AI

MDPI

Article

Optimization Strategies for Atari Game Environments: Integrating Snake Optimization Algorithm and Energy Valley Optimization in Reinforcement Learning Models

Sadeq Mohammed Kadhm Sarkhi [1,*] and Hakan Koyuncu [2]

[1] Electrical and Computer Engineering Department, Altinbas University, 34200 Istanbul, Turkey
[2] Computer Engineering Department, Altinbas University, 34217 Istanbul, Turkey; hakan.koyuncu@altinbas.edu.tr
* Correspondence: sadeqseven@gmail.com

Abstract: One of the biggest problems in gaming AI is related to how we can optimize and adapt a deep reinforcement learning (DRL) model, especially when it is running inside complex, dynamic environments like "PacMan". The existing research has concentrated more or less on basic DRL approaches though the utilization of advanced optimization methods. This paper tries to fill these gaps by proposing an innovative methodology that combines DRL with high-level metaheuristic optimization methods. The work presented in this paper specifically refactors DRL models on the "PacMan" domain with Energy Serpent Optimizer (ESO) for hyperparameter search. These novel adaptations give a major performance boost to the AI agent, as these are where its adaptability, response time, and efficiency gains start actually showing in the more complex game space. This work innovatively incorporates the metaheuristic optimization algorithm into another field—DRL—for Atari gaming AI. This integration is essential for the improvement of DRL models in general and allows for more efficient and real-time game play. This work delivers a comprehensive empirical study for these algorithms that not only verifies their capabilities in practice but also sets a state of the art through the prism of AI-driven game development. More than simply improving gaming AI, the developments could eventually apply to more sophisticated gaming environments, ongoing improvement of algorithms during execution, real-time adaptation regarding learning, and likely even robotics/autonomous systems. This study further illustrates the necessity for even-handed and conscientious application of AI in gaming—specifically regarding questions of fairness and addiction.

Keywords: SOA; EVO; reinforcement learning; metaheuristic; agent; ESO

Citation: Sarkhi, S.M.K.; Koyuncu, H. Optimization Strategies for Atari Game Environments: Integrating Snake Optimization Algorithm and Energy Valley Optimization in Reinforcement Learning Models. *AI* **2024**, *5*, 1172–1191. https://doi.org/10.3390/ai5030057

Academic Editor: Gianni D'Angelo

Received: 15 May 2024
Revised: 10 June 2024
Accepted: 14 June 2024
Published: 17 July 2024

1. Introduction

In the realm of artificial intelligence, machine learning stands as a cornerstone, offering a diverse spectrum of applications that significantly impact various sectors. Among its numerous methodologies, three distinct learning paradigms dominate—supervised learning (SL) [1], unsupervised learning (UL) [2], and reinforcement learning (RL) [3]—each contributing uniquely to the advancement of AI technologies. SL [1], with its foundation in input–output data correlations, has been integral in developing practical applications such as spam detection [4], image recognition, object detection [5], and predictive analytics [6]. UL [2], on the other hand, ventures into the territory of identifying underlying patterns in unlabeled data, employing techniques like k-means clustering and principal component analysis for data classification and dimensionality reduction [7,8].

Deep learning (DL), an extension of conventional machine learning, has emerged as a critical development, especially in its ability to automate feature extraction and process vast datasets [9]. This advancement is particularly evident in fields requiring the analysis of unstructured data types like images and audio. However, it is the dynamic and complex

world of Atari gaming [10] that presents a unique challenge, one that necessitates a blend of DRL and innovative optimization techniques.

This paper delves into the application of DRL, specifically the deep Q-Network (DQN) approach, within the context of the Atari game "PacMan". The focus lies on addressing the challenges inherent in training DRL agents for high-performance gameplay in complex, dynamic environments [10]. The exploration and optimization of DRL algorithms, particularly in the absence of extensive datasets and relying on real-time interactions, underscore the novelty and complexity of this research. Moreover, the integration of metaheuristic optimization techniques, such as Snake Optimization and Energy Valley Optimization, represents a novel approach in fine-tuning the parameters of DRL models for enhanced performance in such intricate gaming scenarios.

The significance of this study is manifold. Firstly, it contributes to the growing body of knowledge in the field of AI and gaming by providing a comprehensive analysis of DRL applications in Atari gaming. Secondly, it addresses the prevalent research gap in the optimization of DRL algorithms for complex gaming environments [10], offering a model that combines the strengths of DRL with advanced optimization strategies. Finally, the study sets a precedent for future research, paving the way for further advancements in AI-driven gaming and beyond.

In sum, this paper aims to rigorously explore, develop, and evaluate DRL models tailored for the "PacMan" game, integrating advanced optimization techniques to enhance the models' performance. This endeavor not only challenges existing paradigms in AI gaming but also lays the groundwork for future innovations in the field.

2. Literature Review

RL has received enormous attention in the domain of gaming and has deployed a learning-by-exploration paradigm for the optimization of both short- and long-run rewards. Strikingly, an RL model was designed for Othello gaming, and this model was trained to play Othello on its own [11]. The results are very striking with the application of DRL, specifically in the Atari games, which significantly improved the outcomes when a mixture of Q-learning and CNN in the processing of the image pixels that generated the value functions was applied. In essence, this shows the improvement generated in seven different Atari games, from the best known to challenging ones [12]. The realm of RL has helped to create interactive learning experiences within educational gaming environments and has also unlocked a new horizon for educational technology [13–15]. Self-play is one of the techniques of RL—where the agents improve their skill by the play between them. The technique has been very successful within the domain of gaming for a long period of time. The best example of its use is in the game of Go, with the development of the AlphaGo model, one of the best models. By combining neural networks and Montecarlo tree search, it largely outperformed all the existing Go programs. The techniques were later validated in a general way with the AlphaZero framework, which was able to be applied to games such as Go, Chess, and Shogi, where Shogi is a Japanese variant of chess. The AlphaZero framework proved to be effective in learning game rules without knowing the domain and defeating world champions in all the three games [16]. Real-time strategy games have been conquered by RL. In Boulder Dash, various DRL techniques such as DQN were applied to reduce decision latency [17]. In addition, experience replay is moved to the actor-critic algorithm, and tasks being solved with this algorithm are continuous and discrete tasks—by, for example, Atari and Mujoco benchmarks. This brings a new value function estimation beyond classic Q-learning [18]. StarCraft II is, by all means, one of the most well-known games with respect to both complexity and competitiveness, and at the same time, one of the most well-known subjects of AI studies, especially regarding its importance in esport events [19]. In an unprecedented event at the beginning of 2019, Google DeepMind introduced AlphaStar. Some months later, this AI won its first significant victory against a top professional player, Grzegorz Komincz, in the game of StarCraft II. The final score was 5:0 [20]. AlphaStar is unique in its integration of DL and RL and DNN

(deep neural network) that, consequently, can process raw data efficiently [21]. Central to the design of this system is the "Transformer" AI architecture. It has attention mechanisms based on RNN (recurrent neural networks) and CNN (convolutional neural networks) support. Further, the system is advanced with an LSTM core and adaptive strategies to enhance "black box" quality [22,23]. AlphaStar's learning system first used the SL, which was pre-trained to the imitation level by watching and copying human gameplay. After this pre-training, the Multiagent mix AlphaStar—a system of AI agents—fought in ubiquitous battles over two weeks, further tweaking and enhancing their in-game capabilities to advance past both singular and collective enemies. The system uses the off-policy actor-critic approach, experience replay, and self-imitation to improve the neural networks weights [24]. In another original work, scientists developed a new DRL framework for text-based adventure games [25]. This system employed a knowledge graph to correctly prune actions and used question-answering techniques for training. The system had superior utility in comparison to current techniques. Independently, another DRL-based agent, LeDeepChef, demonstrated its prowess in text-based gaming when it clinched a second position in the First TextWorld Problems competition organized by Microsoft Research [26]. Pushing the boundaries of using RL, the ReBeL self-play AI framework exhibited tremendous effectiveness in imperfect information games, for example, Texas hold'em poker, with a cocktail of RL and search techniques [27]. To push DRL further, it has been applied in the conquest of intricate control problems in 1v1 MOBA games. An example is the application of "Honor of Kings" by the Solo AI agent from Tencent. More specifically, the building of algorithms to conquer mainstream games has been a domain primarily accessible to large technology corporations, among which are Google's DeepMind and Microsoft. Most of the pioneering work has been reported in the space of RL. It bears repeating that almost all of the comparisons between the AI players are performed based on games like Atari and fundamental board games; there is no performance benchmarking or societal understanding set for strategy or MOBA games, and this is the biggest hurdle. Again, the work from DeepMind, for example, the performance of DQN algorithms that combine Q-learning with DNNs in Atari games, has set the benchmark, since there is a measurement of human performance to which to compare [28].

Machine learning encompasses various methodologies, with three primary types being supervised, unsupervised, and reinforcement learning (RL) [3]. Each type has unique capabilities, particularly in handling data and learning from it. Machine learning techniques typically require explicit programming to extract features from data. In contrast, deep learning automates this process, allowing it to manage massive datasets efficiently and effectively process complex, unstructured data like images and audio.

Due to their adaptive capacities, deep learning models are particularly suited to dynamic and real-time applications such as recommendation systems and autonomous vehicles. However, these applications often face challenges due to the lack of extensive datasets. RL [8] addresses these challenges by enabling models to learn optimal behaviors through direct interactions with their environment, making it ideal for scenarios requiring sequential decision making. The RL agent receives feedback in the form of rewards, guiding it toward beneficial actions while penalizing fewer effective ones. This process involves balancing exploration of new strategies and exploitation of known paths to maximize rewards.

RL's adaptive learning approach is beneficial for recommendation systems (RS), which struggle with issues like cold starts and data sparsity. RL-based RS [29] improves the accuracy, relevance, and diversity of recommendations across various domains, including news, education, retail, and entertainment. When combined with deep learning, RL can also process high-dimensional sensory inputs, as demonstrated in applications like playing Atari games.

The deep Q-Network (DQN) is a prominent algorithm in deep RL, addressing decision-making challenges in environments with high-dimensional sensory inputs. The multi-armed and contextual bandits are simpler forms of RL that focus on the exploration–ex-

ploitation trade-off, crucial for personalized recommendations. Popular algorithms in this area include the upper confidence bound (UCB), Thomson sampling, and LinUCB.

In security, particularly within the Internet of Things (IoT), RL is leveraged to enhance protection against threats, though its application is primarily within simulated environments due to the high costs of real-world implementation. RL also plays a significant role in robotics, especially in developing social robots for healthcare applications. These robots employ cognitive empathy to better interact with and care for the elderly.

In the realm of natural language processing, RL enhances performance significantly. Inverse reinforcement learning (IRL) [30], a variant of RL, is particularly useful in environments where the reward structures and transition probabilities are unknown. IRL learns from observing expert behaviors, aiming to replicate these actions effectively.

Notably, RL has shown impressive results in gaming, where models like AlphaZero have learned complex games through self-play without prior domain knowledge, eventually surpassing human champions. Automated decision-making processes using DQN have been applied successfully in real-time gaming environments like Boulder Dash [31], and enhancements such as experience replay have refined the efficiency of actor-critic algorithms [18]. These techniques allow RL models to operate effectively across both continuous and discrete tasks, demonstrating their versatility and robustness.

Natural language processing (NLP) employs various applications such as intelligent assistants, language translation, and text analytics. A review discussed the use of reinforcement learning (RL) in NLP, focusing on applications including syntactic parsing, language understanding, text generation, and machine translation [32].

RL models are particularly adept at navigating environments that are in constant flux, and when combined with deep reinforcement learning (DRL), they tend to yield superior outcomes. Text summarization, which comes in two forms—extractive and abstractive—is an area where RL has been applied. Extractive summarization focuses on identifying key sentences, whereas abstractive summarization involves rephrasing and condensing text, which is more complex. A comprehensive review covered various aspects of automatic text summarization through RL and transfer learning, examining algorithms, datasets, challenges, solutions, and performance evaluation [33].

In the domain of automatic trading, models such as ResNet and LSTM have been shown to outperform RL-based approaches [34]. A novel application of a random neural network utilizing DRL predicted trends in market data, including upward, downward, and neutral movements [35]. For stock market analysis, it has been suggested that a short-term memory approach is more effective than relying on a long-term historical analysis. Additionally, a multi-agent DQN strategy was specifically designed for automated trading, with fine-tuning of hyperparameters such as activation functions, the number of Q-Networks, learning rates, and discount factors, particularly as tested on a Forex (EUR/USD) dataset [36].

The use of DQN has also been extended to financial models for calculating credit scores, incorporating a dynamically changing reward function [37]. Inverse reinforcement learning (IRL) is applied in situations where it is impractical for developers to specify reward functions explicitly. IRL enables the modeling of expert behavior in RL agents, improving their performance in desired tasks. This technique has been successfully applied in multiplayer, non-cooperative settings [38]. Furthermore, a data-driven approach to IRL has been proposed to enhance learning in multiplayer environments [39].

These studies collectively demonstrate the broad applicability and effectiveness of RL and DRL across various fields, from NLP and automatic trading to complex multiplayer environments, showcasing their capability to adapt and excel in dynamically changing conditions.

3. Critical Analysis and Contribution

In spite of the huge advancements in the field of RL, there are still significant gaps in terming DRL models for complex gaming environments. Most of the research work is

dedicated to traditional RL and DRL techniques and hardly provides detailed exploration in advanced optimization strategies. Promising routes of enhancement for the model's performance are provided by this thin line of research in the cases served by the fusion of metaheuristic optimization techniques like Snake Optimization and the Energy Valley Optimization Technique with the DRL framework. Incorporating DRL through advanced optimization technologies will close this gap, so that the hyperparameters can be fine-tuned to make AI agents more adaptable and efficient with changing gaming environments. This novel methodology in the present research not only contributes to the field of AI and gaming but also makes way for new research arrangements that shall inevitably benefit AI-driven development in games and so forth.

4. Materials and Methods

The refined approach to mastering the Ms. Pac-Man game using deep reinforcement learning (DRL) is synergized with sophisticated optimization strategies. Central to the system is the DQN agent, leveraging a Q-Network—a convolutional neural network adept at discerning the most advantageous actions in the game scenario. The proposed approach is supported by the Replay Buffer, an integral feature that archives and re-examines previous gameplay, ensuring a robust and progressive learning journey. The technique is further refined by intertwining the Snake Optimization Algorithm (SOA) and Energy Valley Optimization (EVO), both inspired by genetic-based algorithms, for the optimization of critical hyperparameters, thereby enhancing the overall efficiency of the system. The SOA, inspired by the natural behaviors of snakes, particularly their feeding, fighting, and mating patterns, categorizes the population into males and females and simulates their complex survival strategies through dual-phase operation: exploration and exploitation phases. During the exploration phase, snakes search randomly for food, encouraging the exploration of diverse solutions and preventing premature convergence to suboptimal solutions. In the exploitation phase, when enough food is available, the search behavior becomes more directed, refining the solutions found during the exploration phase and guiding the search towards optimal solutions. The SOA uses temperature as a critical factor influencing the snakes' behavior, allowing for a balance between exploration and exploitation, making it suitable for optimizing hyperparameters in dynamic and complex environments like Ms. PacMan.

Meanwhile, EVO draws inspiration from particle physics, specifically the behavior of subatomic particles striving for stability, based on the concept of an "energy valley", representing the state in which particles are in their most stable form, bound by optimal levels of neutrons (N) and protons (Z). EVO mimics the natural tendency of particles to emit energy and transform into more stable forms by adjusting the N/Z ratio, thus moving particles towards their energy valley. This process involves evaluating the stability of each particle configuration and iteratively adjusting the parameters to achieve a more stable state, analogous to finding the optimal solution in a problem space. The integration of SOA and EVO into the ESO targets not only the hyperparameters but also the training loop, continuously updating the DQN agent parameters until the optimal ones are identified, as evidenced by rewards feedback from the environment. ESO begins with initializing a diverse population of potential solutions, each representing a unique set of hyperparameters. The performance of each solution is then evaluated based on the rewards obtained during gameplay, with the best-performing solutions selected for the next generation. The selected solutions undergo crossover and mutation to introduce variability and explore new potential solutions, ensuring a broad search across the hyperparameter space. The interplay of these elements culminates in a comprehensive and effective solution to the challenges posed by the Ms. PacMan gaming environment. Each component—from the Q-Network and Replay Buffer to the innovative ESO optimization—collaborates with the next to refine the strategy, as further elucidated in the subsequent sections.

4.1. Setup and Environment Preparation

Establishing a strong foundation is critical to the success of the Ms. PacMan project. The preliminary actions taken to establish a reliable and effective setup are described in this section. The installation of the required libraries and packages initiates the process. These resources not only enable the procedures to run smoothly but also provide essential utility functions required for the experiment's later phases. Utilizing the most recent versions of these tools ensures compatibility and optimizes efficiency.

Next, OpenAI's Gym toolbox put up the Ms. PacMan gaming setting. In the field of RL, Gym is well-known for its standardized interface, which makes it easier to interact with the game, process observations, and put actions into action. Such standardization is essential to make sure that experiments can be run over and over, with comparable results across a broader research context. We then describe several key parameters of the experiment, including the game space (all possible actions for Ms. PacMan), observation space (how we model an abstract state of a game), and reward structure. These parameters are essential, as they influence the learning process of a DQN agent and help it to learn a strategy and make decisions.

In summary, the setup and environment preparation phase serve as a background for the experiment. This ensures that the implementation of training the Q-Network and using the Snake Optimization Algorithm for hyperparameter tuning are built on top a sound foundation, thus leading to successful outcomes in future stages.

In Figure 1, we delineate the comprehensive setup of our proposed method. This illustration encapsulates the overall approach, detailing each critical component and the interactions between them. The figure is structured to provide a clear visual representation of the workflow and mechanisms that underpin our methodology, facilitating an understanding of its functionality and the sequential processes involved.

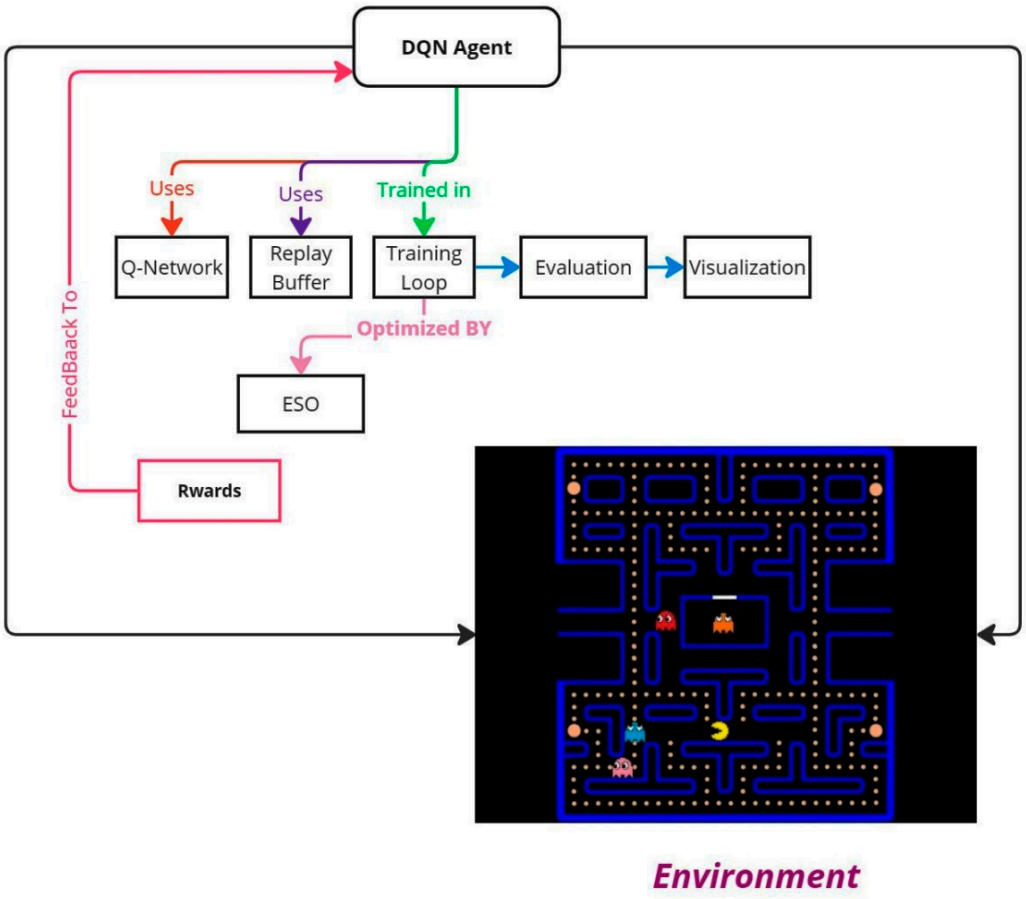

Figure 1. Setup and environment flowchart.

4.2. Design and Functionality of the Q-Network

At the core of the process by which we will master Ms. PacMan is the Q-Network, using a convolutional neural network intended to calculate Q-values for a number of game states. The Q-Network forms the meat of the RL process. Precise estimation of these Q-values, which are nothing but expected rewards concerning particular actions in given states, is what makes it possible to guide the agent toward activities on which benefits are maximal with respect to time.

Q-Network architecture is designed by taking into account the specific requirements of Atari-like games, as in the case of Ms. PacMan, together with the incorporation of convolutional layers for interpreting the pixel-based visual inputs. As the maze layout, the ghost positions, and the location of the pellets around which the game is played in Ms. PacMan are recognized and decoded into spatial relations and patterns, these layers develop their prowess. The visual information is then passed through other layers of the network before finally reaching a dense layer, in which Q-values are furnished for all of the agent's available actions.

The Q-network is optimized by a loss function, usually a mean squared loss, to quantify the difference between the predicted and targeted Q-value. Here, targeted Q-values go directly from the Bellman equation, connecting current Q-values to future rewards and maximal Q-values of subsequent states. That way, a recursive scheme helps in the estimation of long-term value with every action taken, guiding the agent's choices.

In this manner, the agent is trained dynamically and perpetually, gathering new experience of interacting with the game environment. Thus, the agent can make better decisions as it dynamically updates the network with such kinds of upgrades that allow for refinements in Q-value approximations. The experience replay allows the network to relearn from previously stored experience, hence breaking the correlation between consecutive experiences and ensuring the stability and robustness of learning.

Furthermore, adding to it, it stops the fluctuations in the learning process by using techniques like target networks and experience replay. Off-policy methods with target networks maintain the target values constant for a fixed duration to curb oscillations and divergence in Q-value estimation.

Under the iterative learning process, the Q-Network becomes progressively better at predicting Q-values with increasing round-offs, so the agent is enabled to make optimal decisions. This will be a lifelong, continuous process of adaptation for the agent to explore the game environment, attune itself to the new challenges, and boost the performance over time, with a well-designed training mechanism being essential for evolving a proficient RL agent.

The architecture of the Q-Network will be according to the requirements of Ms. PacMan. The network will consist of two main parts: a convolutional part and a fully connected (dense) part. The convolutional part consists of three layers. The first layer takes input from raw pixels and applies 32 filters of size 8×8, with a stride of 4. ReLU is then used for non-linearity. This is done to capture broad and important features at the early processing stages. The second layer uses 64 filters of size 4×4 with a stride of 2 in order to lay more emphasis on the fine-grained, detailed spatial patterns. The third layer uses 128 filters of size 3×3 with a stride of 1 in order to capture even finer details. All layers are followed by a ReLU activation, which introduces non-linearity and makes the network better at recognizing highly complex patterns.

The processed data are then flattened and directed into the dense part after the convolution stage. This contains a linear operation attaching to 512 neurons from the convolutional output, followed by ReLU activation for proper learning and non-linearity. A dropout of 0.5 probability was added to avoid overfitting and for implementing regularization for better generalization. Then, the information goes to 256 neurons, and finally, through a linear operation, the Q-values for each action that the agent can take in the game are obtained. These convolutional and dense layers are the building blocks of a complete architecture to be used for effective visual input processing and the identification of complex patterns

within the game and their translation into optimal game performance by selecting action strategies. The size of layers, dimensions of filters, and the choice of activation functions in the given architecture are chosen in a manner that the complexity related to Ms. PacMan's environment is taken care of, thereby assuring that the agent can navigate the surroundings and make strategic decisions effectively based on the visual input it receives.

4.3. Implementing Experience Replay for Stable Learning

Experience replay is a critical component in the RL strategy, particularly for addressing challenges like temporal correlations and the evolving nature of data in such environments. Unlike traditional methods that learn directly from consecutive experiences, which can lead to correlated data and unstable learning paths, experience replay stores individual experiences or transitions and revisits them randomly. These transitions consist of tuples containing the current state, the action taken, the resultant reward, the following state, and an indicator of whether the game concluded after the action. These tuples are stored in a Replay Buffer, a memory bank continuously filled as the agent interacts with the game.

The efficiency of the Replay Buffer is in its implementation details. Typically, the buffer is fixed at a certain size, e.g., 1 million transitions, so as to guarantee that vast but diverse experiences fill it and do not overly eat up the memory resources. If the old experience justifies itself, then it can be garbage collected in order to give space to the new ones, therefore balancing the quantity of new data with historical data.

The most typical adaptation of the strategies is uniform random sampling, where each experience is selected with the same probability. The randomness of this strategy will break the chain of closely correlated experiences, and even older valuable memories can also be part of the updating process, rendering the complete learning trajectory stable and effective.

In other words, the Replay Buffer allows for the repeated use of data for the learning process, enabling an agent to learn multiple times from the same experience. This will be particularly important in complex environments like that of Ms. PacMan, where rare but valuable experiences truly would count. An agent can then re-enter this experience, further realize the challenge of the game, and learn how to adapt even better. This basically instills randomness and diversity in the learning procedure, from the Replay Buffer that will be described later. By doing so, this method mitigates temporal correlation and diversifies the experience received by the agent with various past interactions. Together, it outlines the paradigm to be mastered in DRL with Ms. PacMan.

4.4. Role and Functions of the DQN Agent

The DQN agent is pivotal in orchestrating the complex interactions between the agent and the Ms. PacMan game environment. It carries the dual responsibility of deciding the agent's actions and training the Q-Network based on the outcomes of these actions. Action selection within the DQN agent operates on a principle balancing exploration and exploitation. Initially, when the agent's knowledge of the environment is limited, and its Q-values are unrefined, the agent emphasizes exploration. This is typically managed through an epsilon-greedy strategy. The agent randomly selects actions with a probability defined by epsilon to explore the environment and relies on the highest Q-value actions for the remaining decisions, exploiting its current knowledge. As the agent's familiarity with the environment improves, and the reliability of its Q-values increases, epsilon is progressively reduced, tilting the balance towards exploitation.

Following action execution, the DQN agent plays a crucial role in training the Q-Network. Through the Replay Buffer, it samples a random batch of experiences and computes target Q-values based on the rewards obtained as well as projected Q-values of future states. The aim of the DQN algorithm is to bring these target Q-values closer to what we measure from our Bellman equation estimates (as predicted by our Q-Network)—usually via gradient descent or some variant thereof. In theory, as the model iteratively chooses

actions, accumulates experience, and updates the network in an endless loop, this learning cycle will progressively improve both the strategies and game understanding of the agent.

Target Q-values are also updated periodically. Instead of updating them continuously and risking instability, these updates are spread out, which facilitates a more gradual path to learning that is consistent and stable.

To sum up, the DQN agent is arguably one of the most important parts, as it provides a connection between exploration and learning. It maneuvers its way through the Ms. PacMan world, learning from its success and failures to evolve strategies and further tune the Q-Network. Depending on what actions the agent takes and what feedback it receives, it will then learn from that to do better than the old version at playing in this environment.

4.5. Execution and Maintenance of the Training Loop

The latter describes the training loop—the dynamic phase in which the RL strategy is run. Over a span of episodes in this iterative process, the DQN agent interacts with the Ms. PacMan game environment and goes through making decisions, receiving feedback on those decisions, storing experiences related to taken actions and rewards for them into Replay Memory, and improving its strategy using these experiences by updating the Q-Network in batch from a sample of stored experience tuples sampled out from a uniform distribution over all memory buffer. This section deep dives into the intricacies of the training loop and how it works.

This loop includes many episodes, with one episode per complete game of Ms. PacMan from start to finish. At the start of each episode, the game state is reset, and a new game begins for the agent. In the course of an episode, this agent observes its new state, takes an action decided on a relaxed form of its epsilon-greedy policy that is normalized at evaluation time without guidance loss contributions from moves already recommended but not yet taken, as enabled by moves enacting deferred reinforcement, and then finally makes that (deferred) recommendation in the game. The game environment then shifts to the next state and gives a reward, depending upon how well the action performed. These include the state transition, the action chosen, the reward received, and the game end status, which are saved as an experience in the Replay Buffer.

When enough experiences are collected, the agent trains the Q-Network using a batch selected randomly from the Replay Buffer. This training works by adjusting the network's weights to minimize the difference between the predicted Q-values and the target Q-values. This feedback-driven method ensures the agent fine-tunes its strategy over time to deliver better performance in an incremental fashion.

One important part of the training loop is that we save the weights for our Q-Network. These weights may generate countless iterations over the life of the agent, so they should be saved periodically. As checkpoints that are saved, these serve a variety of different purposes. The first allows for failures (if things you depend on go down, you still make progress). Second, it permits relatively quick evaluations of the agent and thus provides potential glimpses into its learning progression. Otherwise, saved weights can be used for learning transfer to another similar task or fine-tuning on new ones.

Fundamentally, the training loop is also where the behavioral response of our agent to this gradual learning process manifests as in gameplay strategy. This process continues over a number of iterations, as the agent moves from being a novice to an expert Ms. PacMan player through reinforced learning, all the while making sure that the lessons learned are recorded and stored for later use in further evaluation.

4.6. Performance Evaluation of the Agent

An important step after the intensive training loop is evaluating how well our agent performs. This evaluation checks not only the capability of an agent within the Ms. PacMan environment, but also potential weaknesses. The agent's responsiveness is evaluated with respect to the same attributes against which it has been measured.

The first step in this evaluation is to switch the agent to a purely exploitative mode, which involves disabling the random action selection feature (i.e., setting epsilon to zero in the epsilon-greedy policy). This mode forces the agent to solely depend on its acquired knowledge, choosing actions based entirely on the Q-values provided by the Q-Network. This approach offers an accurate representation of the agent's learning and its application skills.

In this phase, the agent engages in a set number of episodes, mirroring the training process, but with two key distinctions. First, there is no learning or adjustment of the Q-Network's weights based on the agent's actions. Second, detailed records are kept of every action, state transition, and reward received. The primary indicator of the agent's performance is the total cumulative reward accumulated in each episode.

Nevertheless, relying on a single metric might not fully encapsulate the agent's abilities. As such, alternative metrics are taken into account. These might be how many levels were completed, the average amount of ghosts eaten per episode, or the number of times bonus fruit was caught. Additional metrics will provide a more detailed picture of the agent's strategic gameplay. Since this is a repetitive game with randomness and stochasticity in ghost behavior, as well as different fruit appearances on the screen, it is important to check an agent's performance across multiple episodes. In other words, these averages form the true hypothetical benchmark, which also wipes out all random fluctuations and represents a measure of true agent capabilities.

The evaluative stage also uses visual aids to help with the quantitative analysis. Heatmaps of the agent's most commonly taken paths, or plots of the trajectory of accumulate rewards over episodes, can offer some key visual cues about strategic decisions made by an agent.

4.7. Visualizing the Agent's Gameplay

Seeing the model in action helps to demonstrate how well the agent performs at playing through Ms. PacMan using RL. Like watching a human player play in a maze, this visualization provides an interesting and comprehensive story of the agent's strategies, the challenges it met, and potential improvements. The agent tells a tale through its interactions with the Ms. PacMan surroundings, going beyond simple labyrinth navigation to include learned habits, threat assessments, and pivotal decision-making moments. Thorough analysis and comprehension of the agent's path are facilitated through this visual representation.

For this purpose, a simulated screen was developed to record the actual gameplay in real time by an agent—the whole enchilada, including every step, near-death experience, and power pellet scooped up. This visual catalogue serves as a convenient, intuitive means of verification with which to start off. This allows researchers, developers, fans, and other stakeholders to see the results of the training process in a dynamic way. It provides crucial insight into an agent's navigation strategy: How well can it squeeze through narrow gaps? Is it strategic in its use of power pellets to chase ghosts, or is getting the maze completely empty what is most important? How will it react when, all of a sudden, the bonus fruits appear?

This visual representation also helps in debugging and tuning the agent. Oddities or trends in bad decision making, which would not be quite as visible in data form, can now easily be identified through this succinct version. This ultimately means faster and more flexible changes.

Conclusion: Last but not least, these actions are visually recorded, which has wide outreach. The actions can be shared more broadly within our community—shown in presentations or used in an education context to help illustrate DRL principles and practices.

Ultimately, this shows us that video visualizations of gameplay are not just for making our (more DRL) agents look less black-boxy, but for providing a critical diagnostic tool to debug and refine agents as well as help educate users about the used system. Another example would be a good level of demonstration of the actions provided by the agent,

providing useful feedback and showing how DRL is well generalized in environment such as Ms. PacMan.

4.8. Proposed Optimization Algorithm for Hyperparameter Tuning

In the proposed framework, ES is a fusion of energy strength in SOA and tolerance ability of the energy valley in EVO. To accomplish this, we aim to utilize the powers of these algorithms by fine-tuning the hyperparameters that are at heart of where the DQN agent performs, namely, the learning rate (lr) and the discount factor (γ), which affects the foresight and learning performance of our DQN agent.

Snake Optimization a relatively new intelligent optimization algorithm. It was proposed by Hashim et al., based on the behavior of snakes, especially the models of feeding, fighting, and mating. What sets the algorithm apart from other metaheuristic algorithms is the simulation of the intricate survival strategies of snakes. In its working, firstly, the SO characterizes the population into males and females. Secondly, it starts with random populations, and lastly, in their feeding and mating behavior, it characterizes the influence of temperature, given the importance of temperature for cold-blooded animals like snakes. Snakes operate in two phases. The first is the exploration phase, which means that there is no adequate food in the environment; in this case, the snakes forage randomly for food. When food availability is sufficient, the exploitation phase is defined by the behavior of snakes, thus controlling the search behavior of the snakes. These two phases of exploration and exploitation are given mathematical representations—specific equations for the positions of males and females for each of the phases. Several modes and mechanisms, such as fight and mating modes, are also characterized; however, these mechanisms are triggered by the environment, more specifically temperature, thus making the algorithm much more complicated and with a higher degree of optimization.

EVO is based on the principle of particle physics, mainly the behaviors of subatomic particles. It is based on the principle of stability and decay of the particle. In the universe, most particles are unstable, tending to emit energy and transform into more stable forms. EVO is based on the concept of an "energy valley"—a metaphorical state where particles are in their most stable form, bound by optimum levels of neutrons (N) and protons (Z). In this state, particles try to increase their stability through adjusting their N/Z ratio, approaching this energy valley or stability band. The concept is more fundamental to the stability of heavier particles, which demand a higher N/Z ratio for their stability. The idea is that Energy Valley Optimization mimics the natural tendencies of particles, using the idea of stability and transition to guide searches to optimal solutions within a problem space. This is a novel algorithm that employs the essential features of particle physics in algorithmic optimization.

The algorithm kicks off when the SOA initializes a population of "snakes", each quoting a single set of hyperparameters. These snake transverse a metaphorical landscape akin to the DQN agent' s performance under diverse hyperparameter configurations. At the same time, the EVO inserts a separate population that also undergoes an evaluation, whereby these individuals are assessed in relation to the performance of handling the character of the game. The next stage now combines the SOA and EVO methodologies, in that the best-performing snakes from the SOA are combined with the top individuals of the EVO population. This, therefore, then allows the best hyperparameters to take jumps to the other population; hence, crossovers of more robust hyperparameters offer offspring that could be better than the predecessors. This is applied over each of the populations through mutation, hence rendering them variable and permitting a wide search over hyperparameter space. As it progresses across generations, SOA and EVO proceed together to obtain the best set of hyperparameters that are promising in the Ms. PacMan environment for good performance. Then, again, this process continues until some kind of convergence criterion is satisfied, or until a set number of generations has been reached. The DQN agent is configured with the obtained optimized hyperparameters

from these two algorithms. The agent is allowed to play the game appropriately, and its performance is tracked and optimized across many episodes.

To validate the effectiveness of the ESO-tuned hyperparameters, an extensive evaluation is conducted. This not only involves a quantitative assessment of the rewards but also includes a qualitative analysis through visualized gameplay, offering insights into the agent's decision making and strategic gameplay. The detailed steps are shown in Algorithm 1.

Algorithm 1 Energy Serpent Optimizer (ESO)

Input: Environment env, Population size N, Number of generations G
Output: Best hyperparameters: lr*, γ*
Initialize population of N serpents with random lr and γ **for** each serpent **for** generation
- i to G **do**

This is the start of the main loop
Evaluate fitness using EVALUATE(env, lr, γ)
Sort serpents by fitness in descending order for i = 1 to N/2 do
- This is the start of the breeding loop

Select two parents p1, p2 randomly from top serpents
child ← CROSSOVER(p1, p2)
MUTATE(child)
Evaluate fitness of child in env
Replace least-fit serpent with child if child's fitness is higher
- This is the end of the breeding loop

Adapt mutation rates if necessary based on performance trends
- This is the end of the main loop

best_serpent ← serpent with highest fitness
lr* ← best_serpent.lr
γ* ← best_serpent.gamma
return lr*, γ*

The ESO, shown in Figure 2 pseudocode, is an evolutionary algorithm designed to optimize the learning rate and discount factor in RL models. First, it creates a virtual environment, where there is a population of serpents, and every serpent is regarded in a different way as a unique set of hyperparameters. While running from one generation to the next, a serpent is evaluated for his hyperparameters in their performance in the set environment, considering that the performance is measured according to the rewards accumulated by the agent. A serpent is then ranked and selected for fitness, meaning that the ones who have the best configuration of hyperparameters are considered to be fit. Such best serpents go through a breeding procedure, being submitted to basic genetic operations composed of crossover and mutation. Crossover is when hyperparameters of two basic parent serpents are merged to form offspring. Mutation introduces random changes into these offspring in such a way that diversity is visible, helping to explore the hyperparameters available.

The evaluation, selection, breeding, and mutation process take place over a range of generations that help in fine-tuning the hyperparameters. The process then tries to replace less-fit serpents with offspring that have promising results and continues to push forward the whole population by iterating the process until optimal hyperparameters are achieved. The algorithm tries to replace the less-fit serpents with more promising offspring, gradually pushing the whole population forward in searching for optimal hyperparameter combinations. This process gives a new refined process into the training of RL agents, making them ready to work with the maximum efficiency in complex decision-making environments.

In summary, the combined iterative ESO approach for hyperparameter tuning is a dynamic search-and-exploratory process that strategically shifts all-important parameters

critical to the learning process to key in very well. This synergy will level up the DQN agent with regard to strategy, making it more intelligent or proficient in dealing with the Ms. PacMan labyrinth itself, as illustrated in Figure 3.

Figure 2. Reward Over Iterations score of ESO.

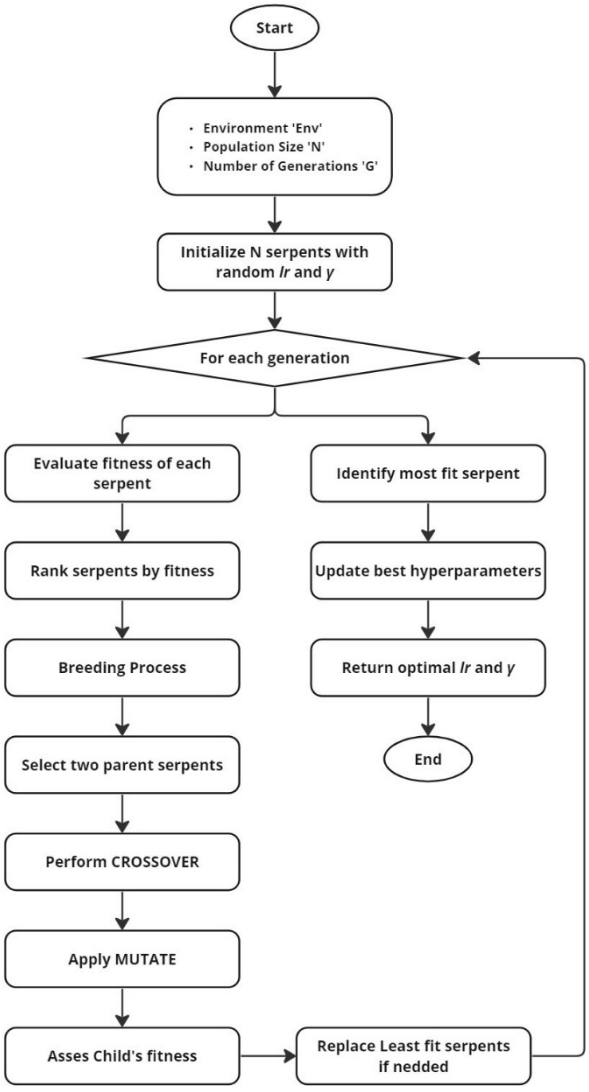

Figure 3. Energy Serpent Optimizer flowchart.

5. Results

Outcomes of the ESO Algorithm

ESO: Game-playing interface: A very complex maze-based idea was developed in this game environment through which the API had to navigate. Maze-based environment: The game board, which was classified by the large number of blue passageways, was designed within various themes, which included the characters in movement, the snake emblem, and the skulls. The main goal of the AI agent in the game was to reach the other side of the maze as quickly as possible, to avoid danger, and to gain points, interacting with some icons in the maze.

The performance of three different optimization algorithms is meant to be shown below, namely, the Snake Optimizer Algorithm (SOA), the Energy Valley Optimizer (EVO), and the Energy Serpent Optimize (ESO), by depicting the reward achieved at the end of each iteration. From the information gathered across the eight iterations, it is easily noticeable that from the beginning of the first iteration, the ESO leads the race by a vast difference, with a reward of 400.0, whereas its counterparts, the SOA and the EVO, have returned rewards of 200.0 and 150.0, respectively. Meanwhile, the reward for the three algorithms increases over the course of the different iterations, indicating that there is learning and updating of the policy, so that the algorithms become capable of selecting better actions.

Figure 2 in the document presents a graph titled "Rewards Over Iterations", comparing the performance of the three optimization algorithms—Snake Optimizer Algorithm (SOA), Energy Valley Optimizer (EVO), and Energy Serpent Optimize (ESO)—across eight iterations. The graph shows the rewards achieved by each algorithm at the end of each iteration. Initially, ESO starts with a significantly higher reward of 400.0, whereas SOA and EVO begin with rewards of 200.0 and 150.0, respectively. As the iterations progress, all three algorithms demonstrate increasing trends in their rewards, reflecting their improvements and learning capabilities in selecting better actions. ESO maintains its lead throughout, reaching nearly 1000 by the eighth iteration.

The other two improve in the second iteration, as illustrate Table 1, with the returns being 350.0 for the SOA and 250.0 for the EVO, but the ESO still leads at 550.0. From this point on, the improvement of the latter two has kept consistent, and both the SOA and the EVO keep growing. But even then, the ESO shows the greatest improvement, as it does the entire time, and returns with the best performance. In the fourth generation, the rewards are up to 750.0 for ESO, 450.0 for SOA, and 350.0 for EVO. Such a huge difference in performance is a measure of how skillful the ESO is in identifying and capitalizing on high-reward strategies in a quickly emergent task.

Table 1. Iterative reward comparison: showcases ESO's superior optimization efficiency over SOA and EVO.

Iteration	SOA Reward	EVO Reward	ESO Reward
1	200.0	150.0	400.0
2	350.0	250.0	550.0
3	400.0	300.0	650.0
4	450.0	350.0	750.0
5	500.0	400.0	850.0
6	600.0	450.0	950.0
7	700.0	500.0	1050.0
8	800.0	550.0	1100.0

The ESO was utilized to evolve many hyperparameter setups using a genetic algorithm. Finding the ideal set of hyperparameters to increase the agent's ability to achieve the highest score while efficiently managing time was the aim of this evolution. It needed 32 s to finish the ESO process, involving multiple revisions and iterations. With a noteworthy score of 1100.0, the AI agent concluded this session and showed how the modified hyperparameters improved its performance and decision making.

Figure 4 is a cumulative illustration of the agent's gaming at different points in time. The figure details the course of the agent through the maze and the strategies and pitfalls it faces. A single look at the illustration will depict the decision-making process of the agent, the diversity of obstacles that the agent must surmount, and the remarkable inroads made in the relentless march toward peak performance. This is a tracking of the path of the agent, thus ensuring complete visibility of the dynamics within the game.

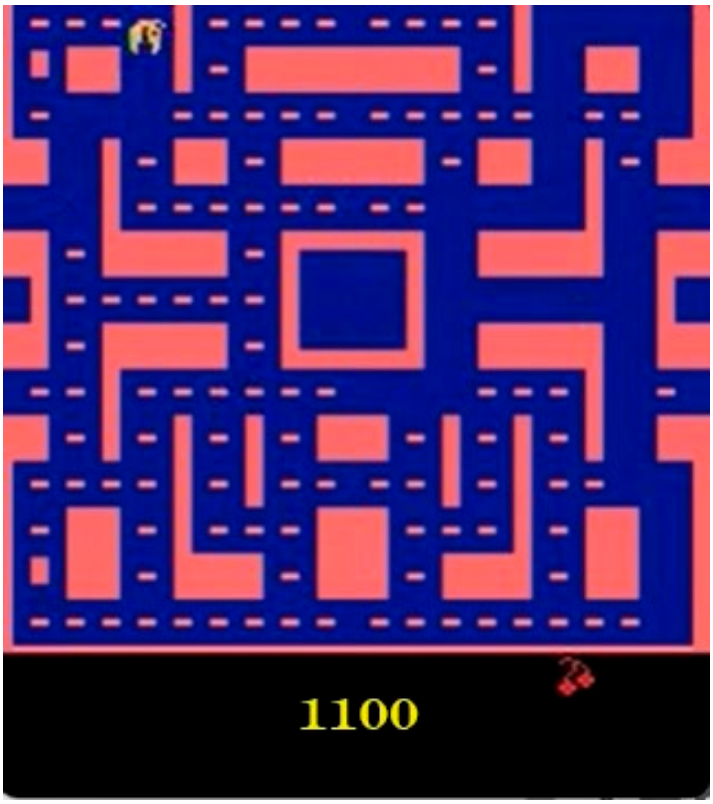

Figure 4. Reward score of ESO.

This visual aid will provide better understanding as far as this comparative study of the EVO and the algorithm (SOA) is seen in terms of rewards collected through its operational paradigm, respectively. In this regard, the SOA, inspired by the serpent's foraging for the behavioral study of optimization, has a performance index of 840.0 in the reward through its operational cycle. This is quite good and shows its excellent searching and optimization within its solution space, possibly due to its state of balance between the phases of exploration and exploitation.

On the other hand, the EVO, inspired by the minimization of potential energy to move in a search space, performs up to an accumulated reward of 640.0. Although lower when compared to the SOA reward, the figure still presents a very important plane of effectiveness in the search for an optimal solution. The possible reason for there being such disparity in the rewards between these two algorithms could be seen in the inherent differences in search heuristics and, in turn, convergence properties between the two.

Figure 5 displays a comparative analysis of the three algorithms—Snake Optimizer Algorithm (SOA), Energy Valley Optimizer (EVO), and Energy Serpent Optimize (ESO)—using a bar graph. Each algorithm is represented by two bars, one for the reward achieved and another for the time taken in seconds. SOA achieves a reward of 840.0 and takes 45 s, EVO attains a reward of 640.0 with a time of 50 s, and ESO leads with a reward of 1100.0 in just 32 s. This visualization underscores ESO's superior efficiency and effectiveness in both reward maximization and computational speed.

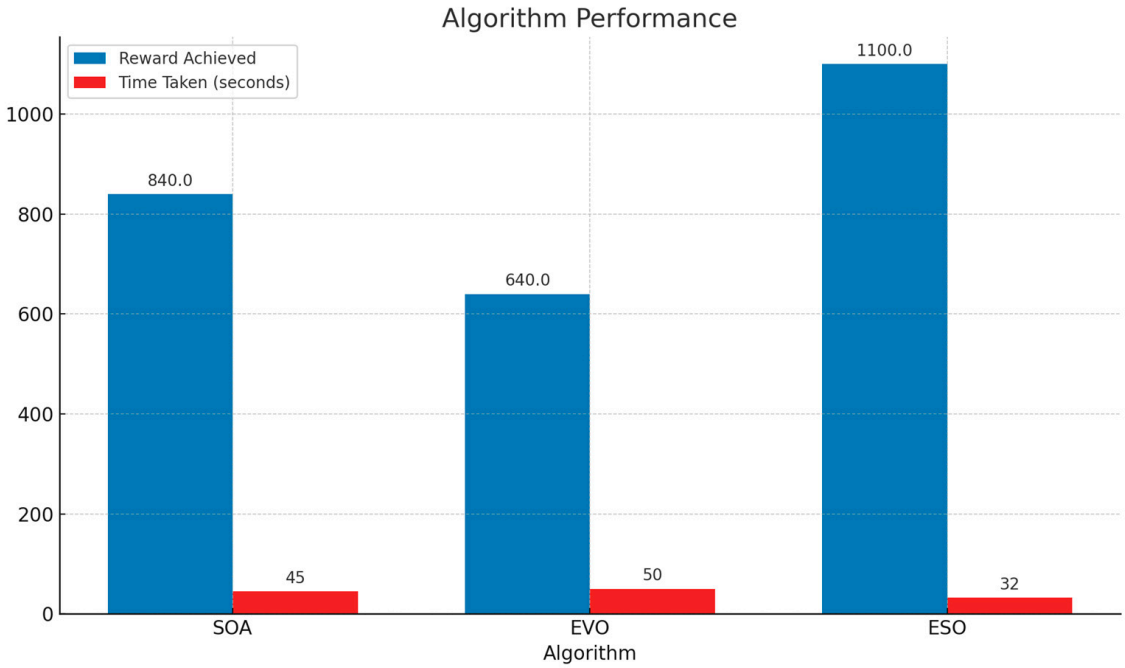

Figure 5. Algorithm performance: reward achieved and time taken.

In the domain of algorithmic optimization, the comparative analysis of the (SOA), (EVO), and ESO algorithms is strikingly telling, as Table 2 illustrates. The SOA, though robust in its mimicry of natural foraging strategies, achieved a moderate reward of 840.0 but required a relatively prolonged duration of 45 s to converge upon a solution. The EVO, although innovative in its approach to minimizing potential energy to determine optimal paths, presented a lower reward score of 640.0 and even surpassed the SOA in time consumption, with a completion time of 50 s. These figures suggest a less efficient optimization process in scenarios in which quick convergence is paramount.

Table 2. Comparative performance of SOA, EVO, and ESO: reward scores and computational times.

Algorithm	Reward Achieved	Time Taken (s)
SOA	840.0	45
EVO	640.0	50
ESO	1100.0	32

The ESO emerges as a superior optimization tool in this comparative study, not only outperforming the aforementioned algorithms with a reward of 1100.0 but also demonstrating expedited computational efficiency by completing its process in a mere 32 s. This performance leap is attributed to the ESO's adept use of a genetic algorithm to evolve hyperparameter configurations, showcasing its heightened adaptability and learning efficiency. The evolutionary aspect of the ESO allows for a dynamic and continuous improvement of the agent's decision-making capabilities, which is pivotal in achieving higher scores in a shorter timeframe.

The limitations of the SOA and EVO, though apparent in this context, may stem from their less dynamic adaptation mechanisms. These algorithms, despite their potential, fall short when tasked with rapidly evolving environments or problems that demand quick iterative improvements. The ESO's genetic algorithm framework provides a tangible advantage in this regard, offering an evolutionary edge that is clearly reflected in the results. Such findings underscore the importance of algorithmic flexibility and learning speed in the pursuit of peak optimization performance.

6. Discussion and Implications

The realized ESO with hybrid fitness in a rich maze environment-based complex game provides intensive information regarding adaptive artificial intelligence strategy, role, and impact in a dynamic setting. The complex pathways in the maze, along with the rest of the game elements, make a platform for the AI agent to fine-tune its decision-making process in a very tight situation that entails time pressure and the different hazards scattered in its path.

The success of the ESO in evolving the hyperparameters can be observed from the resultant ability of the agent to harvest a considerable score within the set time boundaries. This in turn validates not only the efficiency of the genetic algorithm-based approach but vindicates the importance of selecting and updating the correct set of hyperparameters. In detail, the learning rate and discount factor play a vital role in defining the learning curve of the agent and mastering the art of maximizing rewards and minimizing risks.

Moreover, the agent's journey throughout the maze can be analyzed in terms of behavioral patterns and strategic change, as illustrated in Figure 3. This gives a granular impression of the agent's in-game interactions in terms of taking different approaches to avoid obstacles and optimize the path. The relevance of this visualization can be put into perspective in two very important ways: one with respect to showing why the approach is being taken with respect to the agent, and secondly, giving room for the improvement of approaches with respect to indicating possible areas for further optimization.

Furthermore, the score obtained by the agent, 1100.0, is a measure benchmark, which can be further analyzed in terms of a comparison with some other RL models or optimization techniques. This performance metric is a strong outcome of the evolutionary process, giving an empirical way to compare different hyperparameter configurations.

The implications of findings go much beyond the confines of the gaming arena. The tools and methods used here can be extrapolated to a host of real-life applications in which autonomous decision making is vital, like in robotics, autonomous vehicles, and management of complex systems. A salient feature of state-of-art artificial intelligence systems is that they can evolve and adapt in changing environments, and ESO is exemplary for this feature.

In conclusion, the application of ESO to a maze-based game environment has been very informative about the traits of AI agents operating in complex and variable settings. It underlines the adaptations that are ongoing and general signs of intelligent action, prerequisites for advanced AI systems operating in the manifold challenges of both the virtual and real environments.

Eliciting the individual capacities of both SOA and EVO in the same complex maze-based game environment will further help us to understand adaptive optimization strategies in dynamic and constrained environments. In its biological inspiration, SOA tries to maintain a strategic balance between exploration and exploitation, similar to the static foraging behavior of snakes. This approach allows the AI to find a way across the maze through the intuitive understanding of the space but might be suboptimal in decision making should the environment have higher complexity or sharp changes in game dynamics.

EVO, however, uses energy level minimization to find the proper paths and shapes itself as a promising approach for the maze navigation process by continuously decreasing the potential energy levels in a methodical way. However, its performance can be compromised by the sometimes overly deterministic nature that does not always allow for perfect accommodation of the stochastic nature of dynamic obstacles or changing game conditions.

Although both SOA and EVO reveal appreciable potential in maze exploration and interaction with its elements, they, too, have some limitations that are detrimental to their overall merits in guaranteeing high scores within the allocated time frames. Quite simply, the heuristic balance on which SOA stands will not always lead to an optimal path, especially in mazes with patterns that are highly irregular and unpredictable. Also, the deterministic nature of EVO may compromise its flexibility in rapidly changing or highly stochastic environments, consequently leading to suboptimal strategies.

On the other hand, however, ESO is a more robust and adaptive solution that encompasses the strengths of both SOA and EVO, while at the same time overcoming their weaknesses by the implementation of a genetic algorithm. With this capability, the ESO can evolve its hyperparameters and enhance its processes towards decision making by the AI agent in an adaptive and responsive manner towards complex challenges. The ESO has proven its performance by attaining a score of 1100.0 in the maze-based game, and such superiority has been based on its better adaptive learning rate, adjustment of the discount factor, and more general strategic flexibility. Having learned from the failures of SOA and EVO, the ESO was better equipped to navigate the maze with increased efficiency and effectiveness, thus showing the applicability of such hybridized and evolved optimization strategies in complex and dynamic environments.

Thus, with such a value in SOA and EVO for insights into the optimization capability of constrained settings, ESO is a quite unique integrated adaptive approach, offering excellent solutions to very advanced AI systems, using rapid adaptation and learning in contexts of unpredictability and diversity.

7. Conclusions and Future Work

In this paper, an innovative approach has been presented that harnesses the combined power of the Snake Optimization Algorithm and Energy Valley Optimization (ESO) to elevate the performance of RL agents in complex environments. The exploration within the challenging confines of a maze-based game has not only validated the robustness of the RL model but has also underscored the novelty of the ESO combination in effectively tuning hyperparameters to optimize the agent's learning and decision-making abilities. The environment presented to the agents was rich and dynamic, providing a rigorous testing ground for the algorithms to perform and evolve.

The novelty of the approach lies in its adaptive complexity and the nuanced understanding of the agent's interactions within the game, which is made possible by the detailed analysis of the agent's performance and the strategic evolution of hyperparameters. The ESO framework demonstrates a significant advancement in the field by allowing for a more nuanced and fine-tuned approach to the evolution of the learning parameters.

For future work, the aim is to transcend the domain of games and apply the findings to more intricate and practical applications, such as autonomous systems, logistics, and complex problem-solving scenarios, where AI agents must make decisions in unpredictable and multi-faceted environments. It is anticipated that addressing observed limitations, such as the need for extensive computational resources and the potential for overfitting in specific types of environments, will be crucial in advancing the ESO framework. Further exploration aims to focus on the scalability of the approach, its applicability to multi-agent systems, and its effectiveness in multi-objective optimization scenarios. This endeavor hopes to pave the way for RL models that are not only efficient and robust but also versatile in their applications in the ever-growing challenges faced in the realm of artificial intelligence.

Author Contributions: Conceptualization, S.M.K.S. and H.K.; methodology, S.M.K.S.; software, S.M.K.S.; validation, S.M.K.S., H.K.; formal analysis, S.M.K.S.; investigation, S.M.K.S.; resources, S.M.K.S.; data curation, S.M.K.S.; writing—original draft preparation, S.M.K.S.; writing—review and editing, H.K.; visualization, S.M.K.S.; supervision, H.K.; project administration, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable. No human subjects were involved in the study.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kadhim, A.I. Survey on supervised machine learning techniques for automatic text classification. *Artif. Intell. Rev.* **2019**, *52*, 273–292. [CrossRef]
2. Usama, M.; Qadir, J.; Raza, A.; Arif, H.; Yau, K.-L.A.; Elkhatib, Y.; Hussain, A.; Al-Fuqaha, A. Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. *IEEE Access* **2019**, *7*, 65579–65615. [CrossRef]
3. Singh, B.; Kumar, R.; Singh, V.P. Reinforcement learning in robotic applications: A comprehensive survey. *Artif. Intell. Rev.* **2021**, *55*, 945–990. [CrossRef]
4. Rao, S.; Verma, A.K.; Bhatia, T. A review on social spam detection: Challenges, open issues, and future directions. *Expert Syst. Appl.* **2021**, *186*, 115742. [CrossRef]
5. Zaidi, S.S.A.; Ansari, M.S.; Aslam, A.; Kanwal, N.; Asghar, M.; Lee, B. A survey of modern deep learning based object detection models. *Digit. Signal Process.* **2022**, *126*, 103514. [CrossRef]
6. Bochenek, B.; Ustrnul, Z. Machine Learning in Weather Prediction and Climate Analyses—Applications and Perspectives. *Atmosphere* **2022**, *13*, 180. [CrossRef]
7. Keerthana, S.; Santhi, B. Survey on Applications of Electronic Nose. *J. Comput. Sci.* **2020**, *16*, 314–320. [CrossRef]
8. Razzaghi, P.; Tabrizian, A.; Guo, W.; Chen, S.; Taye, A.; Thompson, E.; Bregeon, A.; Baheri, A.; Wei, P. A survey on reinforcement learning in aviation applications. *arXiv* **2022**, arXiv:2211.02147. [CrossRef]
9. Sivamayil, K.; Rajasekar, E.; Aljafari, B.; Nikolovski, S.; Vairavasundaram, S.; Vairavasundaram, I. A Systematic Study on Reinforcement Learning Based Applications. *Energies* **2023**, *16*, 1512. [CrossRef]
10. Gronauer, S.; Diepold, K. Multi-agent deep reinforcement learning: A survey. *Artif. Intell. Rev.* **2021**, *55*, 895–943. [CrossRef]
11. van Eck, N.J.; van Wezel, M. Application of reinforcement learning to the game of othello. *Comput. Oper. Res.* **2008**, *35*, 1999–2017. [CrossRef]
12. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
13. Rajendran, D.; Santhanam, P. WITHDRAWN: Towards digital game-based learning content with multi-objective reinforcement learning. *Mater. Today Proc.* **2021**. [CrossRef]
14. Liu, S.; Cao, J.; Wang, Y.; Chen, W.; Liu, Y. Self-play reinforcement learning with comprehensive critic in computer games. *Neurocomputing* **2021**, *449*, 207–213. [CrossRef]
15. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
16. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general rein- forcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
17. Sarkhi, S.; Koyuncu, H. Optimization Strategies for Atari Game Environments: Integrating Snake Optimization Algorithm and Energy Valley Optimization in Reinforcement Learning Models. *Preprints* **2024**, 2024051262. [CrossRef]
18. Gong, X.; Yu, J.; Lü, S.; Lu, H. Actor-critic with familiarity-based trajectory experience replay. *Inf. Sci.* **2022**, *582*, 633–647. [CrossRef]
19. Such, F.P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K.O.; Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv* **2017**, arXiv:1712.06567.
20. Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q–learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
21. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with large scale deep reinforcement learning. *arXiv* **2019**, arXiv:1912.06680.
22. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *arXiv* **2015**, arXiv:1506.03134.
23. Lee, J.; Lee, Y.; Kim, J.; Kosiorek, A.; Choi, S.; Teh, Y.W. Set transformer: A framework for attention-based permutation-invariant neural networks. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
24. Ammanabrolu, P.; Riedl, M. Playing text–adventure games with graph–based deep reinforcement learning. *arXiv* **2018**, arXiv:1812.01628.
25. Adolphs, L.; Hofmann, T. LeDeepChef Deep Reinforcement Learning Agent for Families of Text-Based Games. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 7342–7349. [CrossRef]
26. Brown, N.; Bakhtin, A.; Lerer, A.; Gong, Q. Combining deep reinforcement learning and search for imperfect–information games. *arXiv* **2020**, arXiv:2007.13544.
27. Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; et al. Mastering Complex Control in MOBA Games with Deep Reinforcement Learning. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 6672–6679. [CrossRef]
28. Tsitsiklis, J.; Van Roy, B. An analysis of temporal-difference learning with function approximation. *IEEE Trans. Autom. Control* **1997**, *42*, 674–690. [CrossRef]
29. Afsar, M.M.; Crump, T.; Far, B. Reinforcement Learning based Recommender Systems: A Survey. *ACM Comput. Surv.* **2022**, *55*, 1–38. [CrossRef]
30. Adams, S.; Cody, T.; Beling, P.A. A survey of inverse reinforcement learning. *Artif. Intell. Rev.* **2022**, *55*, 4307–4346. [CrossRef]

31. Núñez-Molina, C.; Fernández-Olivares, J.; Pérez, R. Learning to Select Goals in Automated Planning with Deep-Q Learning. *Expert Syst. Appl.* **2022**, *202*, 117265. [CrossRef]
32. Uc-Cetina, V.; Navarro-Guerrero, N.; Martin-Gonzalez, A.; Weber, C.; Wermter, S. Survey on reinforcement learning for language processing. *Artif. Intell. Rev.* **2022**, *56*, 1543–1575. [CrossRef]
33. Alomari, A.; Idris, N.; Sabri, A.Q.M.; Alsmadi, I. Deep reinforcement and transfer learning for abstractive text summarization: A review. *Comput. Speech Lang.* **2022**, *71*, 101276. [CrossRef]
34. Kanashiro, L.; Caio, F.; Paiva, L.; Vita, C.; De Yathie, E.; Helena, A.; Costa, R.; Del-Moral-Hernandez, E.; Brandimarte, P. Outperforming Algorithmic Trading Reinforcement Learning Systems: A Supervised Approach to the Cryptocurrency Market. *Expert Syst. Appl.* **2022**, *202*, 117259.
35. Serrano, W. Deep Reinforcement Learning with the Random Neural Network. *Eng. Appl. Artif. Intell.* **2022**, *110*, 104751. [CrossRef]
36. Shavandi, A.; Khedmati, M. A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets. *Expert Syst. Appl.* **2022**, *208*, 118124. [CrossRef]
37. Wang, Y.; Jia, Y.; Tian, Y.; Xiao, J. Deep reinforcement learning with the confusion-matrix-based dynamic reward function for customer credit scoring. *Expert Syst. Appl.* **2022**, *200*, 117013. [CrossRef]
38. Lian, B.; Xue, W.; Lewis, F.L.; Chai, T. Inverse Reinforcement Learning for Multiplayer Non-cooperative Apprentice Games. *Automatica* **2022**, *145*, 110524. [CrossRef]
39. Lian, B.; Donge, V.S.; Member, G.S.; Lewis, F.L.; Fellow, L.; Chai, T.; Fellow, L.; Davoudi, A.; Member, S. Data-Driven Inverse Reinforcement Learning Control for Linear Multiplayer Games. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *35*, 2028–2041. [CrossRef]