uc3m | Universidad Carlos III de Madrid

Dual Bachelor in Data Science and Engineering and Telecommunication Technologies Engineering.
2024-2025

Bachelor Thesis

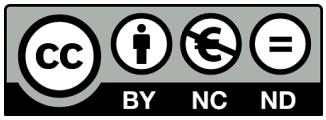
Reinforcement learning algorithms for training software agents in video games

---

Daniel Toribio Bruna

Iván González Díaz

Leganés, September 2025



SUMMARY

This thesis explores the application of Deep Reinforcement Learning (DRL), specifically the Deep Q-Network (DQN) algorithm, to the Atari game Breakout, with the goal of improving the agent's performance and generalization. The project investigates how architectural changes, such as incorporating Gated Recurrent Units (GRUs) to capture temporal dependencies, affect learning, alongside key training elements like experience replay memory size and various subsampling strategies. To assess the robustness and generality of the developed training pipeline, the best-performing configuration was applied to other Atari games, namely Pong and Space Invaders, without modifying the training setup. These experiments evaluated whether the same design principles are transferable across games with differing reward structures and dynamics. The results provide insights into the role of memory size, temporal modeling, and sampling methods in building more adaptable and effective DRL agents.

Keywords: Deep Reinforcement Learning, Deep Q-Network, Gated Recurrent Units, Experience Replay, Subsampling Strategies, Atari Games, Breakout, Generalization

DEDICATION

With this work, I bring to a close a very important and meaningful period of my life. Despite the many years of hard work and sacrifice, the happy moments stand out, enriched by the many people who have influenced me, and to whom I am deeply grateful.

This chapter of my life would not have been possible without the friends I've made along the way. I extend my heartfelt thanks to them for all their support and assistance. I also fondly remember the teachers and classmates who have left their mark on me, guiding me through the countless decisions I've made over the past five years.

I am especially thankful to my tutor, Iván, for his dedication and for helping me complete this work. Most importantly, I am grateful to him for trusting me and giving me the opportunity to collaborate with him on one of his projects.

Finally, I owe my deepest gratitude to my family for their unwavering support throughout the years. To my parents, thank you for your love, encouragement, and advice—for celebrating my triumphs and for guiding me to where I am today.

Thank you all.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Deep Reinforcement Learning (DRL) has emerged as a powerful framework for solving sequential decision-making problems, particularly in environments with high-dimensional state spaces [1]. One of the most influential algorithms in this area is the Deep Q-Network (DQN), which combines Q-learning with deep neural networks to approximate value functions directly from raw input [2], such as image frames in video games [3]. Since its introduction, DQN has achieved remarkable success in a variety of Atari 2600 games, demonstrating the ability to learn effective control policies directly from pixels.

This project explores the application of DQN to the classic Atari game Breakout, with a focus on enhancing the agent's ability to capture temporal dependencies in the environment. To this end, it is investigated the integration of Recurrent Neural Networks (RNNs), specifically Gated Recurrent Unit (GRU) layers, into the DQN architecture.

In addition to architectural modifications, this project also examines key training components that influence learning efficiency and policy quality. Specifically, it is analyzed the impact of the experience replay buffer size and different subsampling strategies used during training.

To evaluate the robustness of the best-performing configuration, a generalization experiment by training the configuration to other Atari games with differing dynamics and reward structures, such as Space Invaders and Pong, is conducted. This shows whether the configuration is transferable to new environments or it is highly task-specific.

Overall, the project contributes insights into the role of temporal modeling, memory size, and data sampling in deep reinforcement learning, while also addressing the broader question generalization across tasks.

1.1. Motivation of the project

Deep reinforcement learning (DRL) has shown impressive capabilities in learning control policies directly from raw sensory inputs. However, training effective DRL agents remains computationally intensive and often sensitive to hyperparameters, architectural choices, and training strategies. A major challenge lies in enabling agents not only to learn from one specific environment, but also to perform well across different tasks without the need to redesign the training pipeline for each new game.

This project is motivated by the desire to better understand how different components of the DQN training process, such as network architecture, memory configuration, and data sampling strategies, affect learning performance and generalization across environments. In particular, recurrent neural networks (RNNs) offer the potential to capture temporal dependencies in partially observable environments, which could enhance the agent's

understanding of the dynamics and improve decision-making.

Breakout was chosen as the primary environment due to its balance of complexity and accessibility. It requires the agent to understand both temporal sequences and reward-driven strategies, making it well-suited for evaluating the effects of recurrent architectures and training techniques. Beyond Breakout, the same training pipeline was applied, without modification, to other Atari games with different reward structures and dynamics, such as Space Invaders and Pong. In this context, generalization refers not to transferring the trained model directly, but rather to evaluating whether the same training configuration can lead to effective learning in new environments.

By analyzing how well the training strategy and architecture carry over to different games, this project aims to provide insights into the design of more general-purpose DRL agents that are less reliant on task-specific tuning.

1.2. Objectives

The main objective of this project is to explore and improve the performance and generalization of Deep Q-Network (DQN) agents through architectural and training modifications. Specifically, the project focuses on the following goals:

- **Integrate recurrent neural networks into the DQN architecture:** Implement Long Short-Term Memory (LSTM) layers to enable the agent to capture temporal dependencies and improve decision-making in environments like Breakout.
- **Evaluate the impact of experience replay memory size:** Analyze how different replay buffer sizes affect learning stability, convergence speed, and final performance.
- **Explore various subsampling strategies:** Investigate the effect of different methods for sampling transitions from the replay memory on the agent's ability to learn effectively, especially in environments with sparse rewards.
- **Assess generalization across games:** Train the agent using the best configuration on Breakout and evaluate its performance on other Atari games such as Space Invaders and Pong to determine how well the learned policy transfers across environments.
- **Identify factors that influence cross-task generalization:** Gain insights into the properties of environments (e.g., reward frequency, action space, game dynamics) that affect the transferability of policies.

1.3. Structure of the report

The list below describes the structure of this report to facilitate comprehension.

- **Chapter 1: Introduction**
  Chapter 1 describes the goals and underlying motivation of the project. It also provides an overview of the report's structure.

- **Chapter 2: State of the art**
  Chapter 2 includes a discussion of Reinforcement Learning and Deep Learning fields, and an analysis of related works, providing the foundation and context for the research conducted in this thesis. At the end of the chapter, the contributions of this project are emphasized.

- **Chapter 3: Methodology**
  Chapter 3 describes the tools, libraries and resources required to build models, train models, analyze results, and describes the benchmark model.

- **Chapter 4: Description of the Deep Reinforcement Learning System**
  Chapter 4 explains the workflow followed to create a system that is able to train and evaluate DQN agents.

- **Chapter 5: Experiments and analysis of results**
  Chapter 5 describes what experiments have been carried out and the obtained results are discussed.

- **Chapter 6: Conclusions**
  Chapter 6 reviews the project's objectives and findings, highlighting the significance of the results, the limitations and provide potential directions for future research.

- **Appendix A: Project Planning**
  Appendix A describes the different tasks done during the development of the project, dependencies between them and the organization of these tasks, illustrating it with a Gantt chart.

- **Appendix B: Budget**
  Appendix B collect all the costs of the project to give an approximated total cost of the project.

- **Appendix C: Regulatory Framework**
  Appendix C summarizes the regulations and laws that affect the project and Artificial Intelligence field.

- **Appendix D: Socio-Economic Framework**
  Appendix D provides an overview of the socio-economic context of the video game industry, highlighting both the opportunities and challenges that deep reinforcement learning introduces within this domain.

- **Appendix E: Video of the best agent developed playing the Atari games**
  Appendix E has links to videos to show how the best agents play each game.

2. STATE OF THE ART

This section introduces the fields central to the project: Deep Learning, Reinforcement Learning, and Deep Q-Learning. It also presents a discussion and comparison of previous studies that share certain characteristics with the current work, highlighting similarities and differences. These studies were selected for their relevance to Deep Q-Learning tasks and their significant contributions to the broader fields of Reinforcement Learning and Deep Learning, which form the foundation of this project.

Towards the end of the chapter, the unique contributions of this project will be highlighted in relation to other research in the same domain.

2.1. Introduction to Deep Learning

Deep Learning is a subset of machine learning, see Figure 2.1, that employs artificial neural networks with multiple layers—often referred to as deep neural networks—to model complex patterns and representations in data [4]. It has revolutionized many fields including computer vision [5], natural language processing [6], and reinforcement learning [1], by enabling algorithms to automatically learn hierarchical features from raw inputs.

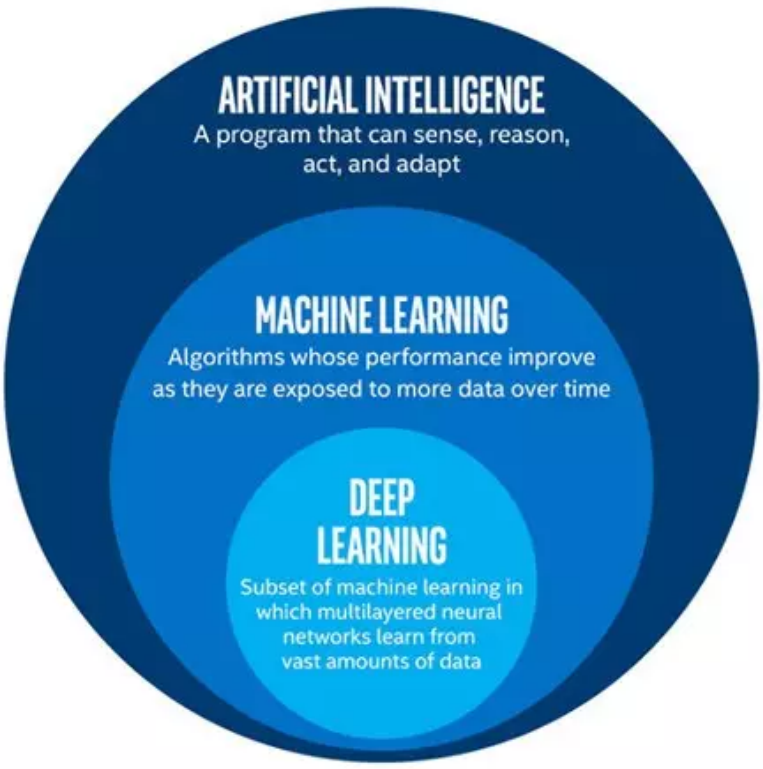


Fig. 2.1. Deep Learning and Machine Learning as subfields of Artificial Intelligence [7].

The power of deep learning arises from its ability to approximate highly nonlinear functions through multiple layers of nonlinear transformations. This hierarchical learning structure allows the model to extract progressively more abstract and useful features, making it particularly effective for high-dimensional data such as images, audio, and sequential information [4] [8].

2.1.1. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specialized neural networks designed to process data with a grid-like topology, such as images [9]. Unlike fully connected networks, CNNs exploit the spatial structure of the input by applying convolutional filters that slide over local regions. This weight-sharing mechanism reduces the number of parameters and captures spatial hierarchies effectively [10].

A typical CNN architecture consists of several layers including convolutional layers, activation functions, pooling layers, and fully connected layers. The convolutional layers apply multiple filters to detect various local features such as edges, textures, or shapes. Pooling layers then reduce the spatial dimensions, providing translation invariance and reducing computational complexity [4]. A common CNN architecture is shown in Figure 2.2, more precisely, the AlexNet architecture which was developed for image classification among 1,000 classes [11].

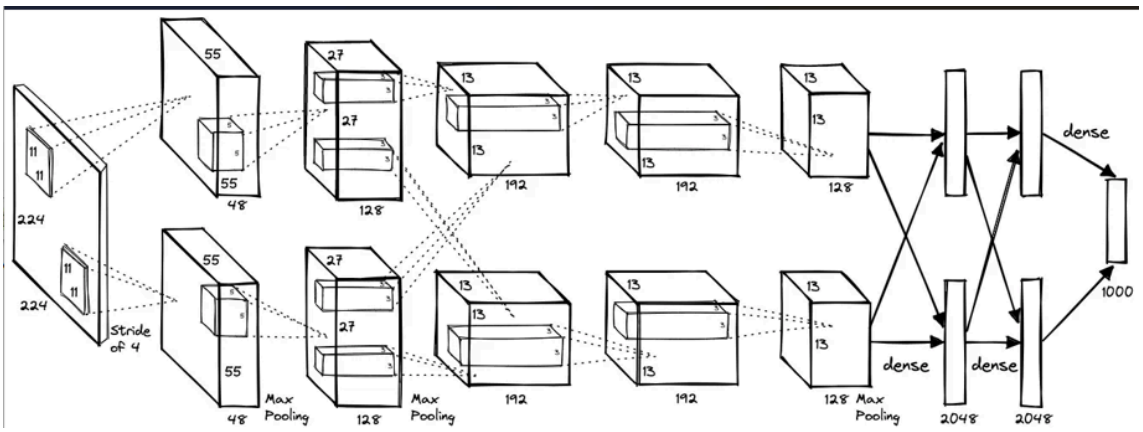


Fig. 2.2. AlexNet architecture [11].

CNNs have demonstrated outstanding performance in image classification, object detection, and many other vision-related tasks [12]. Their ability to automatically learn relevant features from raw pixels makes them ideal for environments where manual feature engineering is infeasible, for example in frames of a game.

2.1.2. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a hidden state that evolves over time, capturing dependencies across time steps. This makes them well-suited for tasks involving time series, language modeling, and any domain where context and order are important [13].

However, vanilla RNNs suffer from the *vanishing and exploding gradient problems*, where gradients become too small or too large during backpropagation, making it difficult to learn long-term dependencies. To address these limitations, gated architectures such as the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) were introduced [4] [14].

The GRU is a simplified gated architecture that combines the forget and input gates of the LSTM into a single update gate, along with a reset gate to control how much past information to forget. This design reduces computational complexity while retaining the ability to capture long-range dependencies effectively [14]. A comparison of a simple RNN, a LSTM and a GRU is shown in Figure 2.3.

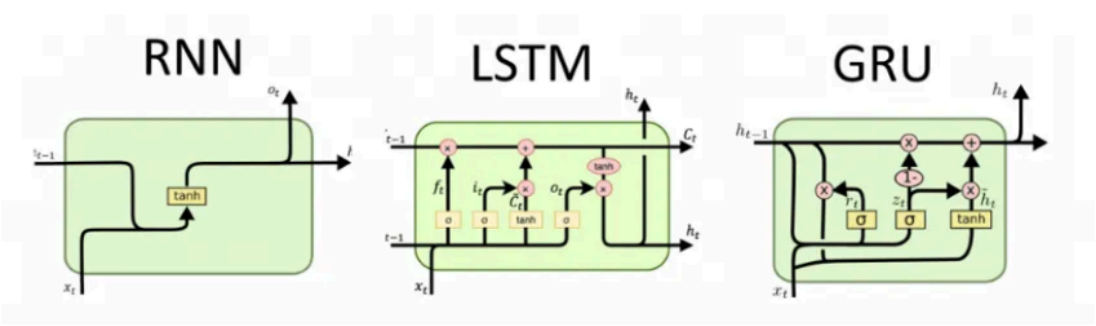


Fig. 2.3. Comparison of different recurrent modules [15].

The GRU's update equations can be summarized as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.1}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.2}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.3}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.4}$$

where z_t is the update gate, r_t is the reset gate, $\tilde{h}_t$ is the candidate activation, h_t is the hidden state at time t, x_t is the input, and σ denotes the sigmoid activation function.

Intuitively, the GRU works by deciding, at each time step, how much past information should be kept and how much of the new input should influence the current state.

The **update gate** z_t controls the balance between the previous hidden state h_{t-1} and the candidate activation $\tilde{h}_t$. A value of z_t close to 1 means that the unit mostly keeps the new candidate information, while a value close to 0 means it preserves more of the past state. This allows the GRU to adaptively remember or forget information over time without needing a separate memory cell, as in LSTMs.

The **reset gate** r_t determines how much of the past state should influence the computation of the candidate activation $\tilde{h}_t$. When r_t is close to 0, the model effectively ignores the previous hidden state, focusing on the current input x_t. This is useful for discarding irrelevant past information when making predictions.

Finally, the candidate activation $\tilde{h}_t$ combines the current input with the filtered past information (controlled by r_t) and applies a non-linear transformation to create a potential new state. The hidden state h_t is then a smooth interpolation between the old state and the candidate, weighted by the update gate.

GRUs have been widely used in various sequence modeling applications due to their balance between performance and computational efficiency [14].

2.2. Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a foundational area of machine learning concerned with how an agent learns to make a sequence of decisions through interaction with a dynamic environment [16]. Unlike supervised learning, where the model learns from labeled input-output pairs, RL is characterized by learning from experience and delayed feedback. This paradigm is inspired by behavioral psychology, where learning is driven by trial-and-error and the consequences of actions [2] [1].

At its core, RL involves an *agent* that interacts with an *environment* over discrete time steps. At each time step t, the agent observes the current *state* $s_t \in \mathcal{S}$, selects an *action* $a_t \in \mathcal{A}$, and as a result, receives a *reward* $r_t \in \mathbb{R}$ and transitions to a new state s_{t+1}. The agent's objective is to learn a strategy, or *policy*, that maximizes some notion of cumulative future reward [2].

2.2.1. Formal Definition: Markov Decision Processes

As Kai Arulkumaran explains [1], most RL problems are formalized using the framework of *Markov Decision Processes (MDPs)*. An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is a finite (or continuous) set of environment states.
- $\mathcal{A}$ is a finite (or continuous) set of actions available to the agent.
- $P(s'|s,a)$ is the transition probability function, defining the probability of transitioning to state s' when taking action a in state s.
- $R(s,a)$ is the reward function, specifying the immediate reward received after executing action a in state s.
- $\gamma \in [0,1]$ is the discount factor, determining the importance of future rewards.

The *Markov property* assumes that the next state depends only on the current state and action, and not on the sequence of past states and actions. This property enables tractable modeling and learning [17].

2.2.2. Policies and Value Functions

A *policy* π is a mapping from states to actions, either deterministically $\pi(s) = a$, or stochastically $\pi(a|s) = \mathbb{P}(a|s)$. The aim in RL is to learn an optimal policy π^* that maximizes expected cumulative reward [16].

Two key concepts in RL are the *state-value function* $V^\pi(s)$ and the *action-value function* $Q^\pi(s,a)$:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right] \tag{2.5}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right] \tag{2.6}$$

These functions quantify how good it is to be in a given state or to take a given action under policy π. The optimal policy π^* corresponds to the one that yields the highest value for all states [2].

Intuitively, the state-value function $V^\pi(s)$ measures the expected cumulative reward (the *return*) that an agent will collect if it starts in state s at time 0 and then follows policy π forever. Formally the return is $G_0 = \sum_{t=0}^{\infty} \gamma^t r_t$, and the expectation $\mathbb{E}_\pi$ averages over the randomness coming from the policy's action selection and the environment's state transitions. The discount factor $\gamma \in [0,1)$ controls how much the agent values immediate rewards relative to future rewards: a γ close to 0 produces a short-sighted agent, while a γ close to 1 makes the agent value long-term outcomes and also ensures convergence of the infinite sum when rewards are bounded [2].

The action-value function $Q^\pi(s,a)$ gives the expected return if the agent starts in state s, takes action a immediately, and thereafter follows policy π. Because Q^π conditions on

the first action, it is the natural object for comparing different actions that could be taken in the same state. The two functions are related by

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi}(s,a)],$$

i.e. the value of a state equals the expected action-value under the policy.

Both functions satisfy recursive (Bellman) identities that make estimation possible:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\Big[r(s,a) + \gamma\, \mathbb{E}_{s'\sim P(\cdot|s,a)}[V^{\pi}(s')]\Big], \tag{2.7}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}\Big[r(s,a) + \gamma\, \mathbb{E}_{a'\sim\pi(\cdot|s')}[Q^{\pi}(s',a')]\Big], \tag{2.8}$$

where $P(s' \mid s,a)$ denotes the environment transition probabilities and $r(s,a)$ the expected immediate reward. These recursive forms are the basis for many algorithms: they allow iterative policy evaluation (estimating V^{π} or Q^{π}) and policy improvement (making the policy greedier with respect to current value estimates).

At optimality we write $V^{*}(s) = \max_{\pi} V^{\pi}(s)$ and $Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$. The Bellman optimality equations are

$$V^{*}(s) = \max_{a} Q^{*}(s,a), \tag{2.9}$$

$$Q^{*}(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}[r(s,a) + \gamma \max_{a'} Q^{*}(s',a')]. \tag{2.10}$$

A policy that picks $\arg\max_{a} Q^{*}(s,a)$ at every state is an optimal policy π^{*}.

From a practical perspective, V^{π} and Q^{π} are learned from data in different ways: Monte Carlo methods estimate returns directly, low bias, high variance, temporal-difference (TD) methods bootstrap from current estimates, lower variance but biased, and dynamic programming methods use a known model P. Value functions are central to value-based methods like Q-learning or DQN and to actor-critic methods where a learned value reduces gradient variance and guides policy updates [2].

2.2.3. Exploration vs. Exploitation

A fundamental challenge in RL is the *exploration-exploitation trade-off* [18]. The agent must balance:

- **Exploitation**: choosing the best-known action to maximize immediate reward.
- **Exploration**: trying new actions that may yield higher rewards in the long term.

Efficiently managing this trade-off is critical for effective learning. Common strategies include *epsilon-greedy* policies, *softmax* action selection, and *Upper Confidence Bound (UCB)* methods [2].

2.2.4. Learning Approaches

RL algorithms can be broadly categorized based on how they learn value functions or policies [19] [20]. These are the most common RL algorithms:

1. **Value-based methods**: Learn an estimate of the value function, from which a policy is derived. A classic example is *Q-learning*, which updates estimates of $Q(s, a)$ using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

2. **Policy-based methods**: Learn a parameterized policy directly without estimating value functions. These are typically optimized using policy gradient techniques.

3. **Actor-Critic methods**: Combine both value-based and policy-based methods. An actor updates the policy, while a critic evaluates it using a value function.

Each approach has its strengths. Value-based methods are often more sample-efficient, while policy-based methods handle high-dimensional or continuous action spaces better [20].

2.2.5. Applications and Significance

Reinforcement Learning has been successfully applied to a wide range of domains, from game-playing in environments such as AlphaGo [21] or Atari [22], to robotics [23], recommendation systems [24], autonomous vehicles [25], and finance [26]. Its appeal lies in its ability to learn complex behaviors from sparse and delayed feedback, making it well-suited for dynamic, real-world decision-making tasks [2].

However, classical RL techniques often struggle in environments with large or continuous state spaces. This limitation has motivated the development of *Deep Reinforcement Learning*, which leverages deep neural networks to approximate value functions and policies, a subject further explored in the following sections.

2.3. Deep Q-Learning

Deep Q-Learning (DQL) is a reinforcement learning algorithm that integrates Q-learning with deep neural networks to handle high-dimensional state spaces. Traditional Q-learning methods, which use tabular representations of the action-value function $Q(s, a)$, become infeasible in environments with large or continuous state spaces due to the exponential growth of state-action pairs [20]. In DQL, the action space is typically assumed to be discrete, since the Q-network outputs a separate value for each possible action given a state. For environments with continuous action spaces, alternative approaches such as

Deep Deterministic Policy Gradient (DDPG) [27] or other actor-critic methods are used [28]. A comparison between traditional Q-learning and Deep Q-Learning is illustrated in Figure 2.4.

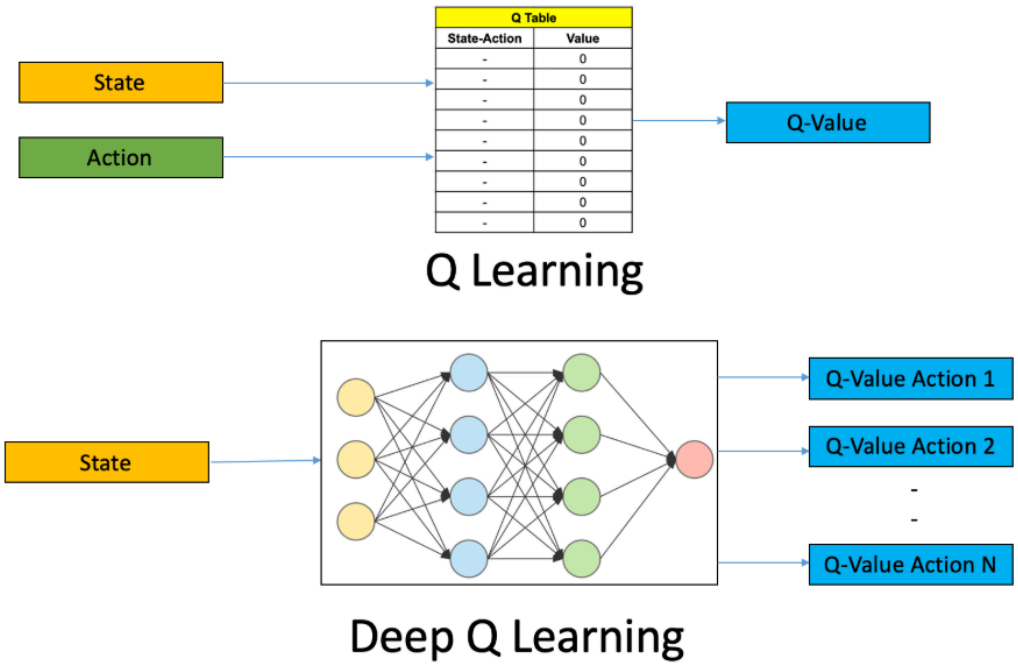


Fig. 2.4. Comparison of Q-learning and Deep Q-learning. [29].

DQL overcomes this limitation by employing a deep neural network, often referred to as a *Deep Q-Network (DQN)*, to approximate the Q-function:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

where θ denotes the parameters of the neural network.

The DQN algorithm uses a neural network to predict Q-values for all possible actions given a state. The network is trained to reduce a loss that encourages the predicted Q-values to align with target values derived from the Bellman equation.

The Bellman equation expresses the value of a state-action pair as the sum of the immediate reward and the discounted value of the next state, capturing the principle of optimality in reinforcement learning:

$$Q(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s', a')\right]$$

where r is the reward received after taking action a in state s, s' is the resulting next state, and γ is the discount factor that balances immediate and future rewards.

A commonly used target value in DQN is:

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

where θ^- represents the parameters of a *target network*, a periodically updated copy of the main Q-network. Using a separate target network helps stabilize training by reducing correlations between target and predicted Q-values [2].

2.3.1. Experience Replay

To further improve training stability and efficiency, DQL employs an *experience replay* mechanism. Instead of updating the network with consecutive samples, the agent stores its experiences (s, a, r, s') in a replay buffer. Mini-batches of experiences are then randomly sampled from this buffer to update the network. This process breaks the correlation between sequential samples and leads to better convergence properties [30].

2.3.2. Enhancements and Variants

Since its introduction, several improvements have been proposed to enhance DQL's performance and stability, including:

- **Double DQN**: Addresses the overestimation bias in Q-learning by decoupling the action selection and evaluation steps in the target computation [31].
- **Dueling Networks**: Separates the estimation of state-value and advantage functions within the network architecture, improving learning efficiency [2].
- **Prioritized Experience Replay**: Samples more informative transitions more frequently based on their TD-error magnitude [32].

These extensions build upon the core DQL framework and have contributed to its success in a variety of challenging tasks.

2.4. Contribution of the project

This project contributes to the field of deep reinforcement learning through a focused experimental study on the Atari Breakout environment. While the Deep Q-Network architecture and its variants have been widely applied to Atari benchmarks, this work provides novel insights by systematically exploring the effects of experience replay configurations, recurrent network components, hidden layer sizes, and subsampling strategies. The main contributions are summarized as follows:

- **Empirical Analysis of Experience Replay Memory Size:** Unlike most prior work that employs a fixed replay buffer size (typically 1 million transitions), this thesis investigates the effects of varying the replay memory size on learning stability and sample efficiency.

- **Integration and Evaluation of GRU Layers:** Extending the traditional DQN architecture, this work incorporates Gated Recurrent Units to enable temporal representation learning. While recurrent architectures such as DRQN exist, this thesis provides a more comprehensive exploration by testing multiple GRU hidden sizes and analyzing their effect on both learning dynamics and final performance in a partially observable setting, where the agent does not have full access to the true environment state at each time step and must rely on past observations to infer hidden information [33].

- **Investigation of Hidden Layer Sizes:** The thesis examines how different hidden layer sizes affect the agent's capacity to learn efficient policies. This study offers insights into the balance between underfitting and overfitting in relatively simple environments like Breakout and contributes guidelines for selecting network sizes in resource-constrained settings.

- **Evaluation of Subsampling Strategies in Experience Replay:** A novel aspect of this work is the exploration of alternative subsampling strategies for selecting experiences from the replay buffer.

- **Evaluation of the generalization of the best agent's configuration:** Different game environments are used once the best configuration is obtained to test the generalization of the training process and agent characteristics.

- **Reproducible Framework and Ablation Study:** All experiments were conducted under a controlled and reproducible training pipeline. Through ablation studies, the individual contribution of each architectural and training choice was isolated, enabling clearer interpretation of experimental outcomes and providing a solid foundation for future research.

2.4.1. Differentiation from Existing Work

In contrast to prior research that often emphasizes large-scale benchmarking across multiple games or the development of novel architectural components, this thesis provides a fine-grained empirical investigation of key design choices within a single environment. Notable distinctions include:

- A detailed study of the influence of replay memory size, which is often fixed or underreported in the literature.

- A modular investigation of GRU-based temporal modeling with varied hidden sizes, going beyond the binary inclusion of recurrence.

- A focus on simple yet effective subsampling strategies that offer practical benefits without added architectural complexity.

- A reproducible and interpretable experimental pipeline designed to isolate and assess the impact of individual design decisions.

These contributions not only enhance our understanding of the design space for DRL agents in the Atari domain but also provide practical insights applicable to other environments and future DRL applications.

3. METHODOLOGY

This chapter presents the fundamental concepts and techniques that form the basis for understanding the work carried out in this study. It includes in-depth explanations of the approaches used in designing, training, and evaluating the agents. A solid grasp of these technical elements is essential for following the analyses and discussions developed in the subsequent chapters. Figure 3.1 illustrates the general workflow of the project.

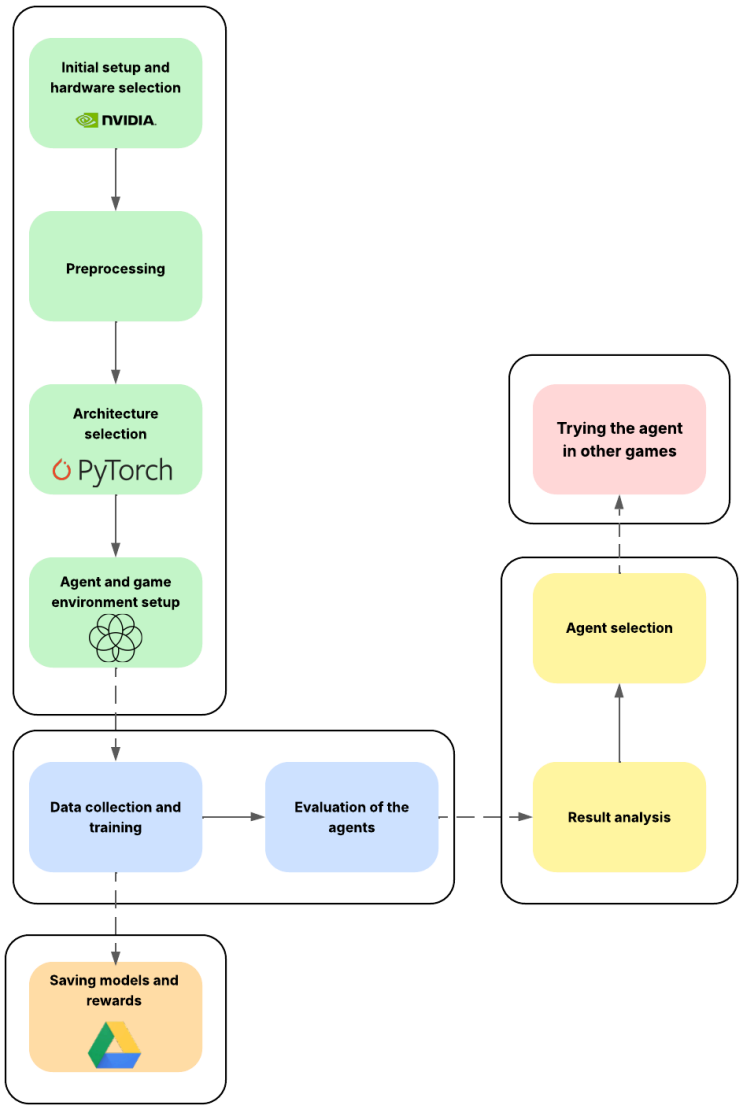


Fig. 3.1. Workflow of the project.

3.1. Development tools and software

This project has been developed using Python[1], which is an interpreted, multi-paradigm, and dynamic programming language. This means that its code is executed directly without prior compilation. This characteristic facilitates rapid iteration, making Python particularly beneficial for data science, which is why it was chosen for the development of this project. With its dynamic typing and automatic memory management, Python removes the need to explicitly define variable types or manually handle memory allocation, streamlining the coding process and reducing potential errors.

Another strength of Python is its extensive number of libraries that significantly enhance its capabilities. Notable libraries include NumPy, Pandas, Matplotlib, PyTorch, Atari Gymnasium, or ALE-py. These libraries facilitate data manipulation, the development, training and evaluation of the deep learning models and the creation of insightful data visualizations. Each library contributes unique features, making Python an indispensable tool for data scientists.

To manage data in Python, the Pandas library was utilized. Pandas[2] is a free, open-source software library built on top of NumPy[3], making it highly effective for data manipulation and analysis. It offers fast, flexible, and powerful data structures designed to handle labeled and ordered data efficiently. The key data structure employed in this project is the Pandas DataFrame, a versatile two-dimensional structure that can handle columns of varying data types, facilitating complex data handling and analysis tasks.

As the development environment, Visual Studio Code[4] was used. It is a free, open-source code editor developed by Microsoft. It features syntax highlighting, version control integration, and a built-in terminal, making it ideal for running scripts locally, especially those requiring extended execution times. These scripts were the ones that made the training of the models.

Jupyter Notebooks[5] were also used because of several key features:

- Interactive coding: The notebook structure allows for cells to be executed individually, with results displayed immediately, facilitating a debug coding style. This is enabled by the Jupyter Kernel, which executes code in each cell and returns the results, enabling dynamic and iterative analysis.
- Visualizations: Jupyter Notebooks provide access to data visualizations and other graphical outputs using libraries like Matplotlib. These visualizations are displayed inline, allowing for seamless integration of code and graphical representations, which helps in understanding data patterns and insights more clearly.

[1] https://www.python.org/
[2] https://pandas.pydata.org
[3] https://numpy.org
[4] https://code.visualstudio.com
[5] https://docs.jupyter.org/en/latest/start/index.html

- Markdown support: Jupyter Notebooks support Markdown, enabling users to combine code, computation outputs, graphical visualizations, and explanatory text within a single document. This allows for live code execution, documentation, and result presentation all in one place.

Due to the project's strong focus on deep learning, the primary library used was PyTorch[6]. It was employed to implement the model architectures, define their layers and forward methods, as well as to train and evaluate the models.

Lastly, the PyDrive2[7] library was used to automatically save the models, experience replay memories, and reward logs to Google Drive. To enable access through this library, a Google Cloud project was configured, and the necessary credentials were managed to authorize and authenticate access to the associated Drive account. A free account of Google Drive has a maximum capacity of 15GB, which was enough to store all models, replay memories and rewards.

3.1.1. Visualization tools

For the creation of graphs, the library Matlplotlib[8] and Seaborn[9] have been used. To effectively visualize data and achieve a better understanding of the training process and evaluation results, various types of plots were utilized. The primary graphs used in the project were line graphs. They displays a sequence of data points that are joined with straight lines. In our case we will use them to visualize the training reward and evaluation score during steps and the decay of epsilon parameter.

The Imageio library[10] was also used to record videos of the game environment during evaluation, allowing for better insight into the situations in which the agent performs poorly.

3.2. Game environment

To train a Deep Q-Network (DQN) agent to play Atari games such as *Breakout*, *Pong* and *Space Invaders*, a simulated environment that mimics the dynamics of the game is required. This is made possible by combining two core libraries:

- Gymnasium[11] provides a high-level API to create and interact with the environment.

[6] https://pytorch.org/
[7] https://pypi.org/project/PyDrive2/
[8] https://matplotlib.org
[9] https://seaborn.pydata.org
[10] https://imageio.readthedocs.io/en/stable/
[11] https://gymnasium.farama.org/introduction/basic_usage/

- ALE-py[12] (Arcade Learning Environment Python) acts as the emulator backend, exposing lower-level game mechanics and configuration options.

Together, these libraries create a robust and reproducible framework for developing and evaluating reinforcement learning agents.

To create and initialize the game environment, a specific game version and render mode must be specified. There are three available render modes:

- **Human:** Displays the game screen and enables sound. Emulation is locked to the FPS defined by the ROM. This mode is primarily intended for human interaction and for understanding the gameplay.
- **Rgb_array:** Returns the current RGB frame of the environment. This mode is commonly used for evaluating agents and recording gameplay videos.
- **None:** Disables rendering entirely, making it ideal for training purposes where visual output is unnecessary.

Depending on the game, different versions may be available; however, one common feature across all versions is the configuration of stochasticity. Since Atari games are fully deterministic by design, agents can achieve state-of-the-art performance by merely memorizing an optimal sequence of actions, without relying on observations from the environment. To prevent such behavior and encourage generalization, several techniques are employed to introduce stochasticity:

- **Sticky Actions:** Instead of always executing the action provided by the agent, there is a small probability that the previously executed action is repeated. In the `v0` and `v5` environments, this probability is 25%, while in `v4` environments it is 0%. This mechanism adds uncertainty to action execution, discouraging hard-coded action sequences.
- **Frame-Skipping:** On each environment step, the same action can be repeated for multiple frames. This can be configured using the `frameskip` argument. If `frameskip` is set to an integer, the same action is repeated for that fixed number of frames, resulting in deterministic behavior. If it is set to a tuple of two integers, a value is uniformly sampled between the first (inclusive) and the second (exclusive) integer at each step, introducing randomness in the number of skipped frames.

3.2.1. Breakout game characteristics

For the development of this project, the training of the agents has been done using the Breakout game environment, more precisely the Atari version of this game. This arcade game was realized by Atari in 1976 and designed by Nolan Bushnell and Steve Bristow.

[12] https://ale.farama.org/

Breakout was developed as an electronic evolution of the popular game Pong, but with a unique twist. While Pong simulated a table tennis match requiring two players, Breakout introduced a single-player experience centered around the destruction of a wall of bricks. The objective of Breakout is straightforward: destroy all the bricks positioned at the top of the screen by bouncing a ball against them. The player controls a paddle located at the bottom of the screen, which can move horizontally to keep the ball in play and strategically aim it toward the bricks. A frame of the game is shown in Figure 3.2.

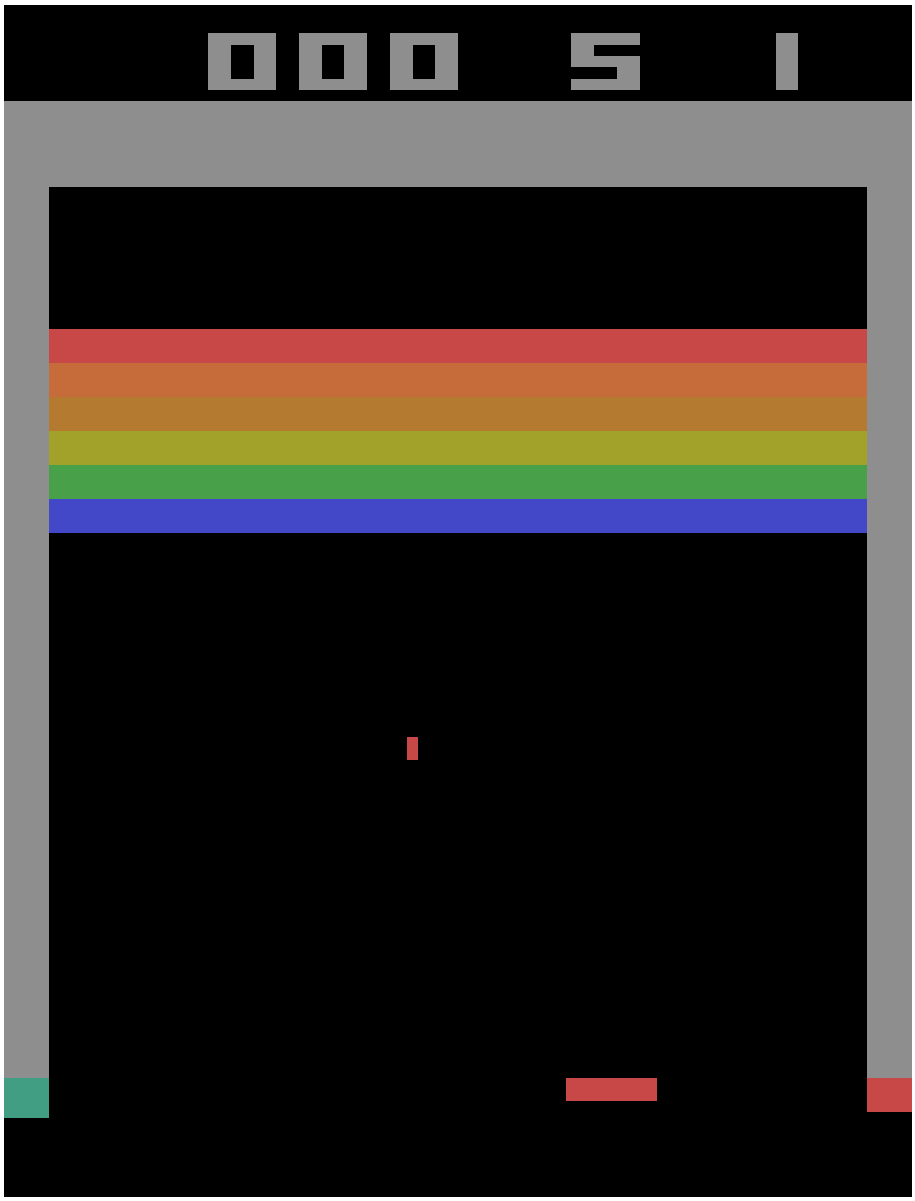


Fig. 3.2. A frame of the initial phase of the Atari's Breakout.

In addition, the ball moves continuously across the screen, bouncing off the side and top walls. If the ball falls below the paddle, the player loses a life. Furthermore, when the ball hits a brick, the brick is destroyed and the ball changes direction and increases its speed. These bricks are arranged in rows and columns at the top of the screen, and each brick destroyed awards points to the player.

The player starts with a set number of lives, usually five, and loses one life each time the ball is missed. If I life is lost the velocity of the ball decreases. When all the bricks on the screen are destroyed, the player advances to the next level, which is just the same layout of bricks but with the speed of the ball increased.

The optimal strategy in Breakout involves creating openings along the sides of the brick wall, allowing the ball to reach the upper area. Once there, the ball can bounce repeatedly above the bricks, destroying them without requiring the player to move the paddle. Figure 3.3 shows the result of this strategy.

Fig. 3.3. A frame of the advanced phase of the Atari's Breakout.

Regarding the scoring system, each level in Breakout features six rows of bricks, with 18 bricks per row, totaling 108 bricks. The point value of each brick depends on its color:

- Red: 7 points.
- Orange: 7 points.

- Brown: 4 points.
- Yellow: 4 points.
- Green: 1 point.
- Blue: 1 point.

If the player destroys all the bricks in a level, the maximum possible score is 432 points.

Breakout has an action space of four discrete actions. Table 3.1 lists the meaning of each action. Note that action 1 is only used to start the game and it will be forced to be the first action.

Action	Meaning
0	Not moving
1	Fire the ball
2	Moving right
3	Moving left

TABLE 3.1. DESCRIPTION OF THE ACTIONS OF THE BREAKOUT GAME.

To promote more stable training, the rewards provided by the environment are clipped. When the ball destroys a brick, an immediate reward of 1 is returned. If the player loses a life, a reward of –1 is given. In all other situations, such as when the ball is simply moving or being hit by the paddle, a reward of 0 is returned.

The available versions of the game provided by Atari Gymasium and ALE-py are described in Table 3.2.

Version	Frameskip	Repeat action probability
Breakout-v0	(2, 5)	0.25
BreakoutNoFrameskip-v0	1	0.25
Breakout-v4	(2, 5)	0.00
BreakoutNoFrameskip-v4	1	0.00
Breakout-v5	4	0.25

TABLE 3.2. STOCHASTIC CHARACTERISTICS OF THE DIFFERENT VERSIONS OF THE BREAKOUT GAME.

3.2.2. Other games

The chosen games for the generalization of the agent are Pong and Space Invaders.

In the game of Pong, the player controls the right paddle and competes against the computer-controlled left paddle. Both players attempt to deflect the ball away from their own goal and direct it toward their opponent's goal. A frame of the Pong game is shown in Figure 3.4.

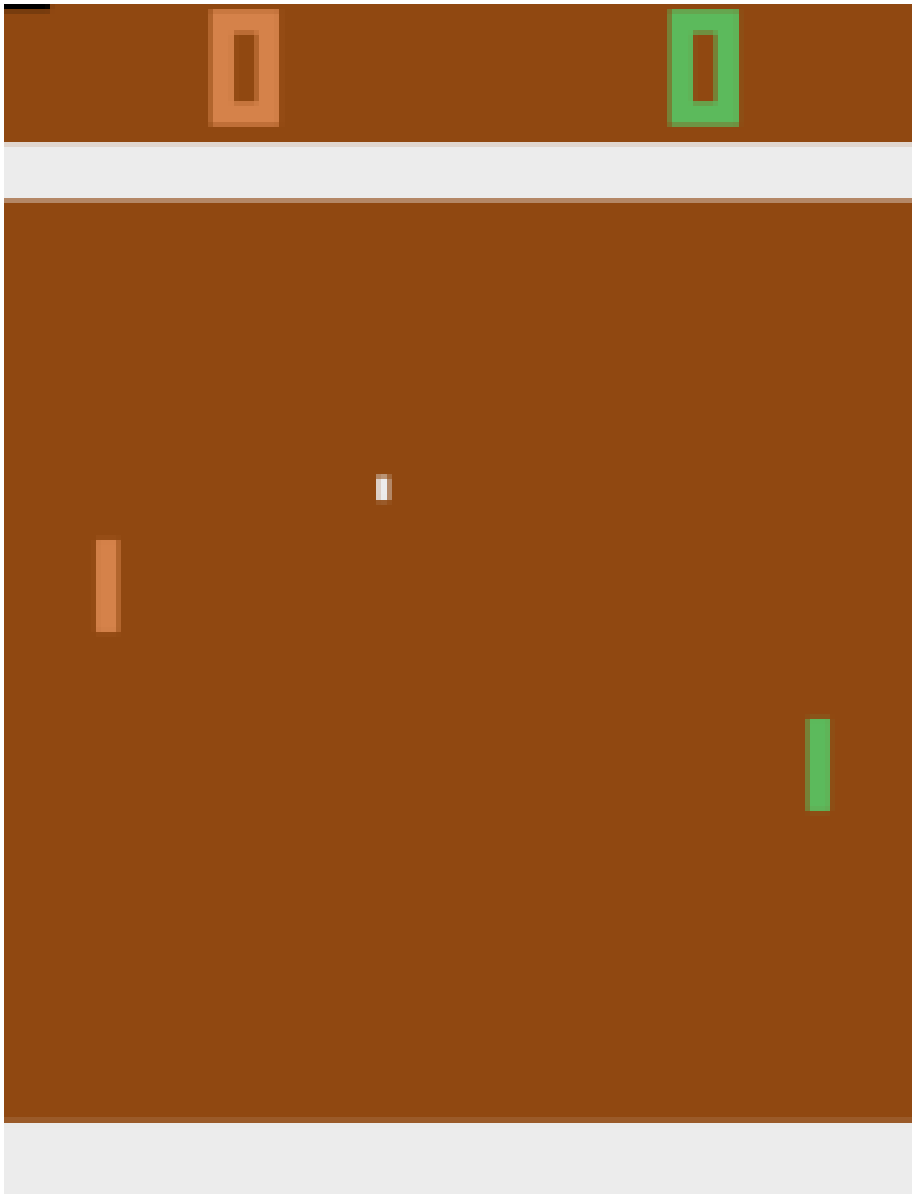


Fig. 3.4. Frame from the Atari Pong game.

The possible actions that the agent can perform in the game are outlined in Table 3.3.

Action	Meaning
0	No movement
1	Fire the ball
2	Move right
3	Move left
4	Fire right
5	Fire left

TABLE 3.3. DESCRIPTION OF THE AVAILABLE ACTIONS IN THE PONG GAME.

The agent receives a positive reward when it scores a goal and a negative reward when the opponent scores.

In the Space Invaders game, the objective is to eliminate the alien invaders by firing a laser cannon before they reach the Earth. The game ends either when the player loses all available lives due to enemy fire or when the invaders successfully reach the Earth. A frame of Space Invaders game is shown in Figure 3.5.

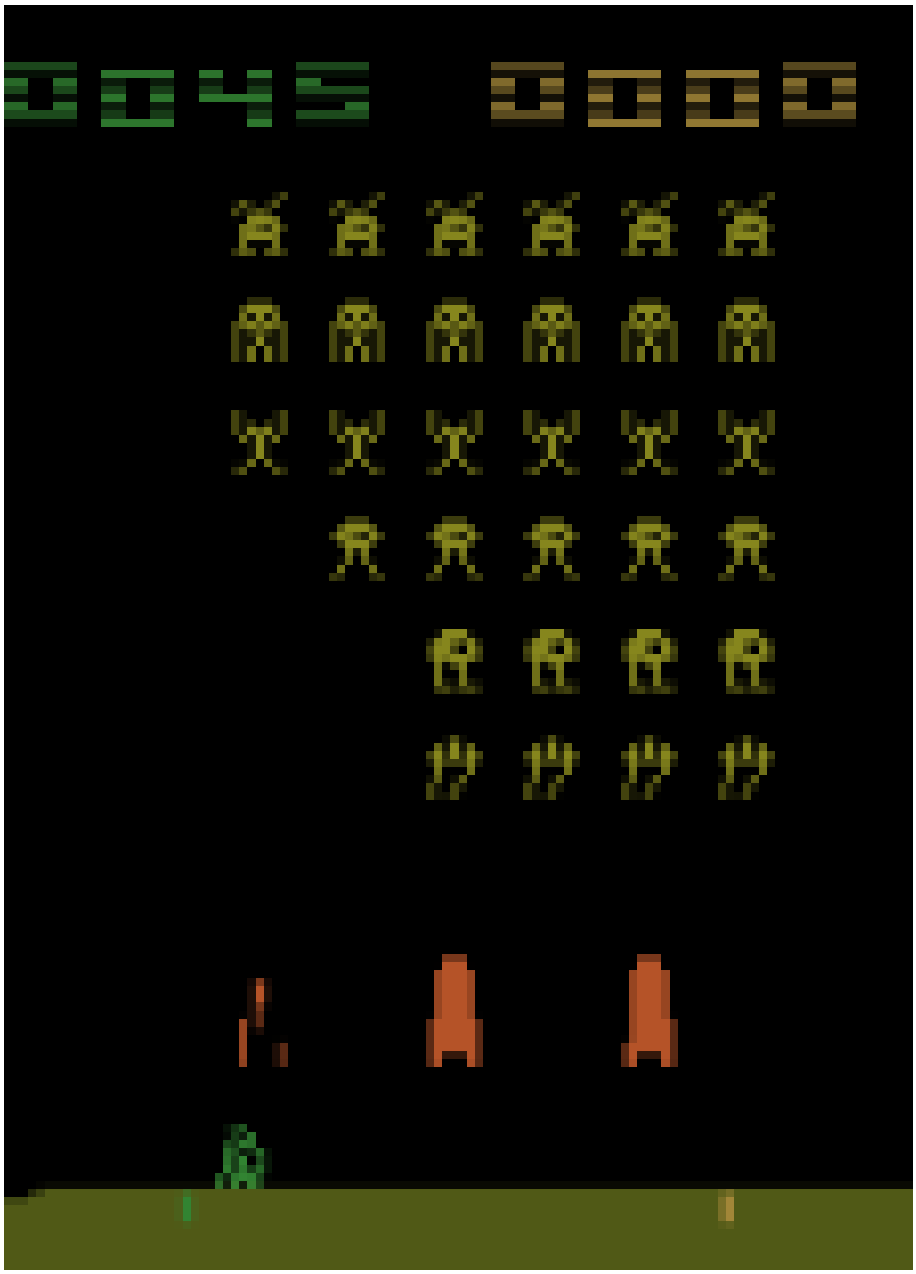


Fig. 3.5. Frame from the Atari Space Invaders game.

The possible actions that the agent can perform in the game are outlined in Table 3.4.

Action	Meaning
0	No movement
1	Fire
2	Move right
3	Move left
4	Fire right
5	Fire left

TABLE 3.4. DESCRIPTION OF THE AVAILABLE ACTIONS IN THE SPACE INVADERS GAME.

The agent receives a positive reward for destroying a space invader, with invaders in the upper rows yielding higher points. A negative reward is given when the agent loses a life or when the invaders reach the Earth.

3.3. Hardware

In this project, Deep Neural Networks are trained to perform complex tasks, requiring significant computational resources. Training such models on CPUs can lead to extremely long and inefficient training times due to their limited parallel processing capabilities. In contrast, GPUs (Graphics Processing Units) are specifically designed to handle large-scale numerical computations in parallel, making them highly effective for accelerating matrix operations and backpropagation.

The use of GPUs significantly reduces training time, enables the handling of larger models and datasets, and facilitates faster experimentation and tuning. Therefore, leveraging GPU acceleration is not just beneficial but essential for achieving practical and efficient deep learning workflows in this project.

The GPU used in this project was an NVIDIA GeForce RTX 3060[13]. The main technical specifications are presented in Table 3.5.

[13] https://www.nvidia.com/es-es/geforce/graphics-cards/30-series/rtx-3060-3060ti/

Specification	Value
CUDA Cores	3584
Memory Size	12 GB
Memory Bandwidth	360 GB/s
CUDA Compute Capability	8.6
FP16 Performance	Up to 16.2 TFLOPS
FP32 Performance	Up to 12.7 TFLOPS
Maximum Power Consumption (TDP)	170 W

TABLE 3.5. KEY SPECIFICATIONS OF THE NVIDIA GEFORCE RTX 3060 FOR NEURAL NETWORK TRAINING.

The *Departamento de Teoría de la Señal y Comunicaciones* provided remote access to this GPU. Since this access was automatically reset every 24 hours, the trained models and reward logs were saved to Google Drive before each reset to prevent data loss.

3.4. Evaluation metrics

To assess how well the different agents perform in playing *Breakout*, several evaluation metrics were selected:

- **Training reward:** The reward obtained by each model during training is recorded to evaluate whether the agents are effectively learning over time.
- **Evaluation reward:** The total reward obtained by each model during the evaluation process, measured after the agent loses all five lives.
- **Maximum frame number:** The highest frame number reached by the agent during evaluation. This metric reflects how far the agent was able to progress in the game.
- **Frames per life:** The maximum number of frames the agent survived during a single life in the evaluation process. This metric helps to assess how effectively the agent plays during each individual life.

In addition to the previously mentioned evaluation metrics, convergence is also analyzed to assess whether the agents are able to learn and maintain a stable policy over time. Convergence is an important indicator of the reliability and robustness of the learning process, as it reflects the agent's ability to consistently make optimal or near-optimal decisions after sufficient training. Specifically, we examine the evolution of performance over the course of training to identify whether the agent's behavior stabilizes, fluctuates, or deteriorates. Furthermore, the number of training steps required to reach a stable policy is taken into account, as it provides insight into the sample efficiency of each model. A model that converges more quickly may be more desirable in practical applications, especially when computational resources or training time are limited.

3.5. Benchmark model

In the previous project developed by Jorge Victoria Gijón [34], a baseline agent was implemented. This model serves as a benchmark for the current work, with the experiments conducted aiming to improve upon its performance.

The main characteristics of his model and project are as follows:

- **Architecture:** A CNN with a fully connected final layers architecture was used to design the Q-network. This architecture is illustrated in Figure 4.3.
- **Experience Replay Memory:** The memory buffer had a capacity of 100,000 transitions.
- **Input Frames:** To enable the model to learn temporal features of the game, the network input consisted of a stack of consecutive frames. The study explored configurations of 3, 4, and 6 stacked frames, concluding that the best performance was achieved with 4 stacked frames.
- **Epsilon Decay Strategy:** Balancing exploration and exploitation was crucial. The best results were obtained by linearly decaying the epsilon parameter from 1 to 0.1 over the first 1 million steps.
- **Prioritized Subsampling:** An attempt was made to prioritize sampling transitions from later phases of the game within the experience replay memory. However, this strategy did not yield any improvements in the agent's performance.

The best configuration of his model achieved a maximum score of 384, corresponding to a mean score of 76.5 per life. The longest frame reached by the agent was 7712, after a total of 21,600,000 training steps.

4. DESCRIPTION OF THE DEEP REINFORCEMENT LEARNING SYSTEM

This chapter discusses the implementation of the system used to train agents with the Double Deep Q-Network (DDQN) approach, focusing on the architecture, experience replay memory, sub-sampling techniques, and hyperparameters.

4.1. Workflow of the system

The system begins with configuring the development environment and selecting the hardware device, either CPU or GPU, based on available system resources. Using a GPU significantly improves training efficiency and speed.

Once all libraries are imported and the initial configurations are completed, the preprocessing phase begins. The goal of this step is to reduce computational costs and focus on the most relevant information. During preprocessing, the game frames are converted from RGB to grayscale to simplify the input data. Additionally, the images are cropped to exclude irrelevant areas, such as game metadata, and retain only the gameplay area. This helps eliminate visual noise and allows the agent to focus on the essential elements of the environment. Lastly, the frames are reshaped to fit in the Q-network. Figure 4.1 illustrates a comparison between the original and preprocessed images.

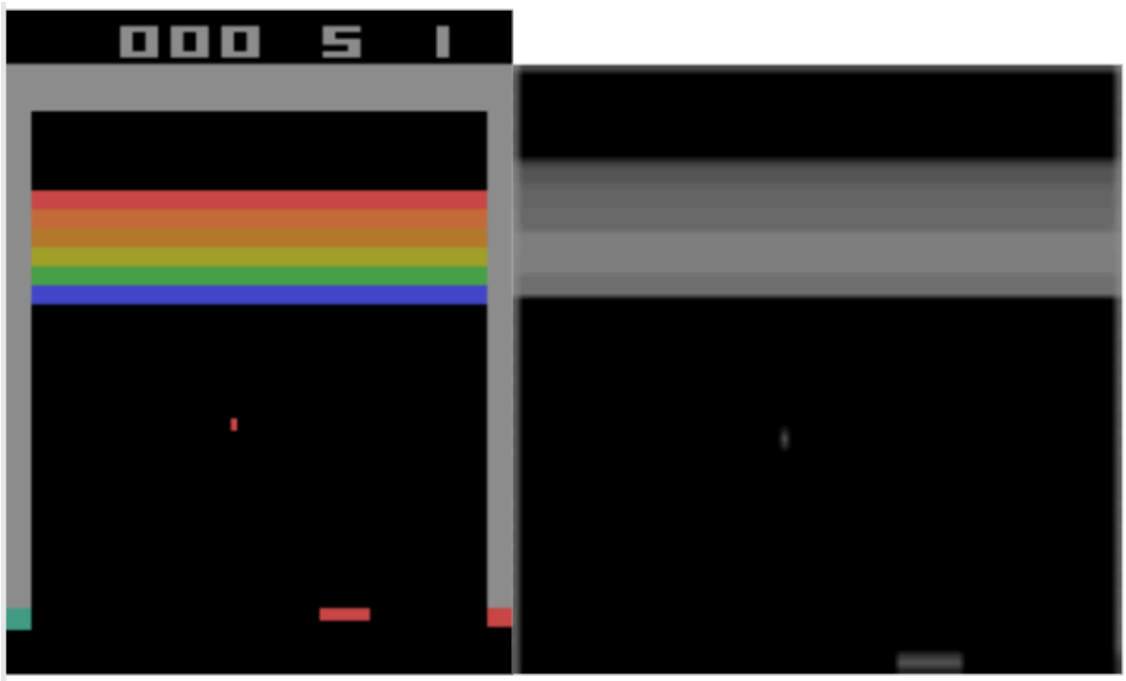


Fig. 4.1. Frames before and after the preprocessing step.

After preprocessing, every four consecutive frames are stacked to capture the temporal dynamics of the game. This is crucial for enabling the agent to learn temporal features, such as the speed and direction of the ball. Additionally, during preprocessing, game rewards are clipped to a fixed range to stabilize training and accelerate the learning process.

Next, the architecture of the model to be used as the Q-network is defined. This involves implementing a deep neural network along with its forward method. During this step, care is taken to properly handle the input stacked frames and accurately compute the Q-values for each possible action.

With the Q-network implemented, the next step is to develop the agent responsible for interacting with the game environment. The agent uses the Q-network to select actions based on the current game state. To maximize rewards, it follows an exploration-exploitation strategy: initially favoring exploration to discover new actions, and gradually shifting toward exploitation as training progresses. This balance is automatically adjusted throughout the training process.

During training, the agent repeatedly plays the game, taking actions based on its current state. In response, the environment provides a new state, a reward and a *done*. These experiences are stored in a replay memory, which is later used to train the model. This data collection process is illustrated in Figure 4.2.

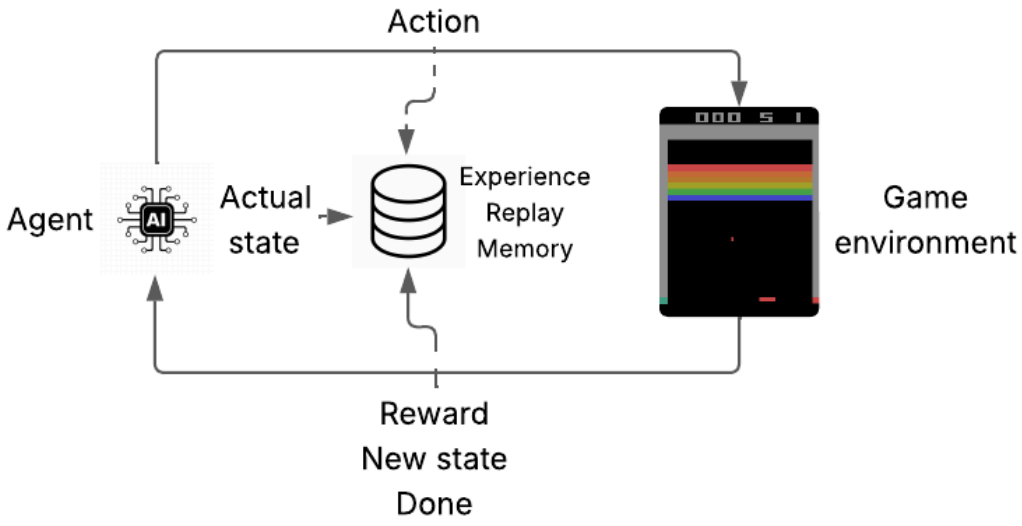


Fig. 4.2. Data collection process.

The experience replay memory is a key component in the training process, as it stores the transitions encountered by the agent during gameplay, each consisting of the current state, action, reward, and next state. During training, a random sample of these transitions is drawn from memory, which helps break the correlation between consecutive experiences. This leads to a more stable and efficient learning process.

During training, the Huber loss function is minimized using backpropagation and the Adam optimizer. Every 100,000 steps, the model and the replay memory are saved and uploaded to Google Drive, along with the corresponding reward. These saved models are later used for evaluation.

4.2. Architecture of the Deep Q-networks

The actions selected by the agent are determined using Deep Q-Networks (DQNs), whose underlying neural network architectures play a crucial role in the agent's performance. In this project, two architectures were implemented: a basic DQN and a Recurrent Neural Network (RNN) built upon the DQN structure. The rationale behind this design choice is to begin with a simpler, feedforward model and then extend it with recurrent layers to capture the temporal dependencies inherent in the gameplay. The basic DQN serves as a baseline for performance comparison, while the RNN aims to enhance the agent's ability to make decisions based on sequences of frames, such as tracking the movement of the ball over time. Figures 4.3 and 4.4 illustrate the architectures of the base DQN and RNN models, respectively.

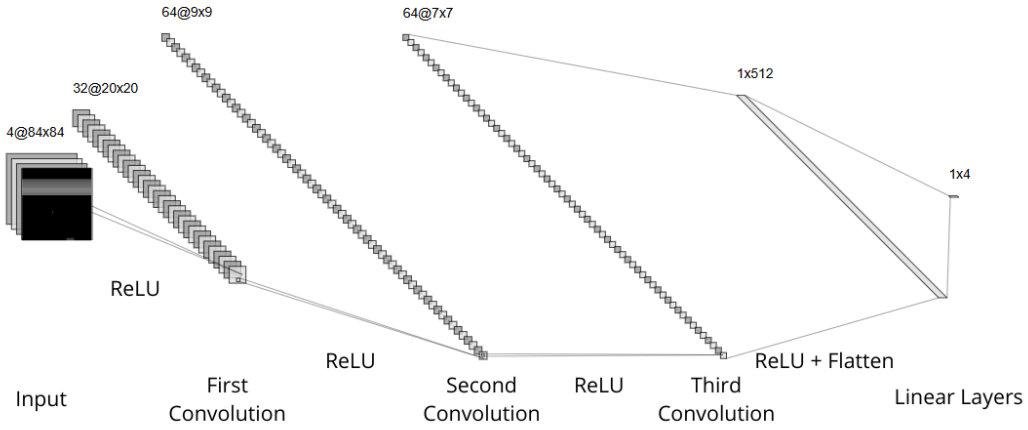


Fig. 4.3. DQN architecture.

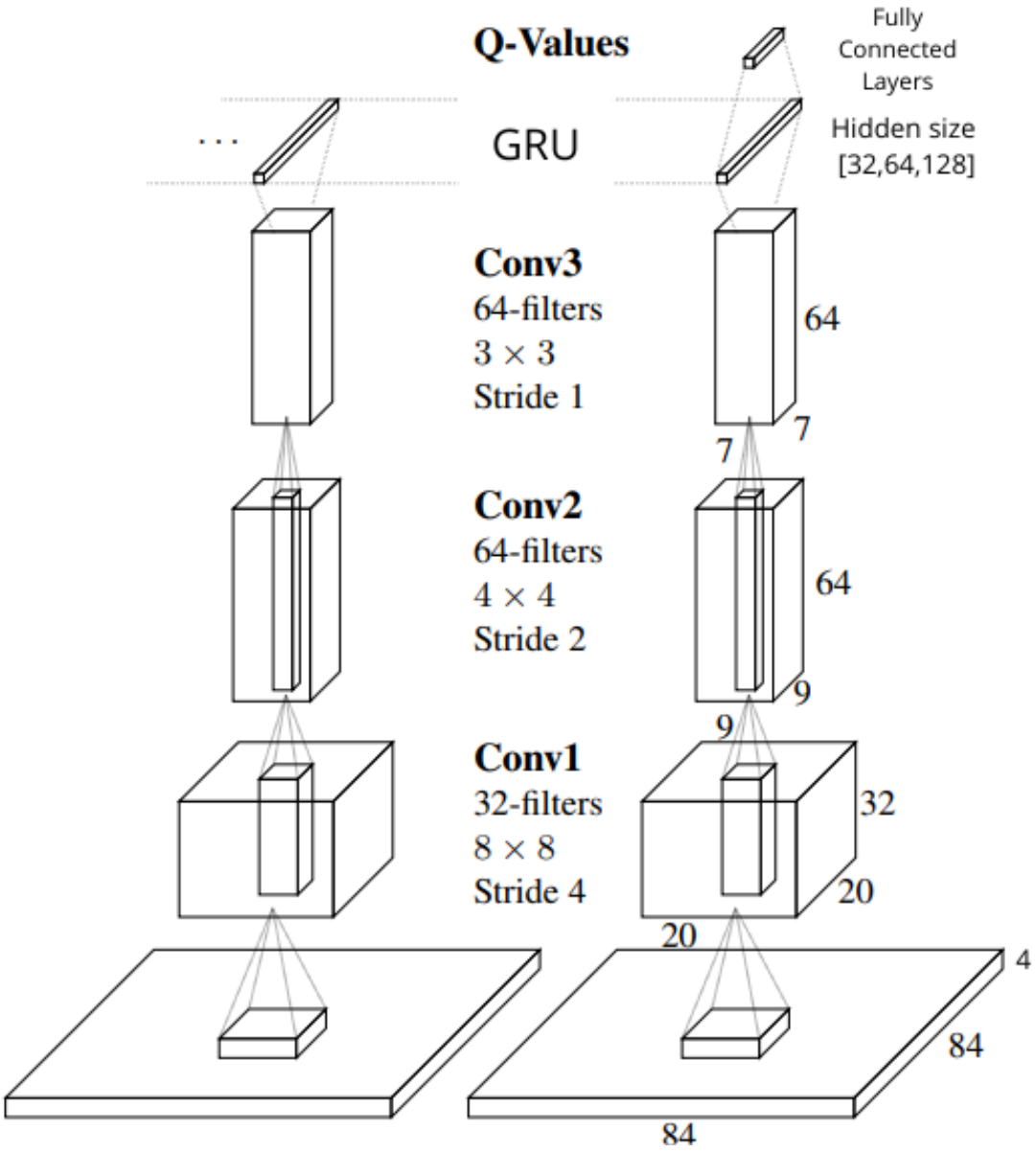


Fig. 4.4. Recurrent DQN architecture with GRU.

The DQN model, Figure 4.3, uses the following layers: three convolutional layers with kernel sizes of 8×8, 4×4, and 3×3, producing output feature maps of sizes 20×20, 9×9, and 7×7, respectively. These are followed by a fully connected layer with 512 units and an output layer predicting Q-values for each action. The recurrent model ,Figure 4.4, shares the same convolutional feature extractor but introduces a fully connected compression layer to reduce the 3136 dimension into 512, a GRU with three possibilities for the hidden state size, 32, 64 or 128, and a final output layer. This design enables better modeling of partial observability and dynamic game states. The activation function used between layers is the ReLU.

One key design decision in the recurrent model is the choice of the Gated Recur-

rent Unit (GRU) over the Long Short-Term Memory (LSTM) architecture. Both GRUs and LSTMs are capable of capturing long-term dependencies and mitigating the vanishing gradient problem common in vanilla RNNs. However, GRUs offer several advantages that align well with the requirements of this project. Firstly, GRUs have a simpler structure with fewer gates, resulting in a reduced number of parameters and consequently lower computational overhead. This efficiency allows for faster training and inference, which is critical when integrating recurrent layers within a Deep Q-Network that already involves convolutional feature extraction and multiple fully connected layers. Secondly, GRUs have demonstrated comparable performance to LSTMs across various sequence modeling tasks, often matching or even outperforming LSTMs in environments where the amount of training data is limited or the temporal dependencies are not extremely long. Given the relatively short time horizons and partially observable nature of the gameplay in this project, the GRU strikes an optimal balance between expressiveness and efficiency, making it a suitable choice for enhancing the agent's ability to model temporal dependencies without incurring excessive complexity.

4.3. Experience replay memory

The purpose of the experience replay memory is to store the agent's interactions with the game environment for use during training. This approach improves training efficiency and stability by reducing the temporal correlation between experiences.

In the implemented system, the experience replay memory stores transitions in the form (s, a, r, s', done), where *done* is a boolean indicating whether the new state is terminal, that is, whether the episode has ended. If the DQN includes a recurrent neural network RNN layer, each transition also stores the hidden state used by the agent to select the action.

The experience replay memory is initialized with a fixed maximum size, meaning that once it is full, the oldest transitions are replaced by the most recent ones. In this project, different memory sizes, 100,000, 250,000, and 400,000 transitions, are used to explore how this parameter impacts the training of the agents.

Once a minimum number of transitions has been collected in the replay memory, a subsampling process is used to extract a batch for training the agent and updating the model parameters.

4.4. Subsampling

Subsampling from the replay memory in reinforcement learning is necessary to break the temporal correlations between consecutive experiences, which can destabilize the training of neural networks. By randomly sampling past transitions, the learning algorithm can approximate the independent and identically distributed assumption required by most

optimization. This process also improves data efficiency by allowing the agent to learn from a more diverse set of past experiences, reducing overfitting to recent events and promoting more stable and generalizable learning.

In this project, three different strategies were used for subsampling from the experience replay memory:

- **Random sampling:** The transition are sampled randomly which is simpler but reduces the computational costs.

- **Sampling Based on Rewards:** A prioritized sampling strategy was employed to increase the likelihood of selecting transitions that are close to non-zero rewards. This approach is based on the assumption that reward-related transitions carry more information than those where the ball is simply in motion without consequence. For instance, transitions in which the ball is near the bottom of the screen represent more critical situations, as failing to return the ball in this region typically results in the loss of a life.

  Given a dataset of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s'_i, d_i, f_i)\}$, where f_i denotes the frame number of the i-th transition, this sampling strategy prioritizes transitions based on their temporal proximity to non-zero rewards. In order to compute it, for each transition i, the distance δ_i to the closest transition j with a non-zero reward is defined as follows:

  $$\delta_i = \min_{\substack{j \\ r_j \neq 0}} |f_i - f_j| \tag{4.1}$$

  To assign a priority score to each transition:

  $$p_i = \frac{1}{\delta_i + \varepsilon} \tag{4.2}$$

  where $\varepsilon > 0$ is a small constant to avoid division by zero and to soften the prioritization. The closer a transition is to a non-zero reward, the higher its priority.

  The sampling probability for each transition is defined as:

  $$P(i) = \frac{p_i}{\sum_k p_k} \tag{4.3}$$

- **Rank-based Prioritized Experience Replay with Smoothing:** In this approach, sampling prioritizes transitions that are expected to have a higher learning impact, typically those with larger Temporal-Difference (TD) errors. First, the TD errors are computed as shown in Equation 4.4. Then, transitions are sorted in descending order based on these TD errors, and the sampling probability is computed using a smoothed rank-based formula (Equation 4.5) to avoid excessively sharp prioritization [32].

$$\delta_i = \left| Q(s_i, a_i) - \left[r_i + \gamma \cdot \max_{a'} Q'(s'_i, a') \right] \right| \tag{4.4}$$

Where:

- s_i, a_i, r_i, s'_i are the **state**, **action**, **reward**, and **next state** for transition i,
- Q is the **current Q-network**,
- Q' is the **target Q-network**,
- γ is the **discount factor**,
- δ_i is the **TD error** for transition i.

To reduce the impact of outliers and ensure smoother sampling distributions, a smoothing parameter $\tau \in (0, 1]$ is introduced. The sampling probability is then defined as:

$$P(i) = \frac{1/\text{rank}(i)^\tau}{\sum_j 1/\text{rank}(j)^\tau} \tag{4.5}$$

Where:

- rank(i) is the position of transition i when TD errors are sorted in descending order (i.e., the highest error has rank 1),
- τ is a **smoothing exponent** that controls how much prioritization is applied. When $\tau = 1$, it recovers the standard rank-based prioritization; lower values make the distribution closer to uniform sampling.

Whenever new transitions are added to the replay memory or existing TD errors are updated after learning steps, the list of ranks is recomputed to reflect the most recent priorities. This dynamic updating ensures that the sampling probabilities remain consistent with the current importance of each transition.

4.5. Training parameters

In the context of deep reinforcement learning, training parameters are values established before the training begins and they significantly affects the performance of the model. Adjusting these parameters is a key to ensure good results. Some of these parameters are learning rate, batch size or epsilon.

4.5.1. Learning rate

The learning rate determines the size of the updates made to the neural network's weights during training. If the learning rate is too low, the model may converge very slowly due to

minimal updates. Conversely, if the learning rate is too high, it can lead to unstable training and prevent the model from converging properly. A comparison of how the different learning rates affect to the loss during the training process is illustrated in Figure 4.5.

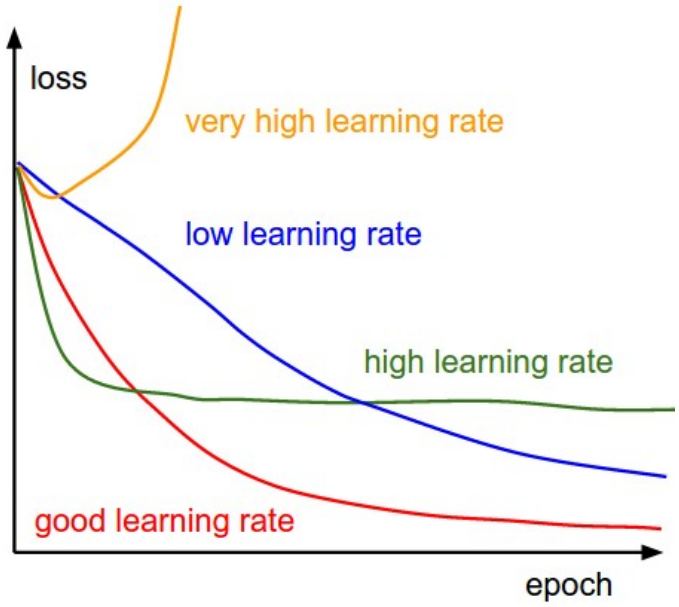


Fig. 4.5. Comparison of different learning rates during the training process [35].

In order to ensure consistency and to compare results with previous work, a learning rate of 10^{-4} is chosen.

4.5.2. Batch size

The batch size refers to the number of transitions sampled from the experience replay memory and used in each training iteration to update the agent. Smaller batch sizes may lead to noisier and less stable training, as the updates are based on limited information. On the other hand, larger batch sizes can improve training stability and lead to more reliable gradient estimates, but they require significantly more memory. Given the hardware constraints in this project, a batch size of 32 was chosen as a balance between stability and memory usage.

4.5.3. Epsilon

This parameter controls the rate at which the value of ϵ decreases in the ϵ-greedy strategy, balancing exploration and exploitation during training. A rapid decay may lead the agent to converge prematurely to a suboptimal policy, while a slower decay generally allows for more thorough exploration but requires longer training times. Based on findings from previous work, the most effective configuration is to start with $\epsilon = 1$ and gradually decrease it to 0.1 over the course of one million steps. This decay schedule is illustrated in

Figure 4.6.

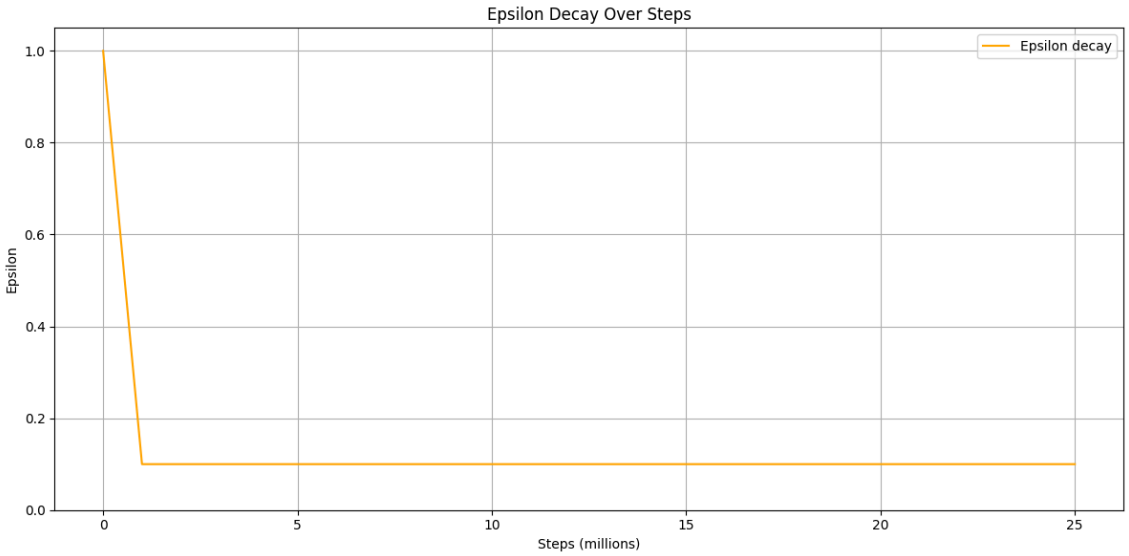


Fig. 4.6. Epsilon decay during training for the first 3 million steps.

4.5.4. Loss function

The loss function chosen for this project is Huber's loss function. This function effectively combines the advantages of both the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). It is particularly well-suited for reinforcement learning tasks because it behaves quadratically for small errors, promoting faster convergence, and linearly for large errors, thereby reducing sensitivity to outliers. These characteristics make Huber's loss a stable and efficient choice for training models. The mathematical definition of Huber's loss is provided in Equation 4.6.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{if } |a| > \delta \end{cases} \tag{4.6}$$

Where:

- $a = y - \hat{y}$ is the residual (true value minus predicted value).
- δ is a threshold parameter that controls the transition between quadratic and linear behavior.

4.5.5. Optimizer

For this project the chosen optimizer was Adam (Adaptive Moment Estimation). It is an optimization algorithm that combines the strengths of two widely used methods: AdaGrad

and RMSProp. It adapts the learning rate for each individual parameter by considering the first and second moments of the gradients, that is, the mean and the variance.

The first moment, the mean, helps guide the updates in the correct direction, while the second moment, the variance, is used to scale the step size, reducing it for parameters with high variability and increasing it for those with more consistent gradients. This adaptive mechanism makes Adam efficient and robust, especially in scenarios with sparse or noisy gradients.

5. EXPERIMENTS AND ANALYSIS OF RESULTS

This chapter presents the experiments conducted and discusses the results obtained.

5.1. Experimental setup

Firstly, to ensure consistency across all experiments, the following considerations were applied:

- Each agent was trained for a total of 25 million steps.
- Rewards were recorded every 100,000 steps to monitor the model's performance and track its progress throughout training. These rewards were stored in a file for further analysis.
- Each 100,000 steps the model is saved in order to evaluate it later.
- All agents were trained using GPU-equipped computers provided by the *Departamento de Teoría de la Señal y Comunicaciones*, allowing for a more efficient training process.

The final selected values for the experimental setup have been summarized in Table 5.1:

Parameter	**Value**
Learning rate	10^{-4}
Discount factor (γ)	0.99
Batch size	32
Target network update frequency	10k steps
Exploration decay	1M steps
Epsilon (ϵ)	[0.1,1]
Network saved frequency	100k steps

TABLE 5.1. TRAINING PARAMETERS USED IN THE EXPERIMENTS.

For the evaluation process, in the trained agents, ϵ was set to 0.025, making the agents play in the exploitation mode. At every 100,000 training steps, the corresponding model was used to play the game for five lives for evaluation purposes. During these evaluation episodes, several metrics were recorded to assess agent performance, as described in Section 3.4.

5.2. Experience replay memory size

The first experiment conducted aimed to analyze whether the size of the experience replay memory affects the training process and the performance of the model. The goal was to determine if having a larger memory, containing a broader range of transitions from different phases of the game, can positively influence the learning process. However, the memory size must remain within reasonable limits, as excessively large memories can exceed the available RAM, making the training process infeasible.

The memory sizes explored in this experiment are 100,000, 250,000, and 400,000 transitions. The architecture selected for this analysis is the base DQN model. The rewards obtained during training for each memory size are shown in Figure 5.1, while the evaluation rewards are presented in Figure 5.2.

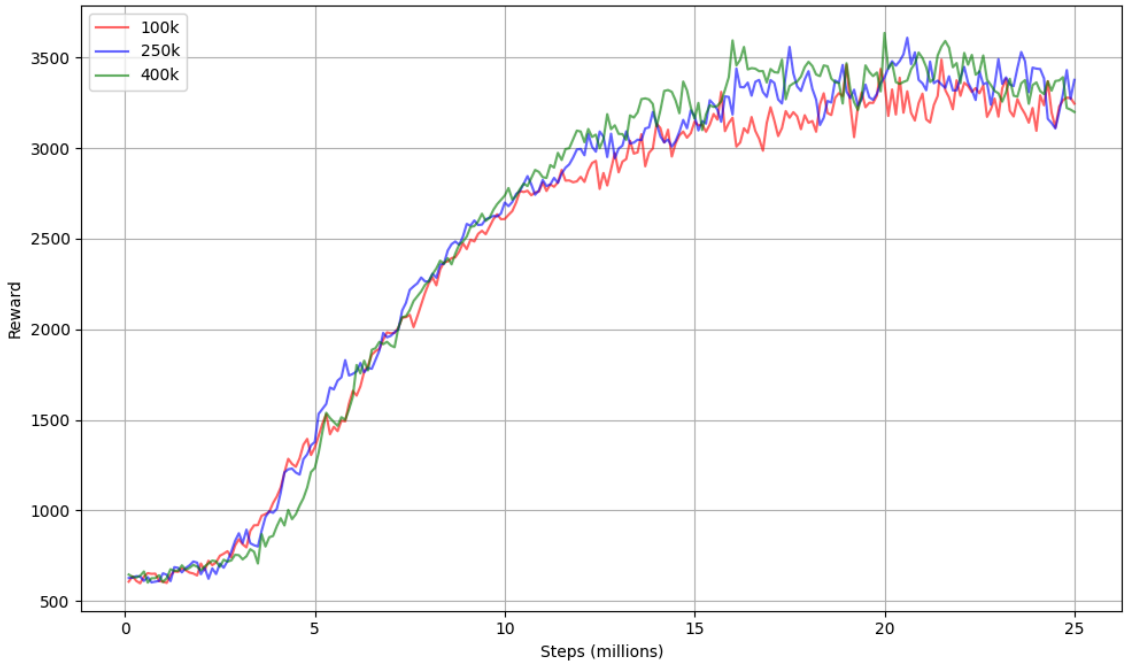


Fig. 5.1. Evolution of the rewards during the training process for the different memory sizes.

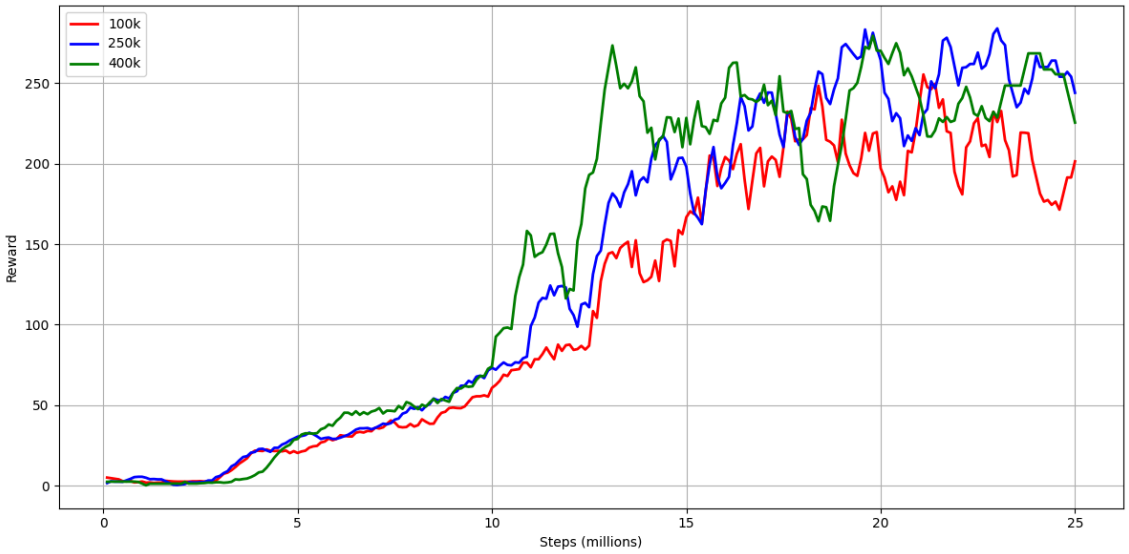


Fig. 5.2. Evolution of the rewards during the evaluation process for the different memory sizes.

The results suggest that a larger replay memory, 250,000 or 400,000, allows the DQN agent to store a broader and more diverse set of transitions, improving the stability and generalization of the Q-value estimates. In contrast, a 100,000 buffer lacks the capacity to capture enough varied experience, leading to poorer learning due to overfitting to recent transitions and insufficient representation of complex gameplay dynamics.

Comparing the larger replay memory sizes of 250,000 and 400,000, we observe that the 400,000 size agent achieves slightly better performance during the early stages of training. This can be attributed to the increased diversity and longer retention of past experiences, which can help stabilize learning in the initial phases. However, this advantage diminishes over time. By around 17 million steps, both agents converge to similar performance levels, with the 250,000 size buffer even showing marginally better results.

Given that the 250,000 buffer achieves comparable or slightly better final performance while using less memory and computational resources, it represents the more efficient and optimal choice for this particular experiment. It offers a good trade-off between experience diversity and resource usage without sacrificing long-term performance.

5.3. Hidden state size

One of the experiments was to extended the architecture of the Q-network with RNN layers, introducing a new parameter that can affect the performance of the model, the size of the hidden state of the RNN layers.

In this experiment, hidden sizes of 32, 64 and 128 were explored, using 250,000 transitions as the memory size. Results in training and evaluation are shown in Figures 5.3 and 5.4.

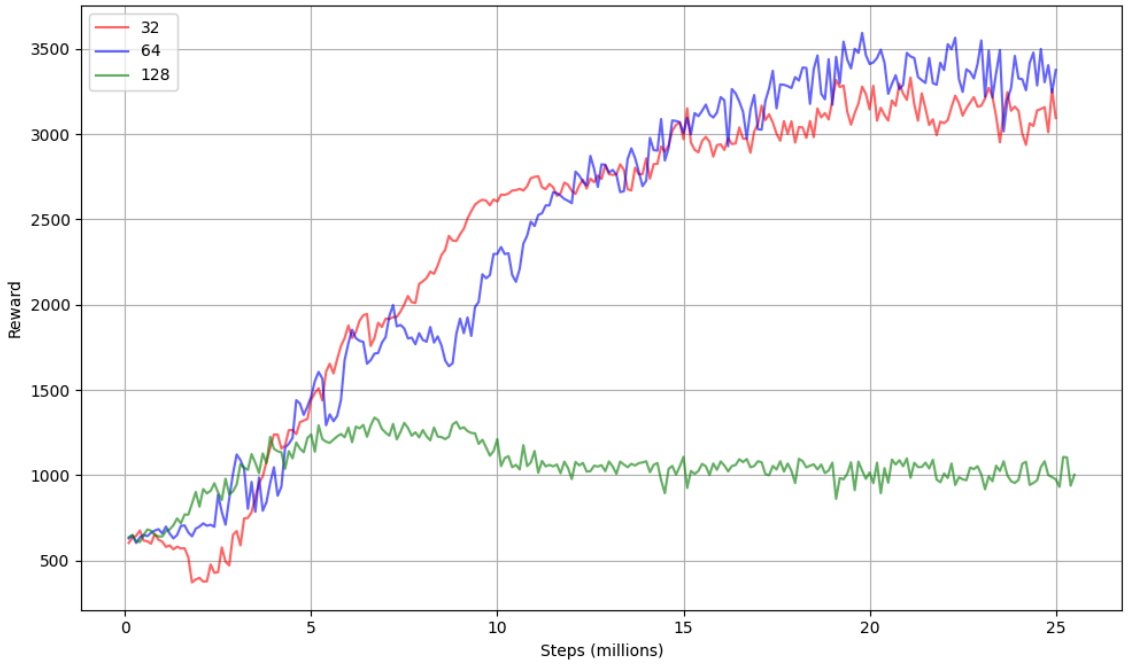


Fig. 5.3. Evolution of the rewards during the training process for the different hidden state sizes.

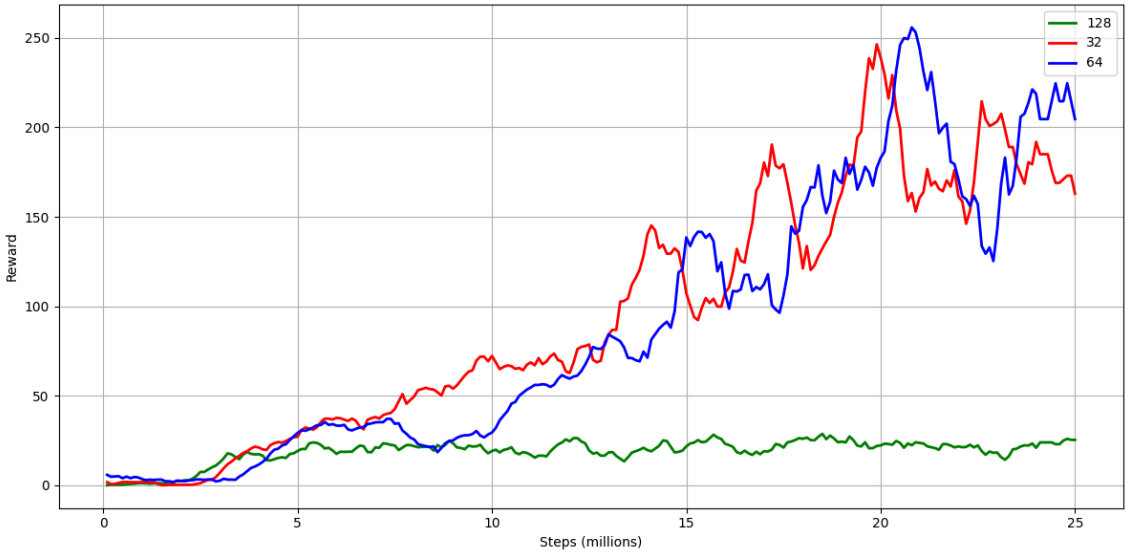


Fig. 5.4. Evolution of the rewards during the evaluation process for the different hidden state sizes.

The results show that the configuration with a hidden state of 64 yielded the best performance, followed by 32, while the 128 size configuration performed significantly worse.

The hidden size of 64 offers the best balance between capacity and stability. A size of 32 may be slightly underpowered to capture useful temporal dependencies in the environment, leading to slightly reduced performance. On the other hand, increasing the hidden size to 128 appears to introduce excessive model complexity, which can lead to unstable training and the introduction of normalization layers to solve this problem.

Additionally, since the input to the network already includes a stack of consecutive frames, some temporal information is directly encoded in the observations. This reduces the need for very large recurrent models, as the GRU is not required to learn all temporal dependencies from scratch. In this context, a moderate hidden size like 64 is sufficient to extract additional sequential patterns without incurring redundancy or instability.

5.4. Changes in architecture of the models

In this project, the architecture of the model was also an important aspect of exploration. The objective of these experiments is to introduce modifications to the baseline architecture and evaluate whether such changes lead to improved performance.

The two architectures implemented are the base DQN and a Recurrent Neural Network (RNN). The RNN incorporates Gated Recurrent Unit (GRU) layers to better capture and interpret the temporal dependencies present in the gameplay.

A comparison of the training and evaluation performance of both architectures is presented in Figures 5.5 and 5.6.

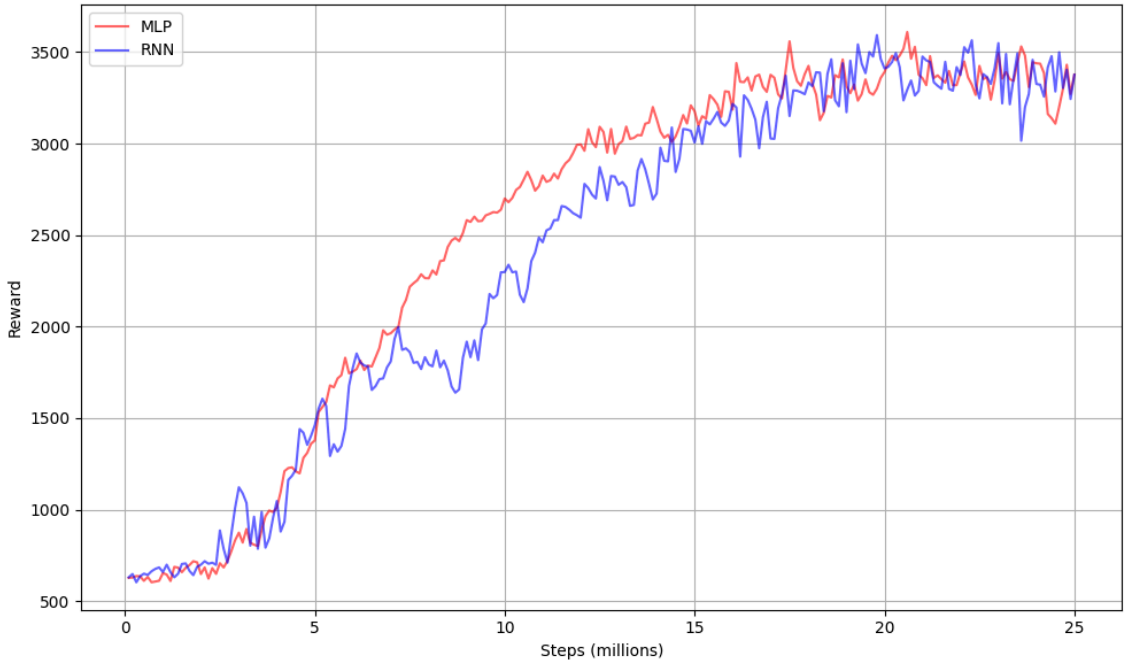


Fig. 5.5. Evolution of the rewards during the training process for the different architectures.

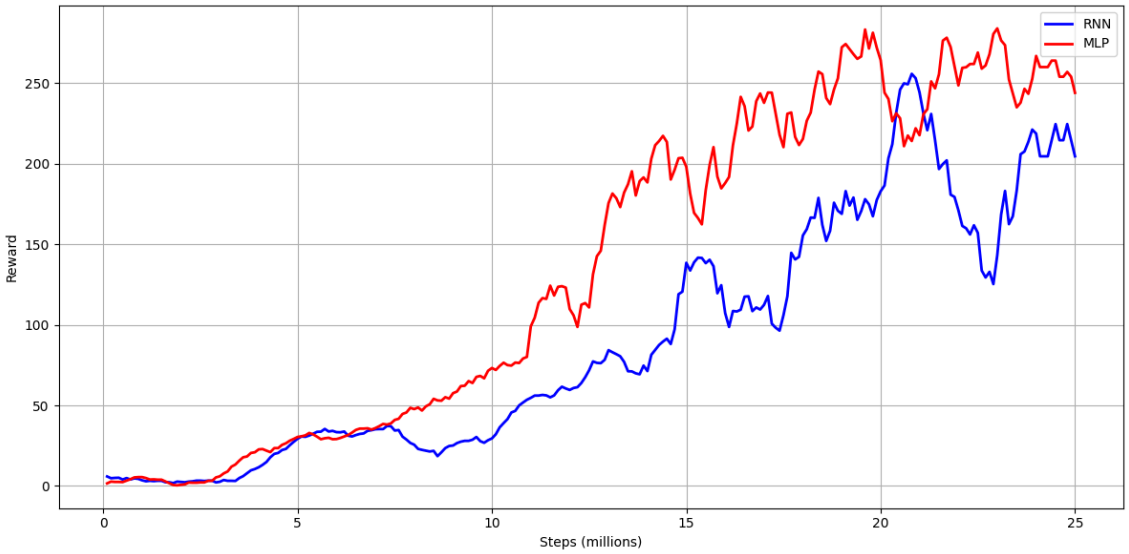


Fig. 5.6. Evolution of the rewards during the evaluation process for the different architectures.

The results showed that the base DQN model outperformed the RNN variant during evaluation and had slightly better result during training. This suggests that in the specific setting of this task, where the agent already receives stacked frames as input, the additional temporal modeling introduced by the GRU layer may be redundant or even detrimental. The base DQN model, being simpler and more stable, is likely better suited to leverage the frame-stacked input without the overhead and potential instability of a recurrent component. Furthermore, the GRU's additional parameters may increase the risk of overfitting or make optimization more difficult, especially given the sparsity and variability of reinforcement learning. These results indicate that, in this case, the added complexity of the GRU does not translate into performance gains, and the simpler DQN architecture remains more effective.

5.5. Subsampling

In this experiment, three different subsampling strategies were tested to enhance the performance of the best-performing agent—DQN with an experience replay memory of 250k. The evolution of rewards during training and evaluation is shown in Figures 5.7 and 5.8.

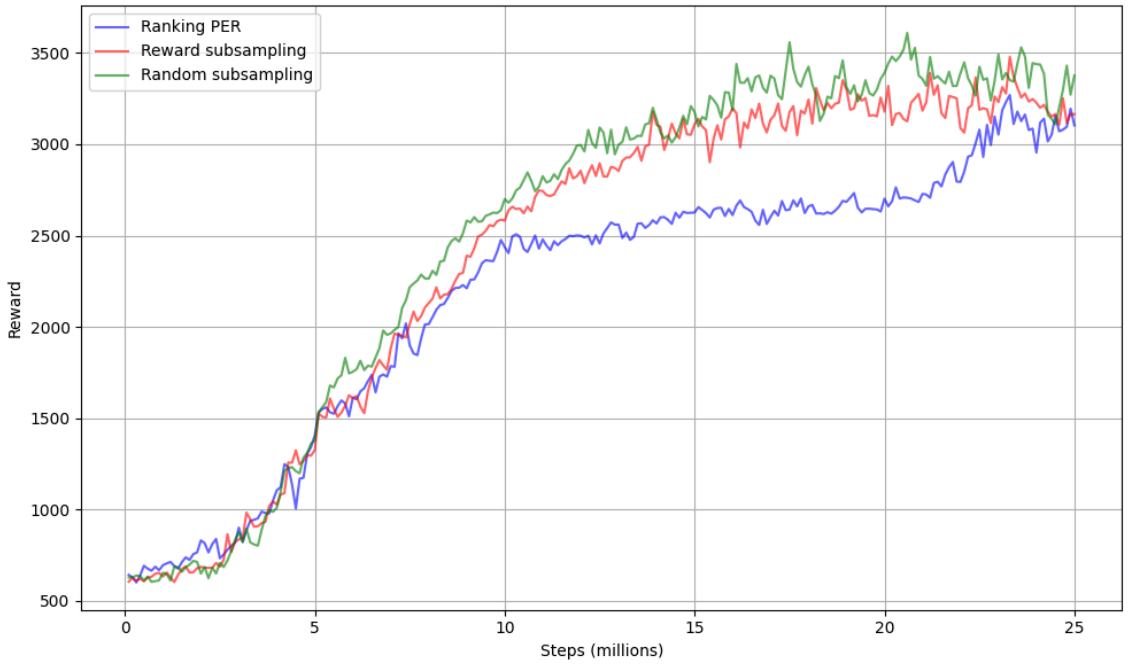


Fig. 5.7. Evolution of the rewards during the training process for the different subsampling strategies.

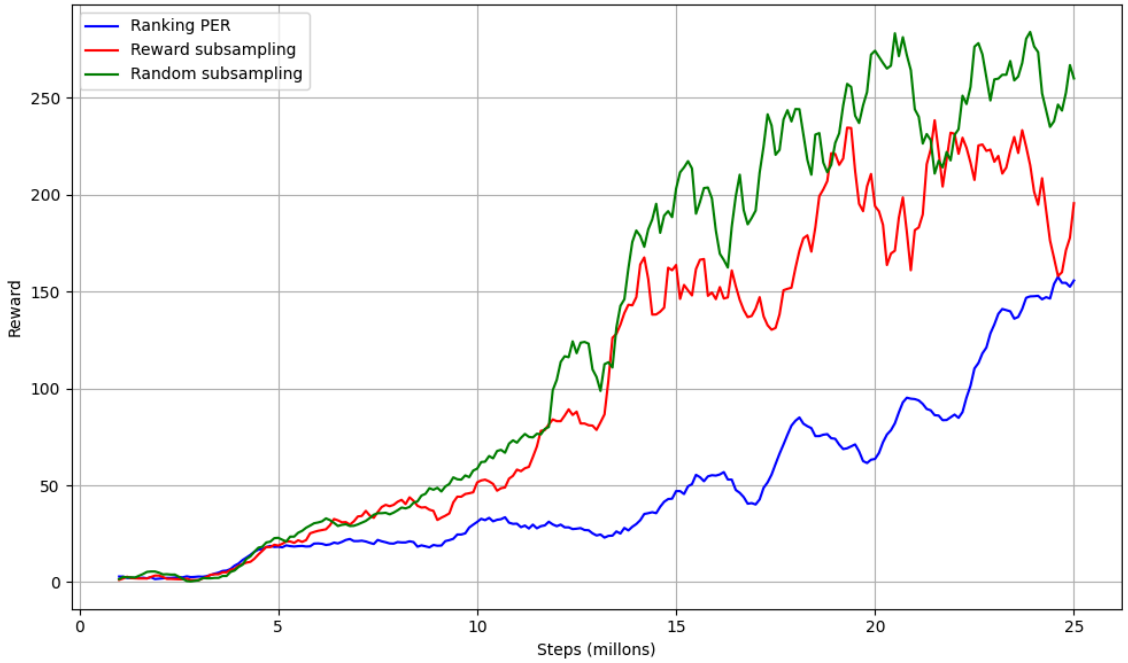


Fig. 5.8. Evolution of the rewards during the evaluation process for the different subsampling strategies.

The best performance was achieved using random subsampling. This result suggests that, despite its simplicity, uniformly sampling from the replay buffer ensures a broad and diverse representation of the environment's state space. It avoids overfitting to rare or specific types of experiences and provides a balanced distribution of both rewarding

and non-rewarding transitions, which is essential for stable Q-value estimation and policy learning.

Subsampling with a focus on transitions with non-zero rewards led to slightly lower performance. While this method intuitively concentrates learning on more informative feedback, it tends to discard a large number of neutral transitions that are crucial for understanding the dynamics of the environment. Since most transitions yield zero reward, excluding them can limit the agent's ability to learn long-term dependencies and propagate value effectively.

The Ranking Prioritized Experience Replay (Ranking PER) strategy yielded the weakest results within the evaluated training window. One possible explanation is that the ranking mechanism may have introduced instability or noise in the sampling process, leading to a skewed experience distribution. If the priority ranking fails to reflect the actual learning potential of transitions, due to noisy or misleading TD errors, the agent may overfit to suboptimal or redundant experiences. However, it is worth noting that the performance curve for Ranking PER shows a positive trajectory. This upward trend suggests that, given more training time, the agent may continue improving and eventually close the gap with or even surpass the other strategies. The initially poor results might be attributed to slower convergence, implying that Ranking PER could be better suited for longer training regimes.

5.6. Generalization of the agent in other games

The final experiment in this project involved evaluating the generalizability of the best-performing agent configuration, simple DQN with a replay memory size of 250,000 and random subsampling, by training it on different games and analyzing its performance across them.

The training and evaluation rewards for Pong and Space Invaders are presented in Figures 5.9 and 5.10, and in Figures 5.11 and 5.12, respectively.

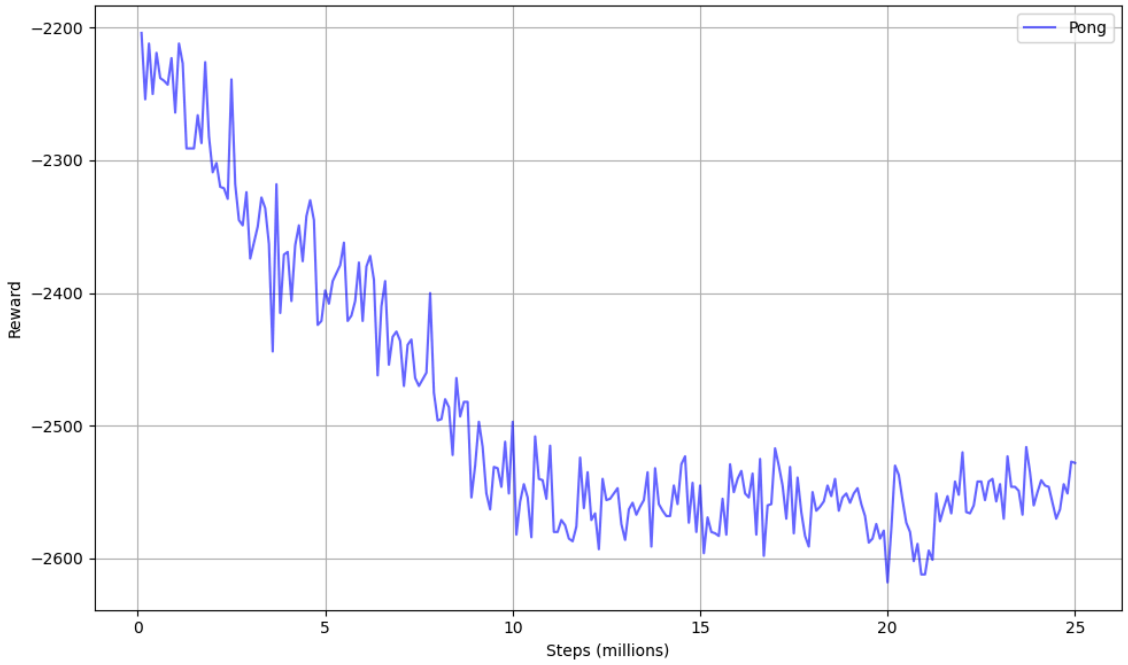


Fig. 5.9. Evolution of the rewards during the training process for the Pong game.

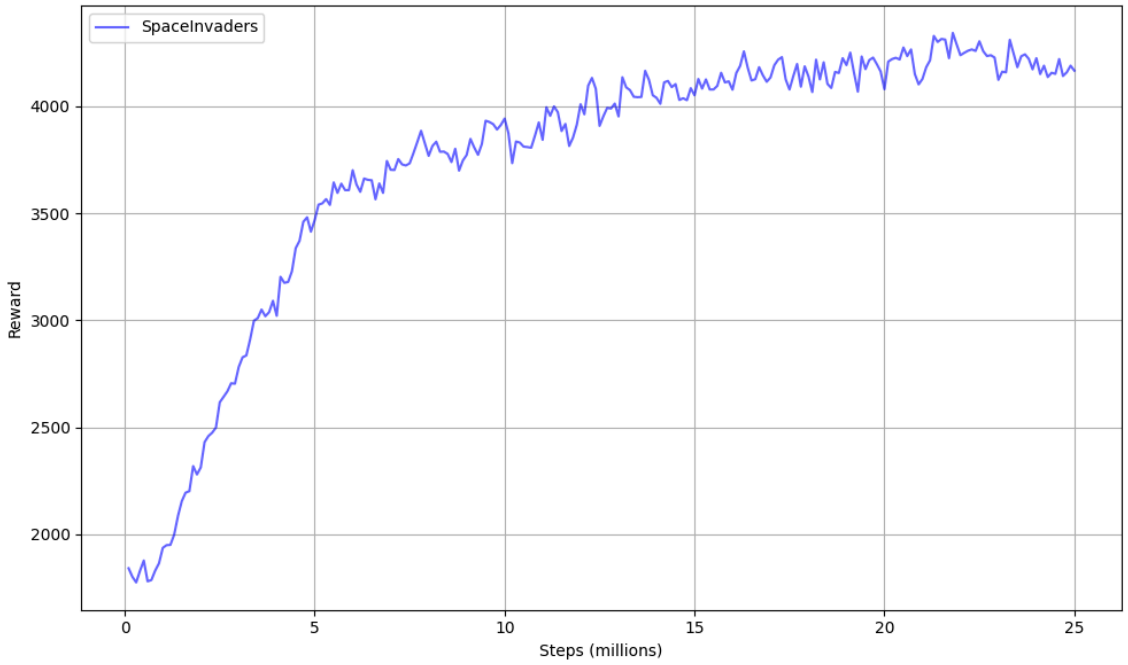


Fig. 5.10. Evolution of the rewards during the training process for the Space Invaders game.

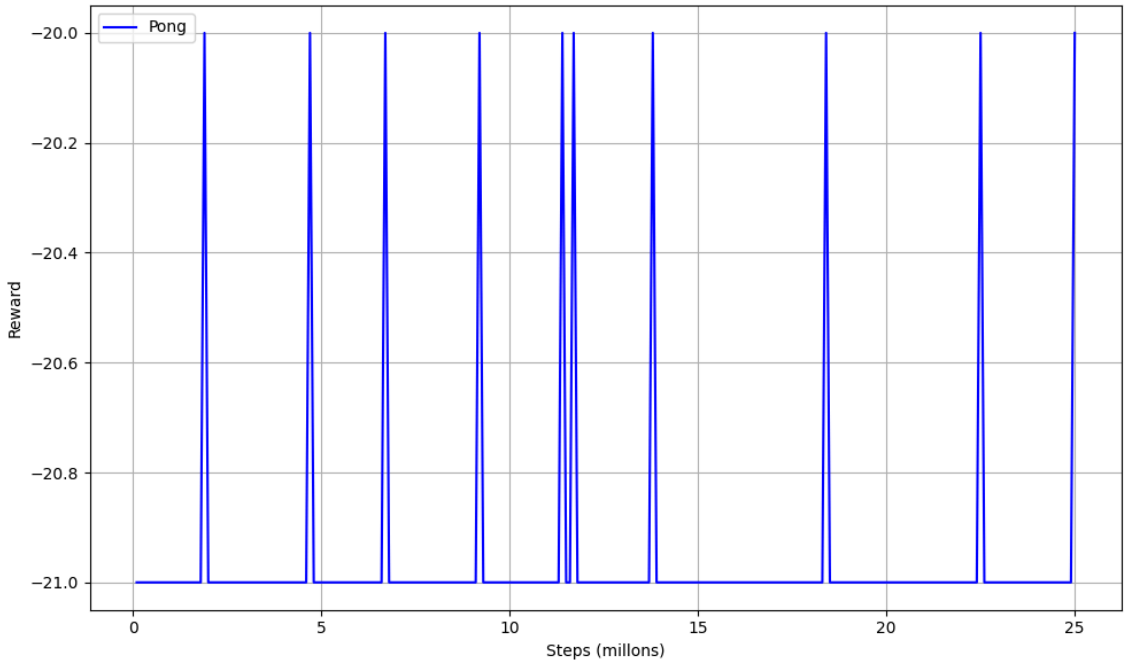


Fig. 5.11. Evolution of the rewards during the evaluation process for the Pong game.

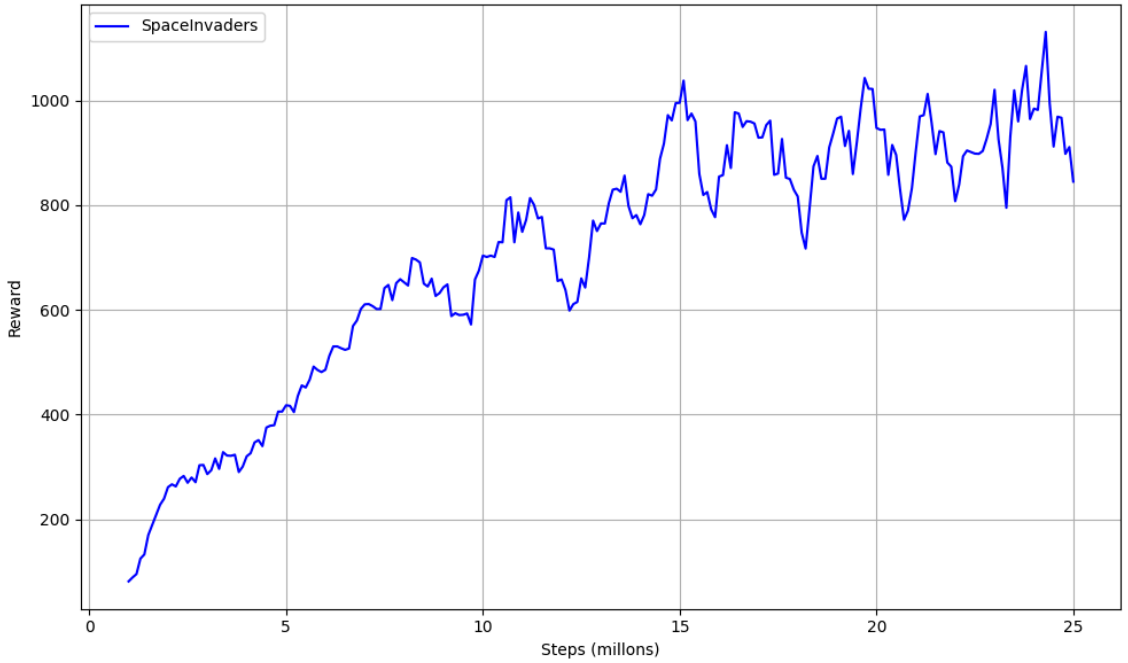


Fig. 5.12. Evolution of the rewards during the evaluation process for the Space Invaders game.

As observed, the chosen configuration demonstrates strong performance in the Space Invaders game. This can be attributed to the structural similarities it shares with Breakout. While Space Invaders introduces a slightly larger action space, requiring the agent to make more varied decisions, the absence of complex physical dynamics, such as the angled ball trajectories and unpredictable bouncing behavior found in Breakout, simplifies the learning task. In Space Invaders, the motion of the enemies is more predictable,

and the shooting mechanic is more direct, allowing the agent to focus on timing and positioning rather than mastering intricate movement physics. This relative reduction in environmental complexity likely contributes to the agent's configuration to fit well for this game.

In contrast to the results obtained in Space Invaders, the agent's performance in the Pong environment is notably poor. The reward consistently declines over the course of training, indicating that the agent is not learning an effective policy. Despite initial fluctuations, the overall trend suggests a deterioration in performance.

This suggests that the chosen configuration does not generalize well to Pong. One possible explanation for the poor performance in Pong is the sparse reward structure of the game. In Pong, the agent only receives a positive reward when it successfully scores a goal, which occurs relatively infrequently, especially in the early stages of training when the agent has not yet developed an effective strategy. This sparse and delayed feedback makes it difficult for the agent to associate its actions with successful outcomes, thereby slowing down the learning process. Unlike games such as Breakout or Space Invaders, where positive rewards can be obtained more frequently, Pong provides fewer learning signals, which likely contributes to the agent's inability to improve its policy over time.

6. CONCLUSIONS

6.1. Summary of Results

This project presented an in-depth empirical study of Deep Q-Networks (DQNs) applied to the Atari Breakout game. Through a systematic evaluation of architectural modifications, changes in the replay memory size, different subsampling strategies, and generalization, the following key findings were obtained:

- **Experience Replay Memory Size:** Larger experience replay memories, such as 400,000 transitions, showed slightly improved performance, particularly during the earlier stages of training. However, this performance gain comes at the expense of significantly higher memory usage and computational load. A size of 250,000 transitions offers a practical trade-off, achieving competitive results while maintaining reasonable resource consumption.
- **Network Architecture:** The simpler DQN architecture outperformed the recurrent GRU-based model, indicating that the temporal information captured through frame stacking is sufficient for Breakout. Adding recurrent layers increases the number of parameters and negatively impacts the training process.
- **GRU Hidden State Size:** Among the tested hidden sizes (32, 64, 128), a size of 64 yielded the best results, suggesting that a moderate capacity is most effective for partially observable environments. A smaller size like 32 may lack sufficient representational power, while a larger size like 128 can lead to overfitting and increased training complexity without significant performance gains.
- **Subsampling Strategies:** Uniform random sampling from the replay buffer consistently outperformed more complex sampling methods, including rank-based prioritized experience replay and reward-proximity sampling, due to possible overfitting to particular situations of the game.
- **Generalization:** The best agent configuration in Breakout demonstrates strong and consistent learning progress in Space Invaders, this success does not extend to the Pong environment. The favorable results in Space Invaders can be attributed to its structural similarity with Breakout and the relative simplicity of its dynamics.

  Conversely, Pong's sparse reward structure, where positive feedback is infrequent and only given upon scoring, hampers the agent's ability to learn meaningful associations between actions and outcomes. This delayed reinforcement creates a challenging learning environment that the current configuration is ill-suited to handle, resulting in poor policy development and declining rewards.

Overall, these findings suggest that while the chosen configuration generalizes well to games with more frequent and direct feedback, it struggles in environments characterized by sparse and delayed rewards.

The performance results evaluation are summarized in the Table 6.1.

Architecture	Subsampling	Replay Experience Memory	Max Score	Max Frame
Base DQN	Random	100k	324	8601
Base DQN	**Random**	**250k**	**406**	**9779**
Base DQN	Random	400k	408	11071
GRU (32)	Random	250k	342	8179
GRU (64)	Random	250k	360	9794
GRU (128)	Random	250k	42	1903
Base DQN	Reward-Proximity	250k	345	8719
Base DQN	Rank-based PER	250k	200	6690

TABLE 6.1. SUMMARY OF MODEL PERFORMANCE

The best score is achieved by the Base DQN model with random subsampling when the replay experience memory is increased to 250,000. Meanwhile, the farthest frame is reached using the same model configuration but with an even larger replay memory of 400,000. Interestingly, although the 400,000-memory model plays 1,292 frames more, it only improves the score by 2 points compared to the 250,000-memory model. This suggests that the larger memory model struggles to effectively clear the final bricks, indicating diminishing returns from increasing the replay experience memory beyond 250,000.

These results improve not only the previous work of Jorge Victoria Gijón but also the studies of Google DeepMind [3] whose maximum score in evaluation of Atari Breakout game was 225.

6.2. Application of the Project

The capabilities of Deep Reinforcement Learning (DRL) extend well beyond controlled experimental settings and benchmark environments. This project, which focuses on learning under partial observability, has practical relevance for many real-world domains where agents must act based on incomplete, noisy, or delayed information. By improving decision-making in such contexts, the methods explored in this work contribute toward building more robust, adaptive, and intelligent autonomous systems.

- **Autonomous Driving in Simulated and Real Environments:** Autonomous vehicles must make real-time decisions based on limited sensor data, often without complete visibility of their surroundings. The techniques developed in this project,

such as frame-stacking and recurrent models, can be directly applied to train agents that navigate safely and efficiently in complex driving scenarios. These include tasks like merging in traffic, responding to pedestrians, or adapting to changing weather and lighting conditions.

- **Robotics and Industrial Automation:** In robotics, agents often operate with partial information due to occlusions, limited fields of view, or sensor noise. The DRL approaches evaluated in this project are applicable to tasks such as object manipulation, warehouse automation, and autonomous drone flight. In such settings, learning policies that are robust to uncertainty is essential for reliability and safety.
- **Intelligent Game AI Development:** Games are a natural application area for DRL, particularly under partial observability where agents must infer opponent intentions or hidden game states. The methods explored here can be used to create more intelligent, adaptive non-player characters (NPCs) that exhibit strategic thinking, learn from experience, and respond dynamically to players. This enhances game immersion and reduces the need for hand-crafted behavior trees, making AI development more efficient and scalable.
- **Smart Decision Systems in Complex Environments:** Beyond robotics and games, this project's findings are relevant to decision-making systems in areas such as air traffic control, disaster response coordination, and multi-agent systems. In these domains, agents must collaborate or compete under uncertainty, requiring models that can reason effectively with limited data and evolving context.

In summary, the techniques and insights from this project contribute to the broader goal of enabling intelligent agents to operate reliably in the real world. As DRL continues to mature, its applications across transportation, robotics, entertainment, and critical infrastructure will become increasingly impactful.

6.3. Limitations

Despite the contributions, this project has several limitations:

- **Computational Constraints:** Access to GPUs plays a critical role in accelerating the training of deep reinforcement learning models. However, GPUs can be costly or limited in availability, which may lead to longer training times, reduced experimentation, and constraints on model complexity. These limitations can hinder the ability to iterate quickly or explore more advanced architectures, ultimately affecting the overall performance and scalability of the system.
- **Reward Clipping:** Although reward clipping helps stabilize training by preventing large gradients and reducing variance, it can also obscure subtle differences in reward signals. This may hinder the agent's ability to learn more nuanced strategies,

potentially slowing down convergence and limiting performance in tasks where fine-grained rewards are essential for optimal behavior.

- **Limited Training Steps:** With 25 million steps, some of the slower-learning strategies may not have reached their full potential.
- **Poor generalization in some environments:** The best configuration optimized for one game only generalizes effectively to other games that share similar reward structures. When the reward properties differ significantly, as seen in Pong, the configuration fails to support effective learning and generalization.

6.4. Future Work

Based on the current findings, several directions for future research are proposed

- **Cross-Game Generalization:** Apply the same experimental pipeline to other Atari games with varying characteristics, instead of only using the best configuration of a specific game. In games like Pong, where the reward signal is sparse, alternative subsampling strategies, beyond random subsampling, such as those implemented in this project, can enhance learning and improve performance in these challenging environments.
- **Reward function:** When designing the reward function, it is important to incorporate multiple factors beyond just the immediate score. One key aspect to consider is the time taken to complete a level, penalizing longer completion times can encourage the agent to act more efficiently and avoid unnecessarily slow strategies. Additionally, the reward function can be enhanced by including progress-based incentives, such as rewarding the agent for reaching intermediate milestones, which helps guide learning in complex environments with sparse rewards. It is also beneficial to incorporate penalties for undesirable behaviors, like repeated actions or inactivity, to discourage unproductive loops.
- **Attention:** Investigate the integration of attention mechanisms within the model architecture to improve its ability to selectively focus on the most relevant parts of the input. By directing computational resources toward key features or important time steps, attention can help balance model complexity and performance, potentially enhancing learning efficiency and interpretability. Attention mechanisms can also enable the model to better capture long-range dependencies and contextual information, which is especially useful in environments with complex or sequential data.
- **Unclipped Rewards:** Explore the impact of removing reward clipping on the training dynamics, convergence speed, and overall performance of the model.

- **Extended Training:** Allow longer training windows to observe the long-term benefits of prioritized experience replay.

BIBLIOGRAPHY

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE signal processing magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[2] X. Wang et al., "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064–5078, 2024. DOI: 10.1109/TNNLS.2022.3207346.

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

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, no. 1, p. 7 068 349, 2018.

[6] D. Yu, G. Hinton, N. Morgan, J.-T. Chien, and S. Sagayama, "Introduction to the special section on deep learning for speech and language processing," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 4–6, 2011.

[7] R. Alonso. "Diferencias entre ia, machine learning y deep learning. "[Online]. Available: https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/.

[8] X. Du, Y. Cai, S. Wang, and L. Zhang, "Overview of deep learning," in *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, IEEE, 2016, pp. 159–164.

[9] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[10] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1511.08458.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[12] D. Bhatt et al., "Cnn variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, vol. 10, no. 20, p. 2470, 2021.

[13] L. R. Medsker, L. Jain, et al., "Recurrent neural networks," *Design and applications*, vol. 5, no. 64-67, p. 2, 2001.

[14] T. Perumal, N. Mustapha, R. Mohamed, and F. M. Shiri, "A comprehensive overview and comparative analysis on deep learning models," *Journal on Artificial Intelligence*, vol. 6, no. 1, pp. 301–360, 2024. DOI: 10.32604/jai.2024.054314. [Online]. Available: http://dx.doi.org/10.32604/jai.2024.054314.

[15] S.-H. Tsang. "Review: Empirical evaluation of gated recurrent neural networks on sequence modeling (gru). "[Online]. Available: https://sh-tsang.medium.com/review-empirical-evaluation-of-gated-recurrent-neural-networks-on-sequence-modeling-gru-2adb86559257.

[16] G. A. Vouros, "Explainable deep reinforcement learning: State of the art and challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–39, 2022.

[17] M. L. Puterman, "Chapter 8 markov decision processes," in *Stochastic Models*, ser. Handbooks in Operations Research and Management Science, vol. 2, Elsevier, 1990, pp. 331–434. DOI: https://doi.org/10.1016/S0927-0507(05)80172-0.

[18] M. Coggan, "Exploration and exploitation in reinforcement learning," *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, vol. 51, 2004.

[19] A. K. Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement learning algorithms: A brief survey," *Expert Systems with Applications*, vol. 231, p. 120 495, 2023.

[20] D. Mehta, "State-of-the-art reinforcement learning algorithms," *International Journal of Engineering Research and Technology*, vol. 8, no. 1, pp. 717–722, 2020.

[21] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016. DOI: 10.1038/nature16961.

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

[23] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[24] M. M. Afsar, T. Crump, and B. Far, "Reinforcement learning based recommender systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–38, 2022.

[25] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[26] B. Hambly, R. Xu, and H. Yang, "Recent advances in reinforcement learning in finance," *Mathematical Finance*, vol. 33, no. 3, pp. 437–503, 2023.

[27] H. Tan, "Reinforcement learning with deep deterministic policy gradient," in *2021 International conference on artificial intelligence, big data and algorithms (CAIBDA)*, IEEE, 2021, pp. 82–85.

[28] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[29] P. Baheti. "Deep reinforcement learning: Definition, algorithms & uses. "[Online]. Available: https://www.v7labs.com/blog/deep-reinforcement-learning-guide.

[30] R. Liu and J. Zou, "The effects of memory replay in reinforcement learning," in *2018 56th annual allerton conference on communication, control, and computing (Allerton)*, IEEE, 2018, pp. 478–485.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, 2016. arXiv: 1511.05952 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1511.05952.

[33] G. Papoudakis, F. Christianos, and S. Albrecht, "Agent modelling under partial observability for deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 210–19 222, 2021.

[34] J. V. Gijón, "Algoritmos de aprendizaje por refuerzo profundo para el entrenamiento de agentes software en videojuegos," Bachelor's Thesis, Universidad Carlos III de Madrid, Madrid, Spain, 2024.

[35] L. Fei-Fei, J. Johnson, and S. Yeung, *Cs231n convolutional neural networks for visual recognition: Neural networks part 3: Backpropagation*, 2024. [Online]. Available: https://cs231n.github.io/neural-networks-3/.

[36] European Parliament and Council of the European Union, *Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence (artificial intelligence act)*, 2024.

[37] Parlamento Europeo y Consejo de la Unión Europea, *Reglamento (ue) 2016/679 del parlamento europeo y del consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la directiva 95/46/ce (reglamento general de protección de datos)*, 2016.

[38] International Organization for Standardization (ISO), *ISO/IEC JTC 1/SC 42 - Artificial Intelligence*, 2023.

[39] J. I. Schaap, "The growth of the native american gaming industry: What has the past provided, and what does the future hold?" *American Indian Quarterly*, vol. 34, no. 3, pp. 365–389, 2010.

[40] S. A. Yakan, "Analysis of development of artificial intelligence in the game industry," *International Journal of Cyber and IT Service Management*, vol. 2, no. 2, pp. 111–116, 2022.

[41] Statista Research Department, *Industria mundial del videojuego - datos y cifras*, 2023. [Online]. Available: https://es.statista.com/temas/9150/industria-mundial-del-videojuego/.

[42] P. Zackariasson and T. L. Wilson, *The video game industry: Formation, present state, and future*. Routledge, 2012.

[43] Infobae, *En los próximos tres años las empresas necesitarán un millón de especialistas en inteligencia artificial y aprendizaje automático*, 2024. [Online]. Available: https://www.infobae.com/fortune/2024/01/31/en-los-proximos-tres-anos-las-empresas-necesitaran-un-millon-de-especialistas-en-inteligencia-artificial-y-aprendizaje-automatico/.

[44] L. Alekseeva, J. Azar, M. Giné, S. Samila, and B. Taska, "The demand for ai skills in the labor market," *Labour economics*, vol. 71, p. 102 002, 2021.

[45] C. Yu, J. Liu, S. Nemati, and G. Yin, "Reinforcement learning in healthcare: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–36, 2021.

[46] Y. Li, *Reinforcement learning in practice: Opportunities and challenges*, 2022. arXiv: 2202.11296 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2202.11296.

A. PROJECT PLANNING

A.1. Task description

The project was planned and done according different tasks.

- **Task 1 – Define the Objective:** Establish the project's topic and main goal.
  - **Task 1.1 – Problem Definition:** Identify the core problem and outline potential approaches for solving it.
  - **Task 1.2 – Research Potential Solutions:** Investigate existing methods and tools relevant to achieving the project's goal.
  - **Task 1.3 – Background Study:** Gain foundational knowledge in Deep Reinforcement Learning and Atari environments.
- **Task 2 – System Implementation:** Develop the Deep Reinforcement Learning system.
  - **Task 2.1 – Tool Familiarization:** Understand the libraries and workflows used in Deep Reinforcement Learning.
  - **Task 2.2 – Frame Preprocessing:** Implement preprocessing steps such as frame cropping and grayscale conversion.
  - **Task 2.3 – DQN Development:** Implement both the baseline DQN and its RNN-based variant.
  - **Task 2.4 – Environment Setup:** Configure the agent and the game environment.
  - **Task 2.5 – Experience Replay:** Develop and implement various subsampling strategies for experience replay memory.
  - **Task 2.6 – Training Loop:** Build the training loop to iteratively improve the agent.
  - **Task 2.7 – Saving Mechanism:** Implement automatic saving of models, replay memories, and performance metrics.
- **Task 3 – Training:** Train agents using different configurations.
  - **Task 3.1 – Base DQN Training:** Train the baseline DQN with varying experience replay sizes.
  - **Task 3.2 – RNN Variant Training:** Train the RNN-based agent with different hidden state sizes.

  - **Task 3.3 – Subsampling Strategies:** Evaluate agent performance using different replay memory subsampling techniques.

- **Task 4 – Evaluation:** Assess agent performance and select the best configuration.
  - **Task 4.1 – Evaluation Setup:** Configure agents for evaluation in exploitation mode.
  - **Task 4.2 – Performance Measurement:** Use predefined metrics to evaluate agent performance.
  - **Task 4.3 – Final Agent Selection:** Identify and select the most effective agent.

- **Task 5 – Generalization:** Test the selected agent configuration in new scenarios.
  - **Task 5.1 – Game Selection and Preprocessing:** Choose new games and implement appropriate frame preprocessing.
  - **Task 5.2 – Cross-Game Training:** Train the final agent configuration on these new games.
  - **Task 5.3 – Cross-Game Evaluation:** Evaluate performance and generalization capabilities in the new environments.

- **Task 6 – Final Report and Defense:** Document and present the project results.
  - **Task 6.1 – Report Planning:** Design the structure and outline of the final report.
  - **Task 6.2 – Report Drafting:** Write the full initial draft of the report.
  - **Task 6.3 – Review and Revision:** Edit, refine, and polish the report; incorporate advisor feedback.
  - **Task 6.4 – Defense Preparation:** Prepare the presentation and rehearse for the final defense, including Q&A readiness.

A.2. Task sequencing

The task that has been explained above, are now indicated how much time does it took to develop them and which were the other task that depend in order to be done.

Task	Duration	Dependencies
Task 1-Define the Objective		
Task 1.1-Problem Definition	1 week	
Task 1.2-Research Potential Solutions	3 weeks	1.1
Task 1.3-Background Study	2 weeks	1.2
Task 2-System Implementation		
Task 2.1-Tool Familiarization	1 week	1.3
Task 2.2-Frame Preprocessing	1 week	2.1
Task 2.3-DQN Development	2 week	2.1
Task 2.4-Environment Setup	1 week	2.2, 2.3
Task 2.5-Experience Replay	2 week	2.4
Task 2.6-Training Loop	1 week	2.5
Task 2.7-Saving Mechanism	1 week	2.6
Task 3-Training		
Task 3.1-Base DQN Training	3 week	2.7
Task 3.2-RNN Variant Training	3 weeks	3.1
Task 3.3-Subsampling Strategies	3 weeks	3.2
Task 4- Evaluation		
Task 4.1-Evaluation Setup	1 week	2.4
Task 4.2-Performance Measurement	1 weeks	4.1, 3.3
Task 4.3-Final Agent Selection	1 week	4.2
Task 5-Generalization		
Task 5.1-Game Selection and Preprocessing	1 week	2.4
Task 5.2-Cross-Game Training	2 week	4.3, 5.1
Task 5.3-Cross-Game Evaluation	1 week	5.2
Task 6-Final Report and Defense		
Task 6.1-Report Planning	1 week	5.3
Task 6.2-Report Drafting	6 weeks	6.1
Task 6.3-Review and Revision	3 week	6.2
Task 6.4-Defense Preparation	1 week	6.3

TABLE A.1. TASKS DURATIONS AND DEPENDENCIES.

As shown in Table A.1, the most time-consuming tasks were those related to code development and those dependent on training durations. Code implementation required extensive debugging and validation, which naturally extended the timeline. Similarly, training-related tasks were inherently time-intensive due to the high computational demands of reinforcement learning and the prolonged durations needed to upload large experience replay memory files to the cloud. The substantial size of these files further compounded the delay. Moreover, reinforcement learning agents typically require extended training periods to achieve acceptable performance. These combined factors significantly

contributed to the overall time investment of the project.

A.3. Gantt chart

For a detailed overview of task distribution and the project's total duration, a Gantt chart can illustrate the structure and dependencies of various tasks.

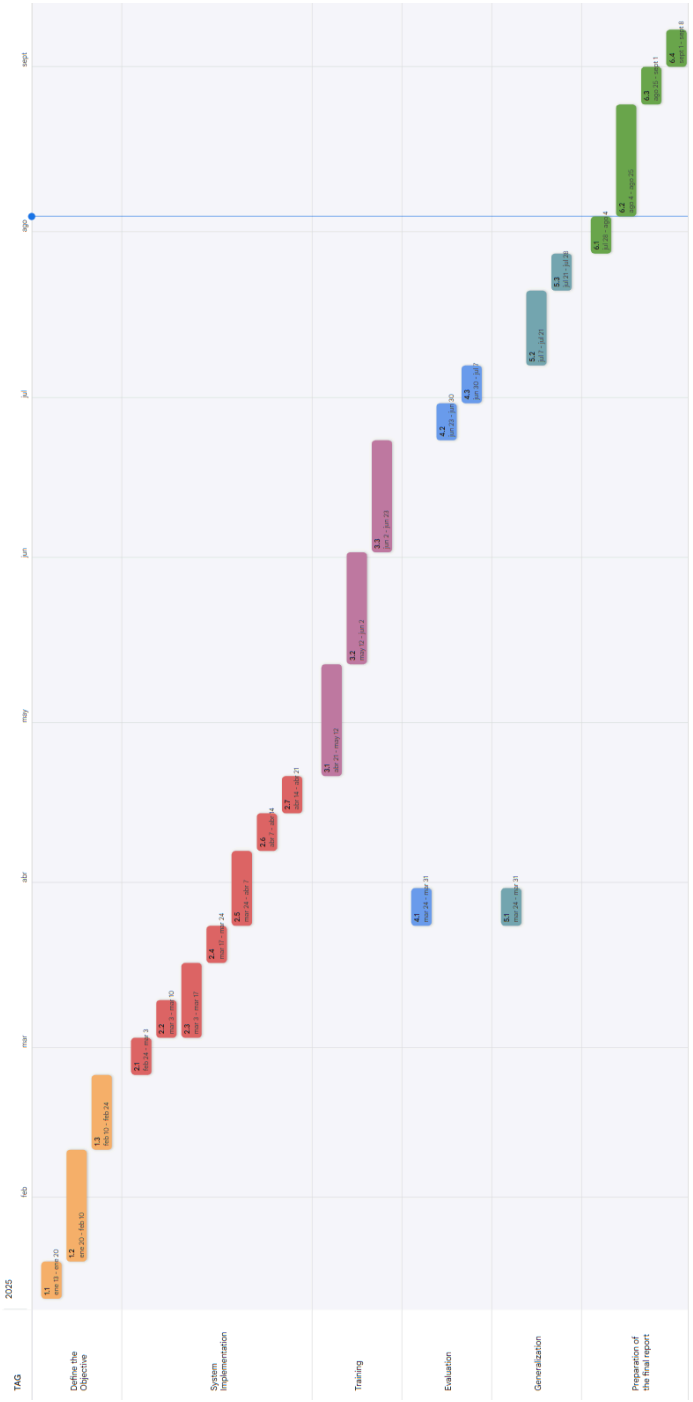


Fig. A.1. Gantt chart.

B. BUDGET

The undertaking of this initiative has necessitated the allocation of fiscal resources, which are enumerated below. They have been divided into two broad categories: expenditures related to human capital and outlays associated with material assets.

B.1. Human resources

This project was developed by a university student pursuing a Dual Bachelor in Data Science and Engineering and Telecommunication Technologies Engineering at Universidad Carlos III. The student's work was supervised by his advisor, Iván González Díaz. To accurately allocate funds for the human resources involved, the hours contributed by both individuals must be considered.

The project spanned approximately a total of 30 weeks. During this period, the student and his advisor held 10 meetings, averaging 1 hour per meeting, totaling approximately 10 hours. Additionally, the advisor dedicated extra time outside these meetings to review drafts and provide feedback via email, amounting to 1 extra hour per month and an additional 10 hours in the final month, totaling 17 hours. Furthermore, the advisor spent an extra 10 hours assisting with the report of the project, defense and related activities. Overall, the advisor contributed 37 hours of guidance to the project.

For financial estimation, the hourly pay rate for an experienced engineer or technical advisor in this field is approximately 24€ in Madrid according to Glassdoor. Assuming a 40-hour work week over 4 weeks, this equates to a monthly salary of around 3,840€. Therefore, the advisor's 37 hours of work amount to an estimated cost of 888€.

For a student engineer, an annual base salary of 30,000€ is assumed based on Glassdoor, Glassdoor, translating to 2,500€ per month or 15.60€ per hour (based on a 160-hour work month). For cost calculations, this rate is rounded up to €16 per hour.

The student was solely responsible for the project workload, estimated at about 12 hours per week for 30 weeks, resulting in a total of 360 hours. Consequently, the estimated cost of human resources attributable to the student is 5,760€.

B.2. Hardware and software costs

In terms of hardware, the project has been carried out with an Asus ZenBook which costs 899€. The depreciation cost over the duration of the project is estimated to be 77.59€, considering a lifetime of 7 years.

$$\text{Amortization} = \left(\frac{\text{cost} \times \text{time of use}}{\text{lifetime}}\right) = \left(\frac{899 \times \frac{7.25}{12}}{7}\right) = 77.59\,€$$

Apart from the computer, an NVIDIA GPU was used for the training of the models, specifically an NVIDIA GeForce RTX 3060 with 12 GB of RAM. It has a cost of 439.90€ and a lifetime of 4 years, resulting in a depreciation cost of 66.44€.

$$\text{Amortization} = \left(\frac{439.90 \times \frac{7.25}{12}}{4}\right) = 66.44\,€$$

With respect to the software, the following open source programs have been used, therefore they do not involve an additional cost:

- Google Colab: A free to use online development environment for Python software.
- Google Sheets: A free online spreadsheet software.
- Google Drive: Free online storage and file sharing platform.
- Overleaf: A free online LaTeX editor for writing and collaborating on technical documents.

All libraries and models were open-source so they do not contribute to the costs, but the estimated cost of consumable materials such as access to the internet, light and other supplies would represent an additional 10% of the total cost.

B.3. Total cost

In Table B.1 the final costs of the project can be seen, adding up all of the human resources needed and their underlying costs, the hardware cost and the additional cost of supplies.

Type	Description	Cost
Human resources	Tutor	888€
Human resources	Student	5,760€
Hardware costs	Laptop	77.59€
Hardware costs	GPU	66.44€
Cost of consumables	Other (10%)	733.34€
Total costs		7,525.37€

TABLE B.1. TOTAL BUDGET.

C. REGULATORY FRAMEWORK

The regulatory framework surrounding this project, particularly concerning the development and implementation of deep reinforcement learning algorithms in video games, is complex due to its specific legislation and ongoing evolution.

The regulation of Artificial Intelligence (AI), including deep reinforcement learning, is primarily outlined in Regulation COM/2024/206 final, commonly referred to as the Artificial Intelligence Act [36], recently published by the European Commission. This regulation categorizes AI applications according to their risk level. In the case of video games and entertainment AI systems, these are generally classified as low-risk. However, they may still be subject to regulations that ensure transparency and responsible use.

In addition, the General Data Protection Regulation (GDPR) 2016/679 of the European Union [37] imposes binding obligations on how personal data is collected, stored, and processed. As a result, any project that handles user data—such as video games that monitor player behavior—must comply with the relevant provisions of this regulation.

Furthermore, innovations in AI and reinforcement learning algorithms can be patentable by their developers. This allows companies to protect their technological advancements in the field and gain a competitive edge.

Video games are also subject to content regulations. In Europe, for instance, organizations like PEGI[14] (Pan European Game Information) classify video games based on their content, which is often evaluated based on the presence of violence or other controversial elements.

Lastly, it's important to highlight the role of ISO/IEC JTC 1/SC 42 [38], a subcommittee under the joint technical committee of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). This subcommittee supports the development of international standards, documentation, and technical specifications in the field of artificial intelligence.

[14] https://pegi.info/es

D. SOCIO-ECONOMIC FRAMEWORK

The development of deep reinforcement learning algorithms in the context of video games is situated within a socio-economic environment marked by the rapid growth of the technology industry and the increasing interest in artificial intelligence and machine learning [39] [40].

First and foremost, it is important to highlight that the video game industry has experienced exponential growth over recent decades, becoming one of the most profitable sectors in global entertainment. As noted in [41], revenues in 2023 exceeded 180 billion U.S. dollars. Moreover, video games are not only a major source of entertainment but have also become a significant testing ground for the development of advanced technologies such as artificial intelligence. Game companies are constantly striving to innovate in order to offer users more immersive and intelligent experiences [42]. This has led to a growing demand for AI techniques that enable the creation of intelligent agents capable of learning and adapting in real-time to dynamic environments.

In addition, as confirmed in [43] and [44], the demand for experts in artificial intelligence and machine learning is on the rise. Universities and training centers are gradually responding to this demand by expanding their programs and courses focused on these disciplines, preparing future professionals to contribute to the continued growth of the video game industry and AI specialists demand.

While deep reinforcement learning plays a crucial role in gaming, it also has transformative applications in other key sectors. In the automotive industry [25], these algorithms are being actively researched and developed to allow vehicles to make real-time decisions in complex environments. In healthcare [45], they are sometimes used to optimize personalized treatments and support clinical decision-making.

Finally, the advancement of deep reinforcement learning presents several challenges and opportunities [46]:

- **Ethical Challenges:** The implementation of AI in video games and other sectors raises ethical concerns, such as the potential for uncontrolled manipulation of user behavior, video game addiction, personal data protection, and the displacement of certain job roles due to automation.
- **Access Inequality:** As with many technological advances primarily developed in the Global North, there is a risk that unequal access to AI technologies and training could widen the existing socio-economic gap, limiting opportunities for less developed regions.
- **Economic Growth:** On the other hand, AI and reinforcement learning are poised

to drive economic growth across all application sectors, generate highly skilled employment, and enhance the competitiveness of companies in the global market.

E. VIDEO OF THE BEST AGENT DEVELOPED PLAYING THE ATARI GAMES

The video showcasing a full Breakout game played by the best agent, continuing until all five lives are lost, can be viewed at the following Youtube link: https://youtube.com/shorts/3aLauuLlEWU

The video showcasing a full Space Invaders game played by the best agent, can be viewed at the following Youtube link: https://youtube.com/shorts/JrK0jPy5XQo?feature=share

The video showcasing a full Pong game played by the best agent, can be viewed at the following Youtube link: https://youtube.com/shorts/PoFstpEMzlM?feature=share

DECLARATION OF USE OF GENERATIVE ARTIFICIAL INTELLIGENCE (AI) IN BACHELOR THESIS (TFG)

I have used Generative AI in my TFG

Check all that apply:

YES	NO

YES

If you have ticked YES, please complete the following 3 parts of this document:

Part 1: Reflection on legal, ethical and responsible behavior

Please be aware that the use of AI carries some risks and may lead to a series of serious academic consequences: the TFG will not be evaluated by the University if the use of AI involves the use of confidential data, copyrighted materials, or personal data, used without complying with the conditions required in each case (authorization of the interested parties, authorization of the owners, following the instructions of the University).

Question	
1. In my interactions with AI tools, I have provided **confidential data**, always with the appropriate authorization of the data subjects. Confidentiality encompasses any information that a person or organization wishes to protect for legal, commercial, privacy, or strategic reasons (such as patents or trade secrets).	
YES, I have used this data with permission	NO, I have not used confidential data
2. In my interaction with Generative AI tools, I have submitted **copyrighted materials** with the permission of those concerned.	
YES, I have used these materials with permission from the copyright holders, or without permission because they fall within one of the exceptions or limitations permitted by law: • Public domain work • Licensed work (Creative Commons licenses) • Use of excerpts for research purposes (fair use)	NO, I have not used copyrighted materials

3. In my interaction with Generative AI tools, I have submitted **personal data** with the consent of the data subjects.	
YES, I have used this data with the authorization of the interested parties and in accordance with the instructions contained in the guide approved by the University.	NO, I have not used personal data
4. My use of the Generative AI tool has **respected its terms of use**, as well as the essential ethical principles, not being maliciously oriented to obtain an inappropriate result for the work presented, that is to say, one that produces an impression or knowledge contrary to the reality of the results obtained, that supplants my own work or that could harm people.	
YES	NO

Part 2: Declaration of technical use

Use the following model statement as many times as necessary, in order to reflect all types of iteration you have had with Generative AI tools. Include one example for each type of use where indicated: *[Add an example].*

I declare that I have made use of the Generative AI system ChatGPT **for:**

Documentation and drafting:

- *Supporting reflection in relation to the development of the work: iterative process of analysing alternatives and approaches using AI*

I have asked for a list of alternatives to address the problem of subsampling in DQN.

- *Revision or rewriting of previously drafted paragraphs*

I have asked for paragraphs to be rewritten to shorten the conclusions while keeping the same ideas already written.

- *Search for information or answers to specific questions*

Not used for this.

- *Bibliography search*
  Not used for this.

- *Summary of bibliography consulted*
  Not used for this.

- *Translation of texts consulted*
  Not used for this.

Develop specific content

Generative AI has been used as a support tool for the development of the specific content of the TFG, including:

- *Assistance in the development of lines of code (programming)*

I used it to optimize some parts of the training code.

- *Generation of diagrams, images, audios or videos*

Not used for this.

- *Optimisation processes*

I used it to optimize some parts of the training code.

- *Data processing: collection, analysis, cross-checking of data...*

Not used for this.

- *Inspiration of ideas in the creative process*

I used it to obtain other approaches to the subsampling part of DQN.

- *Other uses linked to the generation of specific points of the specific development of the paper*

  Not used for this.

Part 3: Reflection on utility

Please provide a personal assessment (free format) of the strengths and weaknesses you have identified in the use of Generative AI tools in the development of your work. Mention if it has helped you in the learning process, or in the development or drawing conclusions from your work.

GenAI has the advantage of performing very well in areas such as writing and code generation and optimization. However, many of the responses it provided were not directly applicable, and a deep understanding of what is expected from these models is necessary. It can support the learning process by addressing speceific doubts and offering alternative perspectives, but caution is needed, as the answers are not always correct. Critical thinking is essential when using these tools.