 1: DQN episodic length (`charts/episodic_length`), showing mean (line) and min–max band (shaded).

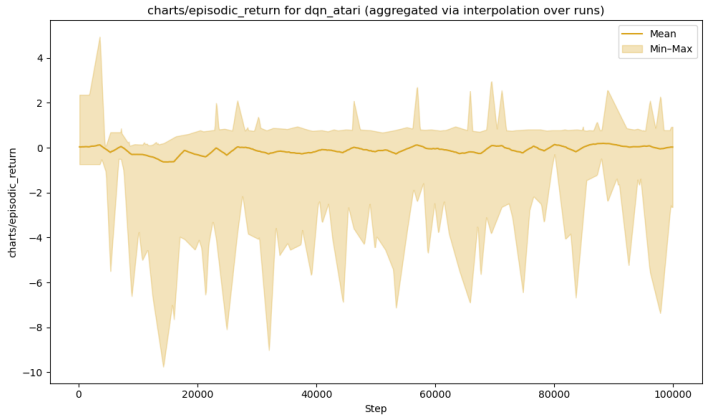

Figure 2: DQN episodic return (`charts/episodic_return`), showing mean (line) and min–max band (shaded).

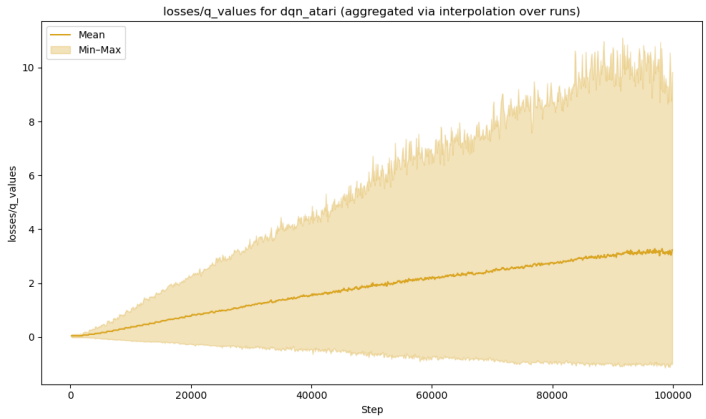

Figure 3: Estimated Q-values (`losses/q_values`) for DQN, showing mean and min–max across runs.

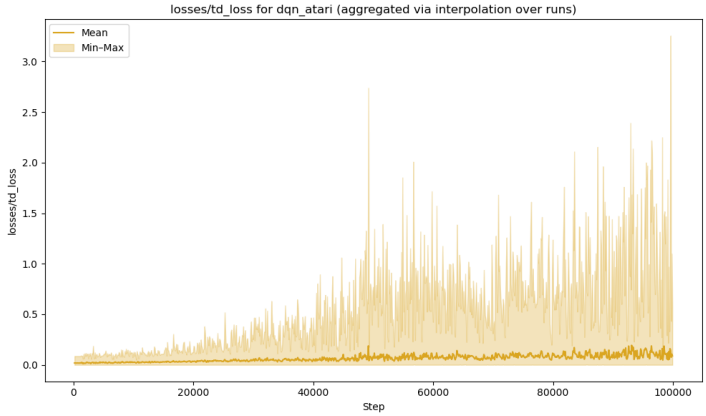

Figure 4: TD loss (`losses/td_loss`) for DQN, illustrating the growing variance across runs.

In the following subsections, we examine how enhancements such as Double DQN and Prioritized Experience Replay improve upon this baseline in terms of both performance and energy consumption.

4.2.2 Double DQN

- Differences from baseline DQN.
- Training execution and results.
- **Comparison with DQN:** Score improvement, energy trade-offs.

Differences from baseline DQN. Training execution and results. Comparison with DQN in terms of score, stability, and energy efficiency.

4.2.3 Prioritized Experience Replay

- How prioritization improved learning efficiency.
- Energy-performance trade-off compared to DQN and Double DQN.

Comparison with DQN and Double DQN. Did it lead to better sample efficiency? Energy consumption comparison.

4.2.4 Dueling DQN

- Did dueling networks reduce variance?
- Performance and energy efficiency compared to other DQN methods.

Impact on stability. Did it reduce variance in Q-value estimates? Performance and energy trade-offs.

4.2.5 C51 (Categorical DQN)

- Impact of distributional reinforcement learning.
- Did C51 improve sample efficiency?

Did distributional learning improve performance? Energy efficiency vs. performance trade-off.

4.3 Overall Comparison of DQN-Based Algorithms

- Graphs comparing all DQN-based methods.
- Summary table with:
  - Final performance scores per game.
  - Total training duration.
  - Energy consumption.
- Discussion: Which method offers the best balance between efficiency and performance?

Graphs comparing all 5 DQN-based algorithms on the same scale. Tables summarizing: Performance (average final score per game). Training duration. Total energy consumption. Key takeaways: Which is the best energy-performance trade-off?

4.4 Policy-Based Algorithms

This section presents results for the three policy gradient methods.

4.4.1 REINFORCE

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

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

4.4.2 PPO (Proximal Policy Optimization)

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

Did clipping help stabilize training? Efficiency compared to REINFORCE.

4.4.3 SAC (Soft Actor-Critic)

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

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

4.5 Overall Comparison of Policy-Based Algorithms

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

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

4.6 Cross-Category Comparison: DQN vs. Policy Gradient

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

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

5 Implications of the Results

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

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

5.1 General Observations

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

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

5.2 Energy Efficiency vs. Performance Trade-Off

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

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

5.3 Practical Implications for AI Sustainability

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

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

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

- How could energy-efficient reinforcement learning impact research and business decisions?

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

5.4 Limitations and Future Work

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

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

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

6 Conclusions

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

6.1 Summary of Findings

- What were the most important insights gained from the experiments?

- Which algorithms proved to be the best trade-off between energy efficiency and performance?
- Which models consumed excessive energy without substantial performance benefits?

6.2 Final Thoughts on Energy-Efficient Reinforcement Learning

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

6.3 Future Research Directions

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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