sensors

MDPI

Article

Playing Flappy Bird Based on Motion Recognition Using a Transformer Model and LIDAR Sensor

Iveta Dirgová Luptáková, Martin Kubovčík * and Jiří Pospíchal *

Institute of Computer Technologies and Informatics, Faculty of Natural Sciences, University of Ss. Cyril and Methodius, J. Herdu 2, 917 01 Trnava, Slovakia; iveta.dirgova.luptakova@ucm.sk
* Correspondence: martin.kubovcik@ucm.sk (M.K.); jiri.pospichal@ucm.sk (J.P.)

Abstract: A transformer neural network is employed in the present study to predict Q-values in a simulated environment using reinforcement learning techniques. The goal is to teach an agent to navigate and excel in the Flappy Bird game, which became a popular model for control in machine learning approaches. Unlike most top existing approaches that use the game's rendered image as input, our main contribution lies in using sensory input from LIDAR, which is represented by the ray casting method. Specifically, we focus on understanding the temporal context of measurements from a ray casting perspective and optimizing potentially risky behavior by considering the degree of the approach to objects identified as obstacles. The agent learned to use the measurements from ray casting to avoid collisions with obstacles. Our model substantially outperforms related approaches. Going forward, we aim to apply this approach in real-world scenarios.

Keywords: reinforcement learning; motion sensors; ray casting; signal processing; time series processing; transformer model; robotics; Flappy Bird game; agent control

check for updates

Citation: Dirgová Luptáková, I.; Kubovčík, M.; Pospíchal, J. Playing Flappy Bird Based on Motion Recognition Using a Transformer Model and LIDAR Sensor. *Sensors* **2024**, *24*, 1905. https://doi.org/10.3390/s24061905

Academic Editors: Arturo de la Escalera Hueso, Fernando Fernández-Martínez, Manuel Gil-Martín and Rubén San-Segundo

Received: 22 December 2023
Revised: 7 March 2024
Accepted: 14 March 2024
Published: 16 March 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The autonomous control of robots using reinforcement learning (RL) has emerged as one of the important topics in machine learning. The extensive use of deep neural network technology has made it the most common choice for creating control systems that rely on information collected from a robot's operating environment. This paper focuses on processing the collected data within a time framework and using motion information to control the robot's actions. The architecture used is the transformer model [1], which can efficiently process long time series [2].

The popular computer game Flappy Bird created by Vietnamese programmer Dong Nguyen [3] acts as the simulation environment here. The goal of the player, who controls a simulated robot bird, is to fly continuously forward without a collision. The bird encounters a succession of pairs of pipes obstructing its path, and they are suspended from the top and protrude from the bottom of the game environment. A constant distance is maintained between each pair of pipes, forming a gap through which the bird can fly. The vertical position of this gap is randomly generated, introducing a dynamic element to the game. The ever-changing environment demands that players adapt and quickly react. Gravity pulls the bird downward, whereas the player's actions push the bird upward. Horizontal velocity remains constant. The game concludes instantly if the bird collides with either a pipe or the ground.

There are several approaches to train players in Flappy Bird. One typical approach is to use the image generated by the game [4], with various adaptations. Another modification involves introducing an extra negative reward when the agent collides with the upper edge of the game screen [5]. Further modifications are based on the creation of three training difficulty levels, easy, medium, and hard [6], which are distinguished by the width of the gap between pipes. The subsequent method involved a computer expert who extracted key information from the pipes and the agent, which is then used to predict actions [7].

In this paper, the player–bird is equipped with a simulated light detection and ranging (LIDAR) sensor represented by the ray casting method to detect pipelines and ground. The player can utilize time-series signal processing to maneuver around pipelines and avoid collisions. The goal is to use motion data to navigate through obstacles, such as pipes and the ground. In this model, a custom-built deep neural network, called here the "motion transformer", is employed for both time series and ray casting signal processing.

A similar approach to processing temporal components is used by [8–10]. However, these papers primarily focus on processing human activity data while also incorporating spatial components. All spatial measurements are interpreted as features. The transformer model in the present study is only looking for time correlations and not for correlations between the rays of the sensor.

The transformer model has already been used for action prediction, where it determines the next action based on the current state, previous actions, and rewards. However, this model specifically uses a causal transformer, limiting information processing only to one direction from the past to the future [11]. Another application utilizes the transformer model in RL as the replacement of convolutional layers for feature extraction. This is the case of the Swin Transformer model used for image processing [12]. It differs from the present paper, which does not incorporate the entire game screen as part of its input. The next application of the transformer model is in the temporal domain, but it only considers using the last timestep for action prediction, while the remaining timesteps are only used in the learning process to compute the model's error. It also uses a causal transformer [13].

Additional strategies for enhancing time-series prediction involve utilizing the last timestep, averaging features across timesteps, and determining the maximum value across timesteps.

The vision transformer [14] uses the features from the last timestep for action prediction, notably through its use of the class token. In the present paper, the last timestep represents the final state of the game, eliminating the necessity for an additional class token in the time series.

The average of the features across timesteps is used in the paper [15], and their maximum value is reported in [16]. The final alternative involves merging features over time, though this approach may result in a proliferation of inputs to the subsequent layer and contribute to overfitting in the model [17].

In contrast to previous research on Flappy Bird, the present paper aims to use the understanding of the temporal context from ray casting measurements. We employed the transformer model to process historical state measurements, subsequently aggregating these data in a judicious manner to forecast the current action of the agent. The sensor simulated in our study has a more restricted field of view when compared to the methods used in prior research [18]. Therefore, our model is designed to leverage its past knowledge of obstacles within the environment for effective navigation. Unlike neural network models that have been already applied to the Flappy Bird problem, our goal is to devise a method that can effectively condense information transmitted over time. This will allow us to express the qualities of actions for the current state of the agent and its corresponding response. Consequently, our model is designed to predict a categorical distribution of actions based on the current state of the agent, taking into account the agent's previously evaluated states. This approach allows the model to make informed decisions based on both the current and past states of the agent.

The key contributions of this paper are as follows:

- Improved performance: Our transformer model with a distance sensor significantly outperformed existing methods (an increase of over 50 times in both average and maximum scores). This suggests that real robots equipped with similar sensors can potentially achieve considerably higher accuracy when processing long sequences of sensor data.

- Sensor-focused learning: Unlike previous approaches, our agent solely relies on sensor data (not on the full game image) to learn from past experiences, identify obstacles, and navigate the environment. This suggests that focusing on relevant sensor data can be an efficient strategy for controlling robots.
- Visualizing and tracking the temporal similarity of sensor data: This research introduces a visualization technique to track similarities within sensor data sequences during a transformer's model training. This technique helps adjust the model to focus on the crucial measurements that impact the game's strategy and ultimate outcome, effectively discarding non-critical information. This approach was developed to reduce training times and lower memory requirements for the agent.
- Real-world applicability: Our findings have the potential to be applied to real robots operating in hazardous environments (comparable to the Flappy Bird simulation, where the agent can crash). By incorporating a "private zone" concept and deep learning guidance, robots could potentially navigate complex tasks while minimizing collisions and extending their operational lifespan.

This paper is organized as follows. Section 2 provides a review of the core algorithmic and computational approaches employed in this work. These include the dueling network architecture for Q-learning, the motion transformer architecture, the DeepMind Reverb database server used for machine learning, ray casting for obstacle detection, episodic memory incorporated into the transformer's input, and the private zone concept that aids in obstacle avoidance. Section 3 details the optimization process for the chosen methods and their hyperparameters. This section explores factors such as the number of timesteps used, the feature reduction techniques applied, and the optimal size of the private zone. It concludes with a crash analysis to assess the potential for enhancing the ultimate outcomes. Section 4 discusses future applications of this method and explores promising ways to improve it. Finally, Section 5 summarizes the key findings and conclusions presented throughout the paper.

2. Materials and Methods

The transformer model is trained by the dueling deep Q network approach. To achieve effective learning, we need to collect data on various paths explored within the state space and share the updated characteristics of our computational model. This task is facilitated by a specialized DeepMind Reverb database server. The state space only contains measurements from ray casting. The measurements from ray casting therefore warrant a dedicated exposition. Since the transformer is built on episodic memory, its usage in the Flappy Bird problem is independently addressed. Finally, an innovative approach involves the establishment of a private zone surrounding the agent to enhance its ability to maintain a secure distance while navigating obstacles. The introduction of this concept markedly improves performance throughout the learning process. A thorough analysis of these methodologies will be conducted in subsequent sections.

2.1. Dueling Deep Q Network

The principle of dueling network architecture is to extract features from the state space that are relevant for value function and advantage function prediction. The value function expresses how advantageous the current state of an agent is for its policy. The agent prioritizes traversing states that possess higher values. This strategy ensures the maximization of the overall value function. In order to make an informed selection among a multitude of potential actions, it is essential to ascertain the benefit associated with each action. This is achieved through the utilization of an advantage function [19]. In the case of a discrete action space, the probabilities of each action need to be expressed in the form of logits, which are predicted by a deep neural network model [20]. Logits represent Q-values, which can be computed according to the following relation [21]:

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

$\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a})$ expresses the quality function for a given action $\boldsymbol{a}$ and in a given state $\boldsymbol{s}$. $\boldsymbol{V}(\boldsymbol{s})$ expresses the value function for a given state $\boldsymbol{s}$. $\boldsymbol{A}(\boldsymbol{s},\boldsymbol{a})$ expresses the advantage function for a given action $\boldsymbol{a}$ in a given state $\boldsymbol{s}$. The average of the advantage function across actions in a given state $\underline{s}$ is subtracted from the $\boldsymbol{A}(\boldsymbol{s},\boldsymbol{a})$ function. Therefore, the advantage action has a zero mean [22].

The model is trained with the logarithmic hyperbolic cosine (LogCosh) error function, which is less sensitive to outliers than the more conventional mean squared error (MSE) function [23]. The error function of the model is expressed by the following:

$$\boldsymbol{L}(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s')\sim\boldsymbol{\mathcal{U}}(\boldsymbol{D})}\left[\boldsymbol{LogCosh}\left(\boldsymbol{y}^{\boldsymbol{DQN}} - \boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\theta})\right)\right] \tag{2}$$

$$\boldsymbol{y}^{\boldsymbol{DQN}} = \boldsymbol{r} + \boldsymbol{\gamma}\max_{\boldsymbol{a}'}\boldsymbol{Q}\left(\boldsymbol{s}',\boldsymbol{a}';\boldsymbol{\theta}^{-}\right) \tag{3}$$

$\boldsymbol{\mathcal{U}}(\boldsymbol{D})$ represents the uniform sampling from the replay buffer $\boldsymbol{D}$ that contains trajectories. $\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\theta})$ expresses the Q-value predicted by the model. The reward is symbolized by $\boldsymbol{r}$. $\boldsymbol{a}'$ is the next action expressed by the maximum Q-value in the next state $\boldsymbol{s}'$. $\boldsymbol{\theta}^{-}$ are parameters of the exponential moving average (EMA) model [24].

2.2. Motion Transformer

The motion transformer architecture is based on the encoder block in the transformer model [25]. The purpose of the encoder block is to traverse the input vector across the timeline in both directions. In this way, it is possible to look for relationships in historical data from past to future or from future to past and possibly associate them appropriately with the last timestep. The last timestep represents the source of information in the classical Markov decision process (MDP) [26]. A state vector representing the local memory of the model is fed to the model's input. The task of the model learning process is then to optimize the global memory (parameters) of the model so that the state space is ideally transformed into an action space. However, the output of the encoder block again represents a sequence; i.e., for each timestep, it predicts a set of extracted features from the input vector. Here, several methods are presented for extracting one particular distribution of the current action $\boldsymbol{a_t}$. One possible approach is to only use the extracted features from the last timestep to predict the distribution of actions $\boldsymbol{a_t}$, similarly to the class token [27]. The idea is to use the last timestep $\boldsymbol{s_t}$ to predict action $\boldsymbol{a_t}$ as in classical MDP. If some historical features are needed, they are inserted during the last timestep thanks to the attention mechanism. Another possibility is to use the average or maximum across all timesteps for each extracted feature separately.

Figure 1 shows the architecture of the motion transformer. The architecture consists of a preprocessing layer that adds position information to the input vector within the time series. This is followed by several encoder blocks that extract features along the time axis. The layer labeled X represents the reduction layer of the extracted features across the time series. Its type was varied during experiments. The last layers are value and advantage, representing fully connected output layers. Equation (1) is then applied to the output of the motion transformer.

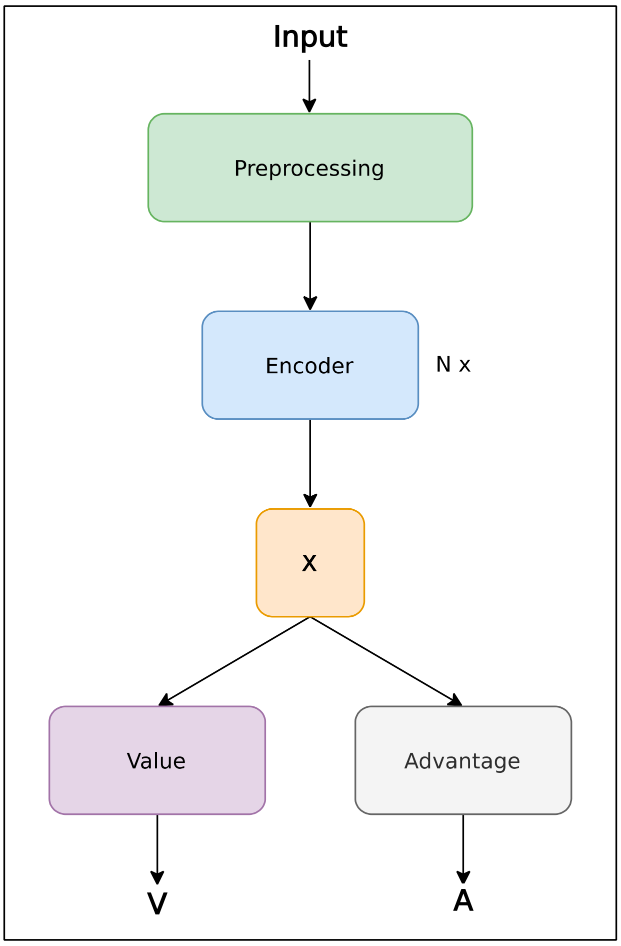

Figure 1. The architecture of the motion transformer model.

Figure 2 depicts the architecture of the preprocessing layer, which includes a projection layer in the form of a fully connected layer with a linear activation function. Its role is to transform the number of input features into the number of hidden features used in the rest of the model. Subsequently, a positional embedding, which is represented by trainable variables, is summed with the output of a fully connected layer. Thus, in this paper, embeddings for the time series are trained, along with the model [28].

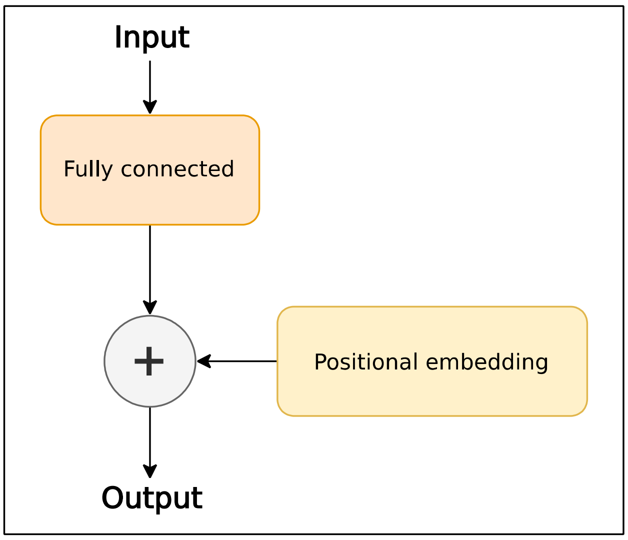

Figure 2. The architecture of the preprocessing layer.

The encoder block architecture is depicted in Figure 3. It consists of a pair of residual [29] sub-blocks. The first is multi-head attention [30], which processes the time series according to the following relations:

$$MultiHead(Q,K,V) = Concat(head_1, \ldots, head_n)W^O + b^O \tag{4}$$

$$head_i = Attention\left(QW_i^Q + b_i^Q, KW_i^K + b_i^K, VW_i^V + b_i^V\right) \tag{5}$$

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6}$$

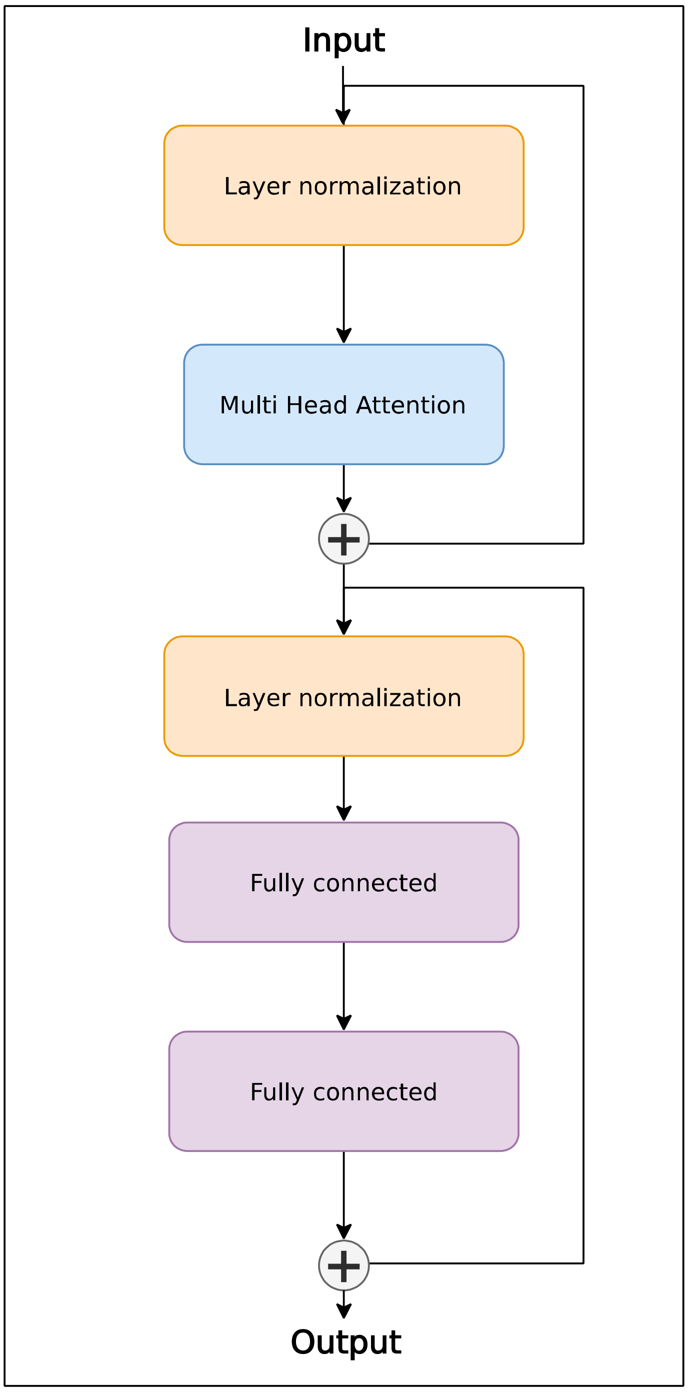

Figure 3. Architecture of encoder layer.

$\boldsymbol{W^O}$ represents weights, and b^O represents the biases of the linear transformation after merging heads. The process of merging heads comprises concatenating tensors along the head dimension (axis). $\boldsymbol{W_i^Q}$, $\boldsymbol{W_i^K}$, and $\boldsymbol{W_i^V}$ represent weights, and $\boldsymbol{b_i^Q}$,$\boldsymbol{b_i^K}$, and $\boldsymbol{b_i^V}$ represent biases of the linear transformation of the input vector of layer $\boldsymbol{Q}$ (Query), $\boldsymbol{K}$ (Key), and $\boldsymbol{V}$ (Value) into the space handled by the attention function. $\boldsymbol{d_k}$ represents the number of dimensions K after the linear projection of the layer input vector.

The second block is a multi-layer perceptron (MLP) [31], and its task is to nonlinearly transform the processed time series. The nonlinear activation function used is Gaussian error linear units (GeLUs) [32], applied after the first fully connected layer. It can be expressed by the following relation:

$$\boldsymbol{y} = \boldsymbol{GeLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{7}$$

$\boldsymbol{W}_1$ and $\boldsymbol{b}_1$ represent weight and bias parameters for the first fully connected layer to which the nonlinear transformation is subsequently applied. The parameters $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$ represent the second layer of the block. This layer is responsible for executing a linear transformation on the output derived from the preceding layer. The dimension of this transformed output equals the dimension of the original input vector. Typically, the first layer of the block has 4 times more neurons than the last layer of the block [33].

2.3. Database Reverb

The DeepMind Reverb database server is used to effectively manage the collected trajectories and distribute the updated model parameters. This dedicated database server is tailored for RL algorithms where it acts as a replay buffer. Users can control strategies for selecting and removing elements from the database and options for controlling the ratio between sampled and inserted elements. The database server may contain several tables where trajectories or the parameters of the model are stored. An important feature is the compression of the stored data that the database server provides. In the case of overlapped trajectories, it is important to avoid storing duplicate trajectories [34]. The strategy used for sampling trajectories is the uniform sampler, which selects trajectories from the table with equal probability. The strategy for removing trajectories from a table is the first-in-first-out (FIFO) method. The ratio between sampled and inserted items is empirically set to 32 with 10% tolerance.

The client–server architecture used is illustrated in Figure 4. The server represents the database repository where trajectories are stored. The actor represents the client in the form of an agent, which gathers the experience in the form of trajectories through its interactions with the environment and stores the trajectories in the database server. The learner represents a client that retrieves trajectories from the database server and uses them to train an agent model. Following the training process, the agent receives the newly updated model parameters via the database server. A similar principle is used in the Acme framework [35].

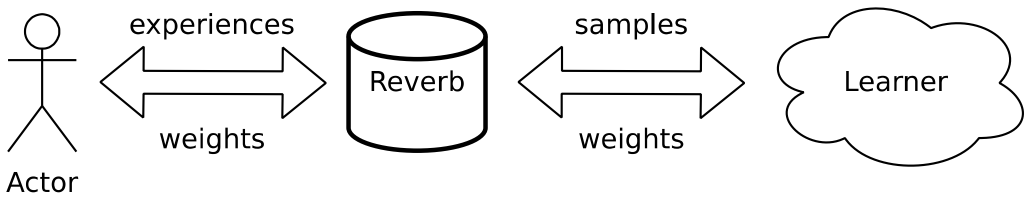

Figure 4. Client–server training.

2.4. LIDAR

The method used to detect nearby objects and track the agent's movement within the game environment employs a ray casting technique, specifically referred to as LIDAR for simplicity. It consists of 180 rays that are directed from the front of the agent to the right edge of the screen (see Figure 5). The endpoints of rays are determined by the following relations:

$$x = d_{MAX} * \cos\left(\alpha - player_{\alpha} - \frac{\pi}{2}\right) + player_{x} \tag{8}$$

$$y = d_{MAX} * \sin\left(\alpha - player_{\alpha} - \frac{\pi}{2}\right) + player_{y} \tag{9}$$

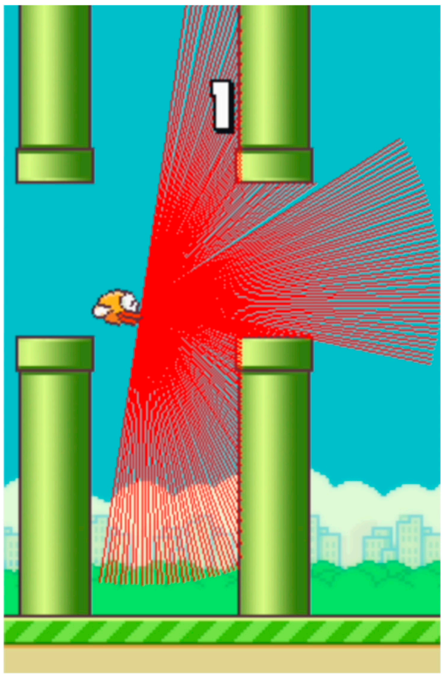

Figure 5. LIDAR sensor represented by ray casting.

d_{MAX} represents the maximum ray's length. The angle α determines the direction of ray radiation, and a $player_{\alpha}$ expresses the pitch angle of the player to the plane of the game space. The starting point of the ray is expressed by the coordinates of the front of the agents $player_{x}$ and $player_{y}$.

When the bird is pushed upward, it rotates towards the sky at a 45-degree angle. In the absence of player input, the bird slowly rotates towards the ground until reaching an angle of −90 degrees and then falls straight down.

The maximum ray length is the distance between the front of the agent and the right edge of the screen. Thus, the perpendicular ray touches the edge of the screen (if there are no obstacles) while other rays at a higher or lower angle usually do not reach the edge of the screen. This behavior mirrors the actual spread of ideal light and the measurements of its reflections from ideally reflective objects at different angles. In contrast, when other methods use an image generated by the game, the system sees it as a whole.

Here, rays emitted through ray casting spread out in a semicircular pattern and can detect obstacles within a limited area ahead of the agent. Moreover, if the bird is not positioned at the correct height and orientation relative to the game environment's plane, the detection rays do not even register the ground. Consequently, the agent cannot know its altitude throughout each episode. The sensor operates at an angular resolution of 1 degree and has a range limited only by the visible part of the environment ahead of the player. Collision with a ray occurs when the ray hits the surface of a pipe or the ground. The distance to the object is measured as the Euclidean distance between the agent's front, where the ray originates, and the collision point. However, these values are not statistically optimal for the model's input; hence, it is convenient to normalize them to the range [0, 1].

$$d_{\alpha}^{norm} = \frac{d_{\alpha}}{d_{MAX}} \tag{10}$$

d_{α}^{norm} represents the normalized distance to the object at an angle α. d_{α} expresses the measured value of the distance to the object. d_{MAX} represents the maximum ray's length. The d_{MAX} is defined as follows:

$$d_{MAX} = 0.8 * Screen_w - Player_w \tag{11}$$

$Player_w$ represents the width of the agent. The following measurements are given in pixels. The agent has a width of 34 and a height of 24. $Screen_w$ represents the width of the screen. The screen is the visible part of a game environment that can be seen when the game is rendered. The screen width is 288, and the height is 512.

2.5. Episodic Memory

The agent's state space consists of a fixed-length window of measurements from the timestep history. Therefore, it is necessary to create memory to store these measurements during a game episode. The entire contents of this memory serve as an input vector for the motion transformer. The first-in-first-out (FIFO) data structure ensures the flow of information in one direction, expressing the passage of time in the game environment. As new measurements are acquired during the episode, they replace the oldest ones in the queue. At the beginning of each episode, the queue is initialized with the initial state of the environment. While similarities are observed in the intended result relative to Atari stacking frames at the channel level [36], the present approach introduces a new timestep dimension in the input vector. This enables the model to exploit temporal relationships among measurements. The size of the memory determines how far back the model can effectively analyze measured states in the local history. Insufficient memory capacity can hinder information availability and impede effective action prediction. Conversely, excessively large memory unnecessarily drains computational resources.

Figure 6 illustrates the principle of applied episodic memory. The new state arrives at the end of the queue from the bottom. The oldest state leaves from the beginning of the queue, i.e., the top part. The input to the motion transformer represents all items that are stored in the queue and are ordered as they come in.

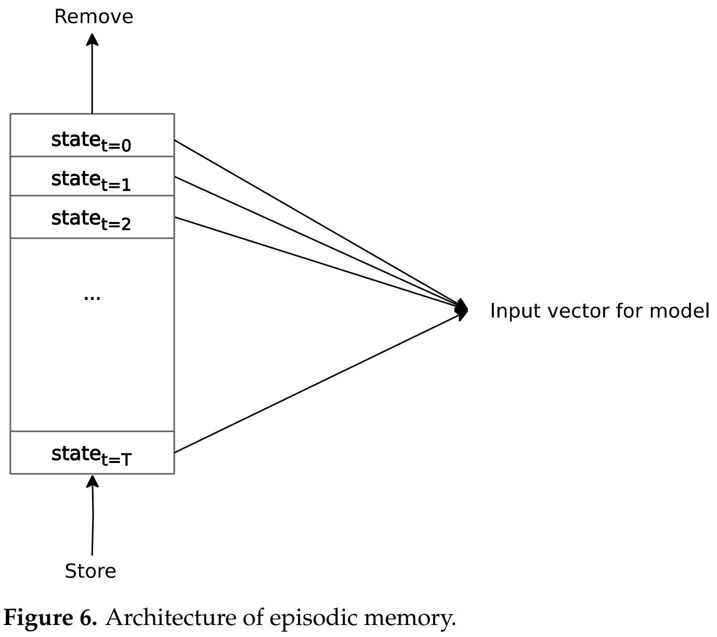

Figure 6. Architecture of episodic memory.

2.6. Private Zone around the Agent

In experiments, it was found that the agent tends to take risks and moves too close to the edges of the pipe when passing through the gap between pipes. There is a possibility of penalizing the agent for risky behavior in the policy. Therefore, a penalty for an obstacle approaching inside the agent's private zone was introduced. With this penalty in place, the agent is motivated to find the optimal solution for the given problem while considering its proximity to recognized obstacles. In a real-world application, object recognition would typically involve a dedicated deep neural network, which aims to distinguish between obstacles and desired objects, such as food or coins, in other gaming scenarios [37].

The agent needs to maintain a safe distance while navigating through obstacles to ensure its policy is not risky. This distance can be experimentally determined by finding the optimal radius for the private zone, which is represented by a circle. The circular model is chosen due to the sensor data being obtained in a circular polar grid format. As the rays are emitted from the agent's surface, the center of the private zone circle must coincide with the sensor's center. In the case where the simulation contains obstacles and objects the agent may need to interact with, such as collectibles, it is necessary to define this private zone dynamically. The classification process of obstacles vs. collectibles can be complex and involve sophisticated methods [38,39], and keeping the identified object between consecutive measurements can require specialized methods [40,41]. However, in the present game environment, where only obstacles exist, simple classification suffices.

$$\boldsymbol{r} = \frac{\boldsymbol{MAX}(\boldsymbol{Player_w}, \boldsymbol{Player_h}) + \boldsymbol{x}}{2} \tag{12}$$

The radius of the private zone circle is defined as $\boldsymbol{r}$, where $\boldsymbol{Player_w}$ represents the width of the agent and $\boldsymbol{Player_h}$ represents the height of the agent. Hyperparameter $\boldsymbol{x}$ specifies the size of the private zone. Since the rays radiate from ethe dges of the agent and not its center, when $\boldsymbol{x}$ is set to 0, the private zone's radius is equal to half of the agent's maximum dimension.

Figure 7 illustrates the agent's private zone, where the parameter $\boldsymbol{x}$ is set to 30. The gap between pipes is fixed at a size of 100 units (i.e., pixels), and the width of each pipe measures 52 units. As can be seen, a high value of $\boldsymbol{x}$ penalizes the agent if it attempts to navigate through the gap between pipes. Conversely, an $\boldsymbol{x}$ value that is too low diminishes the existence of a private zone, prompting the agent to take increased risks.

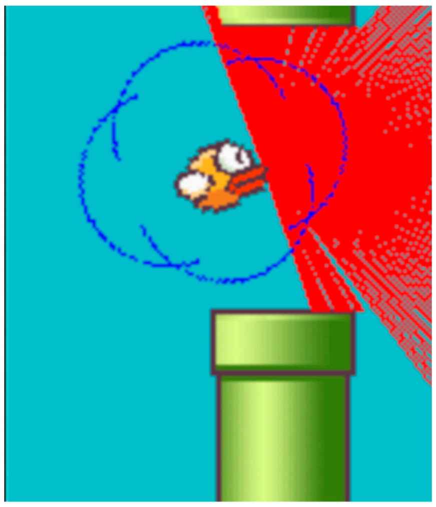
Figure 7. Agent's private zone.

3. Results

In order to improve the performance of the deep neural network controlling Flappy Bird's obstacle avoidance, various techniques required finetuning. This included selecting the right control system architecture and algorithmic techniques, as well as choosing appropriate hyperparameters during implementation. The following section provides detailed explanations of the key aspects of this process.

One aspect involved optimizing the number of timesteps retained in episodic memory. This determined the extent to which the agent would recall the short-term history and utilize it in its action predictions. Additionally, the study focused on refining the architecture of the model. Specifically, it explored whether it was effective to use the last timestep of the output series of actions. Alternatives included the global average or global maximum pooling. These operations involve computing the average or maximum of features across the timestep axis. These methods are commonly employed for reduction tasks, as seen in vision transformers and convolutional neural networks. Lastly, attention was directed towards determining the optimal size of the private zone, with options set to 30, 15, and 0.

The first tested configuration used 16 timesteps as the episodic memory size. Figure 8 shows the cosine similarity between embeddings for different pairs of timesteps. The closest similarities are in the region of the upper-left corner of the heatmap. Therefore, the subsequent experiment aimed to decrease the number of timesteps to diminish the density of similarities among timesteps and refine the optimal number of timesteps. Since there exists similarity among the initial timesteps, it is feasible to restrict their number. A total of 12 timesteps were used, which was anticipated to decrease the density of similarities, particularly in the upper-left corner.

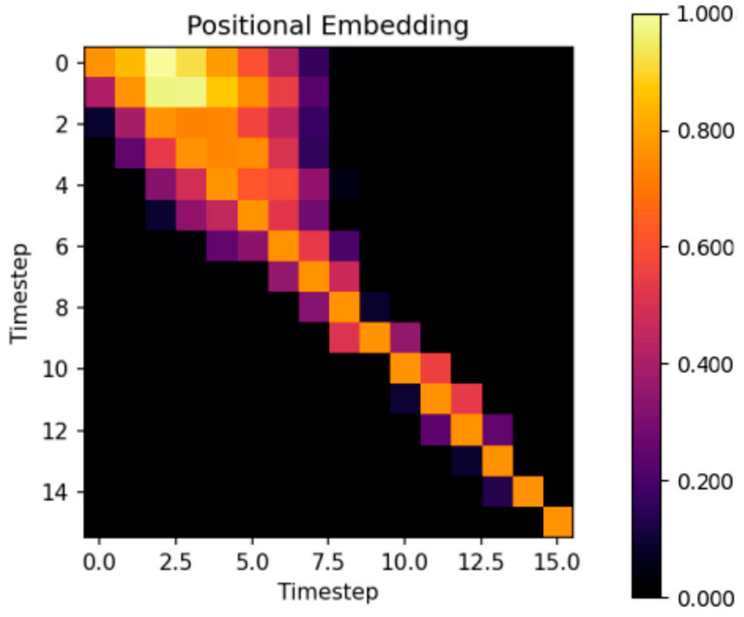

Figure 8. Similarity of timestep embeddings between 16 different timesteps.

The second experiment was to use only 12 timesteps. The cosine similarity between timestep embeddings is depicted in Figure 9. In contrast to using 16 timesteps, the density of timestep embedding similarities in the upper-left corner decreased, but the score of the agent did not significantly deteriorate. Figure 9 is not merely a subset of Figure 8; the difference in the density of similarities is apparent. Additionally, the similarity distribution is not perfectly symmetrical along the diagonal, with past steps showing more resemblance to future steps, especially for distant timeframes. This trend, however, is not observed in recent timesteps. Based on these observations, it could be beneficial in future experiments to explore using fewer past timesteps, as they exhibit similarities to future steps. Timesteps fewer than 12 or higher than 16 were not tested in this study.

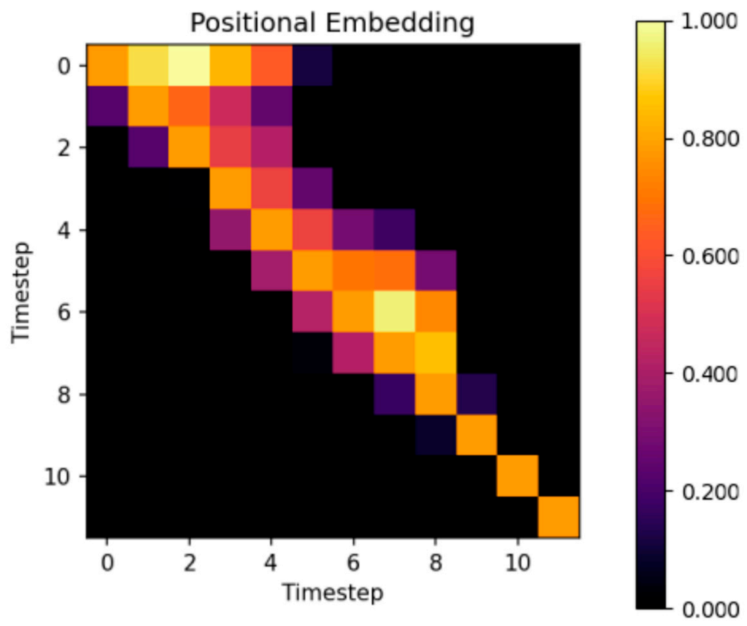

Figure 9. Similarity of timestep embeddings between 12 different timesteps.

In our investigation, we found that as the timestep increases, the similarity of embeddings decreases. This trend is particularly evident in the final timestep, regardless of whether there are 16 or 12 timesteps configured. Typically, the Markov decision process is applied only to the current state s_t. This implies that the most unique timestep must be the last timestep, which was also supported by measurement with both the 16 and 12 timestep configurations. Specifically, the last timestep exhibits the highest similarity only relative to itself. In this paper, a typical Markov decision process is modified. The historical states and current state are used simultaneously $s_{t\text{-}N:t}$, with the exception of the first state s_t due to its lack of existing history. Some historical states can probably have similar meanings for the agent. This adaptation draws parallels with word embedding, where words with similar meanings have a higher positive cosine similarity, but on the other hand, words with much different meanings have a small cosine similarity near zero [42].

In the following measurements, the comparison of the last timestep, global average, and global maximum pooling used data collected from 500 episodes. The average and maximum scores across episodes were measured for a deterministic, pre-trained agent. The score represents the number of pipes that the agent successfully passed through.

Table 1 shows the results of comparisons between different reduction techniques. The average of features along the timestep axis is significantly better than other approaches.

Table 1. Results of tested reduction methods.

Architecture	Timesteps	Highest Score	Average Score
Global average pooling	16	**2970**	324.198
Last timestep	16	2809	286.394
Global maximum pooling	16	1948	329.194
Global average pooling	12	2348	**380.284**
Last timestep	12	1922	335.114
Global maximum pooling	12	1128	152.858

The highest score is emphasized in boldface font.

Table 2 presents a comparison of the best results achieved in both the highest score and average score in this paper in contrast to other papers. This paper has significantly better scores.

Table 2. Caption.

Paper	Highest Score	Average Score
[43]	15	3.300
[4]	80	16.400
[6]	215	82.200
[5]	-	102.170
[7]	1491	209.298
This paper without a private zone	**2970**	**380.284**
This paper with a private zone	**74,755**	**13,156.590**

The score obtained in this paper is highlighted in boldface font.

Figures 10–12 show the tracking of the shifting pipelines along the timeline. The agent uses the history from 16 timesteps. A link to a video showing an animation of the changing attention matrix along with the changing environment is provided in the Supplementary Materials.

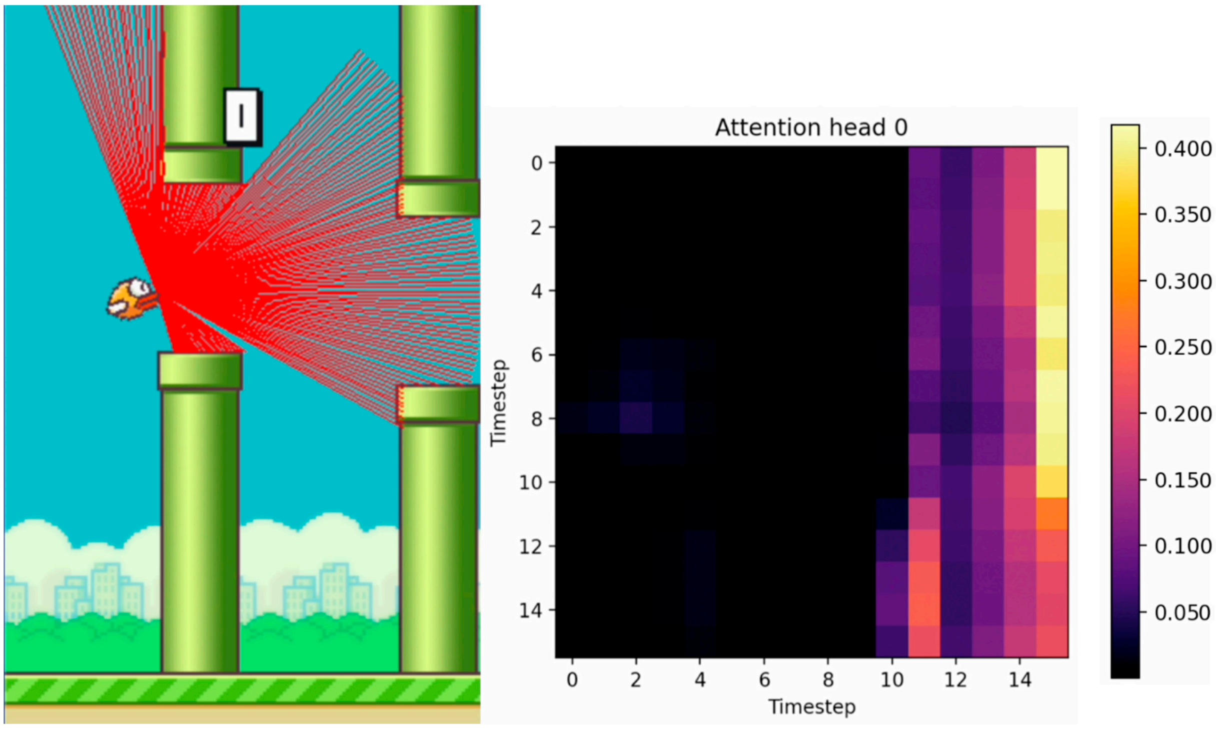

Figure 10. Agent with 16 timesteps entering the gap between the upper and lower pipes. (The brighter the color, the higher the attention value).

From the analysis of the agent's policy, it is evident that the agent takes risks and approaches the upper or lower pipes while passing through the gap between the pipes. To address this issue, it is necessary to designate a zone for the agent, beyond which, if obstacles are detected, the agent incurs a penalty of -0.5.

Likewise, the same -0.5 penalty stipulated in [5] for the agent reaching the top of the screen is also applied to the obstacles in the agent's private zone. In this game environment, all objects are regarded as obstacles.

Conversely, if the agent maintains a distance from the obstacles that is above a critical threshold, it is rewarded with a "still alive" reward valued at +0.1, similarly to [44].

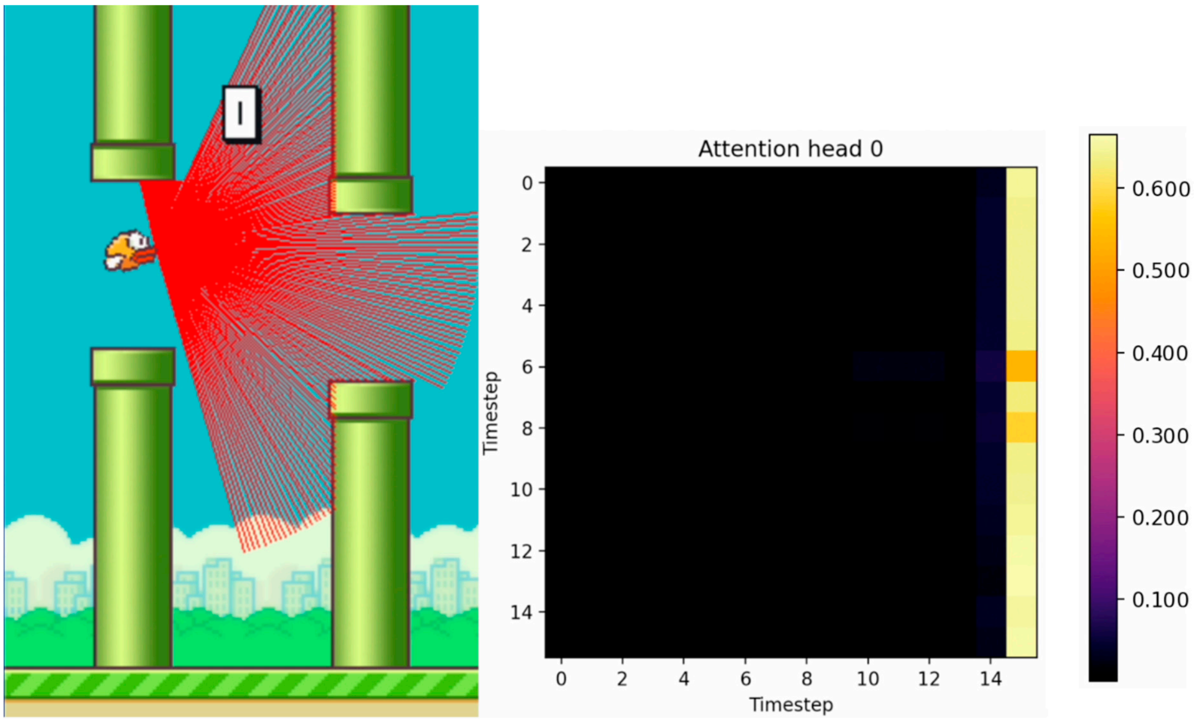

Figure 11. Agent with 16 timesteps in the gap between the upper and lower pipes. (The brighter the color, the higher the attention value).

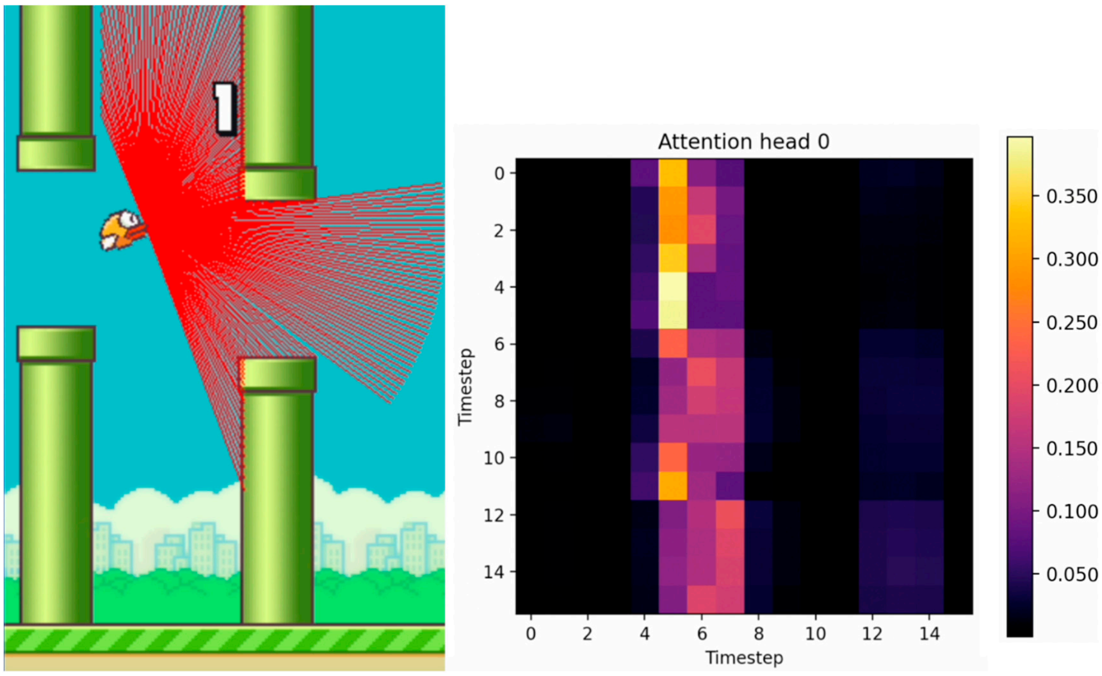

Figure 12. Agent with 16 timesteps passed the gap between the upper and lower pipes. (The brighter the color, the higher the attention value).

Figure 13 shows a histogram of the agent's score across various feature reduction techniques and private zone sizes. Experiments involving different feature reduction methods did not incorporate a penalty in the reward function for approaching obstacles too closely. Meanwhile, experiments with varying private zone sizes utilized global average pooling with 16 timesteps for feature reduction. Comparing the use of global maximum pooling to global average pooling, it is evident that the agent has a higher likelihood of scoring below 10 when employing the former method. The deep Q network generally overestimates the predicted Q-values [45]. Consequently, employing global maximum pooling may result in an overestimation of Q-values and a more risk-prone policy for the agent.

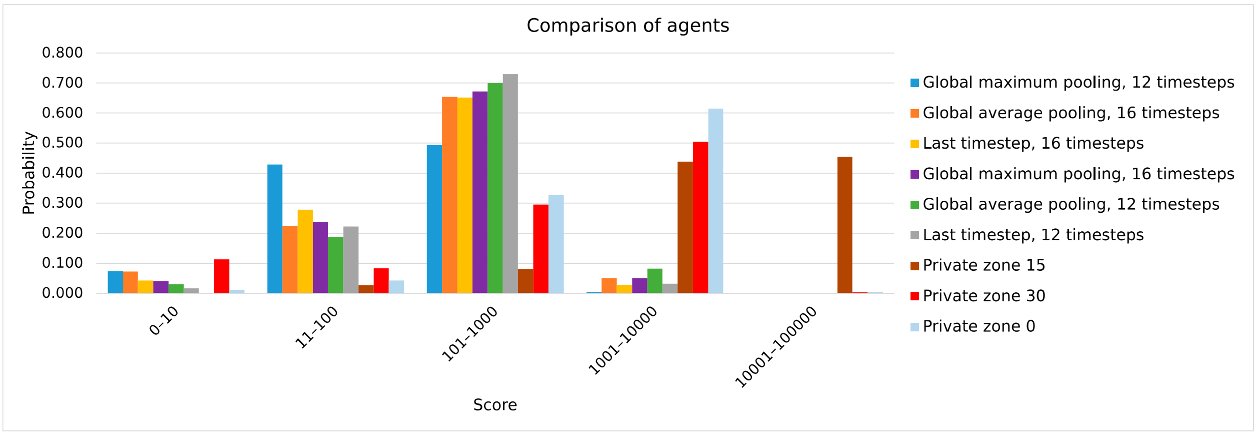

Figure 13. Histogram of score.

In the present study, it was observed that when only the last timestep was utilized, akin to the class token in the vision transformer [46], the global average pooling performed similarly to global maximum pooling.

The most stable control of Flappy Bird among the tested options of feature reduction was achieved via the global average pooling reduction method. It provided the highest maximum scores and average scores compared to the other methods of feature reduction. In contrast to global maximum pooling, global average pooling weighs down the activation by combining maximal and non-maximal activations [47]. This behavior leads to a reduction in the overestimation of Q-values predicted by the model and a less risky policy for the agent.

It was observed that using the optimal private zone size resulted in the agent achieving scores that were many times higher. The probability of obtaining a score of less than 100 was extremely low. Furthermore, a high probability of obtaining a score greater than 1000 was observed compared to agents without a private zone.

Table 3 presents a comparison between different private zone sizes. From a selection of several options, results indicate an optimal private zone size of 15. Excessively large values of private zone sizes would also penalize the agent for flying through the pipeline gap until it passes its center, which counts as a high positive reward of +1.0 to the exclusion of the other values of the reward function [48].

Table 3. Comparison of scores with different sizes of private zones.

Private Zone	Highest Score	Average Score
None	2970	380.284
0	10,250	2138.858
15	**74,755**	**13,156.590**
30	11,383	1645.654

Figure 14 presents the crash analysis for the collisions of the Flappy Bird without the private zone and with the optimal size private zone. While the score with the private zone is better by several orders of magnitude, the crush analysis shows that there still exists room for improvement. A robust solution should have an equal probability of hitting potential obstacles, while the results show that the Flappy Bird tends to crash almost exclusively into the bottom end of the upper pipe. The introduction of the private zone has minimized potential impact points, allowing future focus on suppressing these collisions. One approach to achieving this goal is to design a more robust reward function.

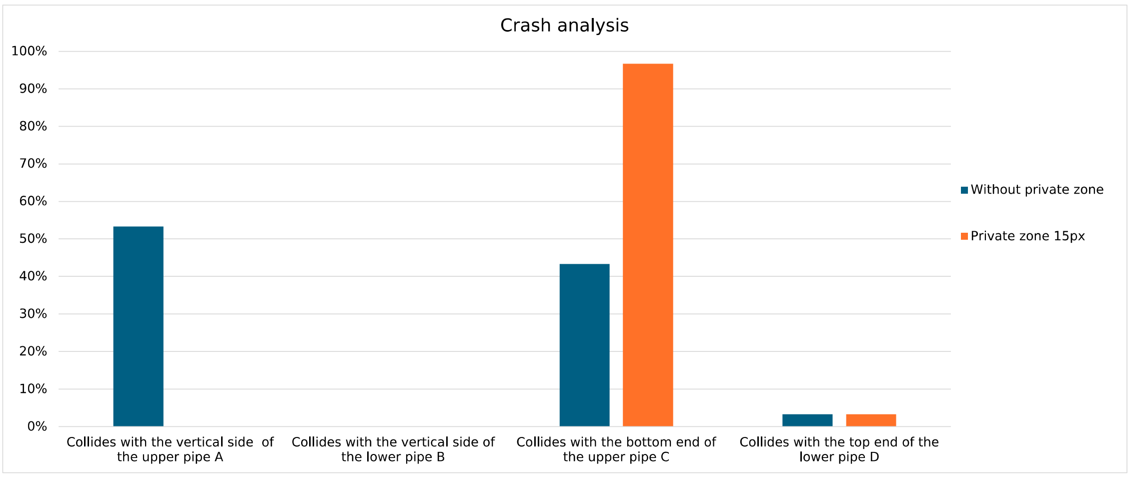

Figure 14. Crash analysis with and without the private zone.

Table 4 displays the hyperparameters used in all the experiments performed. Their values are set based on a combination of recommended settings. The recommended size of the replay buffer and the discount factor are taken from [49]. The multiplier of the MLP block dimension, type of learning rate schedule, and gradient clipping are based on [50]. A private zone with a value of None indicates the absence of a penalty rule for approaching obstacles in the reward function. The other numerical values of the private zone size express the x of Equation (12).

Table 4. Hyperparameters of the agent and learner model.

Hyperparameter	Description	Value
port	Database server port	8000
max_replay_size	Maximum database memory	1,000,000
samples_per_insert	Samples per insert ratio for reverb	32
temp_init	Initial Boltzmann temperature for exploration	0.500
temp_min	Minimal Boltzmann temperature	0.010
temp_decay	Decay of Boltzmann temperature	0.999999
warmup_steps	Warmup steps for learning rate cosine scheduler	1000
train_steps	Training steps	1,000,000
batch_size	Batch size	256
gamma	Discount factor	0.990
tau	Tau factor (for EMA model)	0.005
num_layers	Num. of encoder blocks	2
embed_dim	Embedding dimension	128
ff_mult	Multiplier of MLP block dimension	4
num_heads	Num. of attention heads	6
learning_rate	Learning rate	3×10^{-4}
global_clipnorm	Globally normalized clipping of gradient	1
weight_decay	Weight decay for AdamW optimizer	1×10^{-4}
frame_stack	Size of short-term (episodic) memory	16 or 12
player_private_zone	Size of agent’s private zone	None, 0, 15 or 30

4. Discussion

Utilizing a transformer neural network to control a simulated agent via ray casting as a simple LIDAR sensor has potentially diverse applications across several domains. Remote sensing technology integrated with advanced AI-based control can be beneficial in the following contexts:

In virtual reality and games, avatars or characters can benefit from more natural and responsive interactions.

The method for improving navigation using ray casting in 2D could potentially be expanded to utilize true LIDAR in 3D space. In the future, this could lead to advancements in robotics; autonomous vehicles like self-driving cars, drones, or any kind of mobile robots that require effective navigation; and obstacle avoidance capabilities. In disaster-stricken areas, such robots can aid in search and rescue missions. Enhanced agents can streamline tasks such as inventory management and material handling in warehouses. Additionally, robotic arms could better manipulate objects in dynamic environments.

In each of these contexts, the integration of ray casting and transformer neural network control should enable the agent to make informed decisions based on temporal and spatial information.

When considering the selection of algorithmic procedures and hyperparameters, there exists ample room for exploration and experimentation with various possibilities.

When storing high-dimensional states in episodic memory, it would be more convenient to only extract the significant features for storage. For this purpose, an AutoEncoder-type model could be used to compress the input vector.

Another consideration is the initialization of episodic memory. Currently, it duplicates the initial state, but one alternative includes creating an embedding for the empty state containing episodic memory at each episode's start. The next option is to dynamically adjust the number of timesteps with respect to the input to the motion transformer, while ensuring the proper assignment of positional embedding for incrementing states from the timesteps.

A promising research direction is exploring the impact of cosine similarity on the optimal number of timesteps. This involves investigating whether the similarity of timestep embeddings can reduce the necessary number of timesteps. Further study is needed to validate the effect of positional embedding in reducing timesteps across various game environments.

In the case of the Flappy Bird game, future research should also try the possibility of adding a weighted reward for keeping a safe distance from the upper pipe more than from other obstacles. This research direction follows from the results of the error analysis. The subject of further research is also to study and rectify failures after the agent has performed a very large number of steps in the environment. Potential improvements might be anticipated in algorithms based on the deep Q Learning principle, such as dueling deep Q learning or double deep Q learning. Numerical instability should also be checked, as well as more advanced Actor–Critic-type models such as A2C or PPO.

Further improvements could be attained by establishing a dynamic private zone around the agent. The private zone could be delineated by deep neural network prediction, whether objects crossing the private zone boundary are obstacles or aids in achieving a specific task. Such a model could directly adjust the complex reward function necessary for task completion without exposing the agent to risky behavior.

5. Conclusions

Our study presents novel guidance control using LIDAR sensors represented by the ray casting method for obstacle detection and agent navigation within obstacle-filled environments. The designed motion transformer model effectively grasped the temporal dynamics between sensor readings. The findings demonstrate the model's ability to adaptively respond to the agent's movement among pipelines, as reflected in the attention matrix. The model's attention mechanism prioritizes past or present sensor data, or a combination thereof, based on the spatial distribution of pipelines in the surroundings. Additionally, the results show that employing average reduction techniques helps mitigate the risk of overestimating Q values. Furthermore, the incorporation of a private zone for the agent contributes to the formulation of a less risky navigation policy.

In this paper, the average score (the number of passes through pipeline gaps) obtained by the agent without a private zone is 182 percent better compared to the best results obtained by the competitors. The highest score achieved by the agent without a private zone compared to the best competitors' obtained results is 199 percent better. The agent with a private zone of 15 pixels achieved an average score that was 6286 percent better than the best competitors' average agent score and a maximum score that was 5014 percent better than the competition's best results in terms of maximum agent score.

Supplementary Materials: The following supporting information can be downloaded at the following: interactive charts: https://wandb.ai/markub/rl-toolkit/groups/FlappyBird-v0 (accessed on 16 October 2023); source codes: https://github.com/markub3327/rl-toolkit (accessed on 16 October 2023) and https://github.com/markub3327/flappy-bird-gymnasium (accessed on 16 October 2023); YouTube video: https://youtu.be/aZQxuDCyHoI (accessed on 22 December 2023).

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

Funding: This research was funded by the Cultural and Educational Grant Agency MŠVVaŠ SR, grant number KEGA 020UCM-4/2022, and by Erasmus+ project FAAI: The Future is in Applied Artificial Intelligence—2022-1-PL01-KA220-HED-000088359, work package WP4.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

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

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5998–6008. Available online: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (accessed on 10 December 2023).
2. Zeng, A.; Chen, M.; Zhang, L.; Xu, Q. Are transformers effective for time series forecasting? *Proc. AAAI Conf. Artif. Intell.* **2023**, *37*, 11121–11128. Available online: https://ojs.aaai.org/index.php/AAAI/article/view/26317/26089 (accessed on 10 December 2023). [CrossRef]
3. Wei, S. Reinforcement Learning for Improving Flappy Bird Game. *Highlights Sci. Eng. Technol.* **2023**, *34*, 244–249. Available online: https://drpress.org/ojs/index.php/HSET/article/download/5479/5298 (accessed on 10 December 2023). [CrossRef]
4. Pilcer, L.S.; Hoorelbeke, A.; Andigne, A.D. Playing Flappy Bird with Deep Reinforcement Learning. *IEEE Trans. Neural Netw.* **2015**, *16*, 285–286. Available online: https://www.researchgate.net/profile/Louis-Samuel-Pilcer/publication/324066514_Playing_Flappy_Bird_with_Deep_Reinforcement_Learning/links/5abbc2230f7e9bfc045592df/Playing-Flappy-Bird-with-Deep-Reinforcement-Learning.pdf (accessed on 10 December 2023).
5. Yang, K. Using DQN and Double DQN to Play Flappy Bird. In Proceedings of the 2022 International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID 2022), Xi'an, China, 15–17 April 2022; Atlantis Press: Amsterdam, The Netherlands, 2022; pp. 1166–1174. Available online: https://www.atlantis-press.com/article/125977189.pdf (accessed on 10 December 2023).
6. Chen, K. Deep Reinforcement Learning for Flappy Bird. CS 229 Machine-Learning Final Projects. 2015. Available online: https://cs229.stanford.edu/proj2015/362_report.pdf (accessed on 10 December 2023).
7. Vu, T.; Tran, L. FlapAI Bird: Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques. *arXiv* **2020**, arXiv:2003.09579.
8. Li, J.; Yin, Y.; Chu, H.; Zhou, Y.; Wang, T.; Fidler, S.; Li, H. Learning to generate diverse dance motions with transformer. *arXiv* **2020**, arXiv:2008.08171.
9. Shi, S.; Jiang, L.; Dai, D.; Schiele, B. Motion transformer with global intention localization and local movement refinement. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 6531–6543. Available online: https://proceedings.neurips.cc/paper_files/paper/2022/file/2ab47c960bfee4f86dfc362f26ad066a-Paper-Conference.pdf (accessed on 10 December 2023).
10. Hu, M.; Zhu, X.; Wang, H.; Cao, S.; Liu, C.; Song, Q. STDFormer: Spatial-Temporal Motion Transformer for Multiple Object Tracking. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *33*, 6571–6594. Available online: https://ieeexplore.ieee.org/iel7/76/4358651/10091152.pdf (accessed on 10 December 2023). [CrossRef]
11. Esslinger, K.; Platt, R.; Amato, C. Deep Transformer Q-Networks for Partially Observable Reinforcement Learning. *arXiv* **2022**, arXiv:2206.01078.
12. Meng, L.; Goodwin, M.; Yazidi, A.; Engelstad, P. Deep Reinforcement Learning with Swin Transformer. *arXiv* **2022**, arXiv:2206.15269.

13. Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 15084–15097. Available online: https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf (accessed on 10 December 2023).
14. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image Is Worth 16 × 16 Words: Transformers for Image Recognition at Scale. *arXiv* **2020**, arXiv:2010.11929.
15. Liu, R.; Ji, C.; Niu, J.; Guo, B. Research on intrusion detection method based on 1D-ICNN-BiGRU. In *Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK, 2022; Volume 2347, p. 012001. Available online: https://iopscience.iop.org/article/10.1088/1742-6596/2347/1/012001/pdf (accessed on 10 December 2023).
16. Crocioni, G.; Pau, D.; Delorme, J.M.; Gruosso, G. Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent IoT microcontrollers. *IEEE Access* **2020**, *8*, 122135–122146. Available online: https://ieeexplore.ieee.org/iel7/6287639/6514899/09133084.pdf (accessed on 10 December 2023). [CrossRef]
17. Gholamalinezhad, H.; Khosravi, H. Pooling Methods in Deep Neural Networks, a Review. *arXiv* **2020**, arXiv:2009.07485.
18. Anders, K.; Winiwarter, L.; Lindenbergh, R.; Williams, J.G.; Vos, S.E.; Höfle, B. 4D objects-by-change: Spatiotemporal segmentation of geomorphic surface change from LiDAR time series. *ISPRS J. Photogramm. Remote Sens.* **2020**, *159*, 352–363. Available online: https://www.sciencedirect.com/science/article/pii/S0924271619302850 (accessed on 10 December 2023). [CrossRef]
19. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, New York City, NY, USA, 19–24 June 2016; pp. 1995–2003. Available online: http://proceedings.mlr.press/v48/wangf16.pdf (accessed on 10 December 2023).
20. Haarnoja, T.; Tang, H.; Abbeel, P.; Levine, S. Reinforcement learning with deep energy-based policies. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 1352–1361. Available online: http://proceedings.mlr.press/v70/haarnoja17a/haarnoja17a.pdf (accessed on 10 December 2023).
21. Peng, B.; Sun, Q.; Li, S.E.; Kum, D.; Yin, Y.; Wei, J.; Gu, T. End-to-end autonomous driving through dueling double deep Q-network. *Automot. Innov.* **2021**, *4*, 328–337. Available online: https://link.springer.com/content/pdf/10.1007/s42154-021-00151-3.pdf (accessed on 10 December 2023). [CrossRef]
22. Liu, F.; Li, S.; Zhang, L.; Zhou, C.; Ye, R.; Wang, Y.; Lu, J. 3DCNN-DQN-RNN: A deep reinforcement learning framework for semantic parsing of large-scale 3D point clouds. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 5678–5687. Available online: https://openaccess.thecvf.com/content_ICCV_2017/papers/Liu_3DCNN-DQN-RNN_A_Deep_ICCV_2017_paper.pdf (accessed on 10 December 2023).
23. Saleh, R.A.; Saleh, A.K. Statistical Properties of the Log-Cosh Loss Function Used in Machine Learning. *arXiv* **2022**, arXiv:2208.04564.
24. Tarvainen, A.; Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1195–1204. Available online: https://proceedings.neurips.cc/paper/2017/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf (accessed on 10 December 2023).
25. Tummala, S.; Kadry, S.; Bukhari, S.A.C.; Rauf, H.T. Classification of brain tumor from magnetic resonance imaging using vision transformers ensembling. *Curr. Oncol.* **2022**, *29*, 7498–7511. Available online: https://www.mdpi.com/1718-7729/29/10/590/htm (accessed on 10 December 2023). [CrossRef]
26. Wang, X.; Yang, Z.; Chen, G.; Liu, Y. A Reinforcement Learning Method of Solving Markov Decision Processes: An Adaptive Exploration Model Based on Temporal Difference Error. *Electronics* **2023**, *12*, 4176. Available online: https://www.mdpi.com/2079-9292/12/19/4176 (accessed on 10 December 2023). [CrossRef]
27. Feng, H.; Yang, B.; Wang, J.; Liu, M.; Yin, L.; Zheng, W.; Yin, Z.; Liu, C. Identifying malignant breast ultrasound images using ViT-patch. *Appl. Sci.* **2023**, *13*, 3489. Available online: https://www.mdpi.com/2076-3417/13/6/3489 (accessed on 10 December 2023). [CrossRef]
28. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**, arXiv:1810.04805.
29. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 770–778. Available online: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf (accessed on 10 December 2023).
30. Hasan, F.; Huang, H. MALS-Net: A multi-head attention-based LSTM sequence-to-sequence network for socio-temporal interaction modelling and trajectory prediction. *Sensors* **2023**, *23*, 530. Available online: https://www.mdpi.com/1424-8220/23/1/530/pdf (accessed on 10 December 2023). [CrossRef]
31. Mogan, J.N.; Lee, C.P.; Lim, K.M.; Muthu, K.S. Gait-ViT: Gait Recognition with Vision Transformer. *Sensors* **2022**, *22*, 7362. Available online: https://www.mdpi.com/1424-8220/22/19/7362/pdf (accessed on 10 December 2023). [CrossRef]
32. Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (Gelus). *arXiv* **2016**, arXiv:1606.08415.
33. Sun, W.; Wang, H.; Xu, J.; Yang, Y.; Yan, R. Effective Convolutional Transformer for Highly Accurate Planetary Gearbox Fault Diagnosis. *IEEE Open J. Instrum. Meas.* **2022**, *1*, 1–9. Available online: https://ieeexplore.ieee.org/iel7/9552935/9775186/09828477.pdf (accessed on 10 December 2023). [CrossRef]

34. Cassirer, A.; Barth-Maron, G.; Brevdo, E.; Ramos, S.; Boyd, T.; Sottiaux, T.; Kroiss, M. Reverb: A Framework for Experience Replay. *arXiv* **2021**, arXiv:2102.04736.
35. Hoffman, M.W.; Shahriari, B.; Aslanides, J.; Barth-Maron, G.; Momchev, N.; Sinopalnikov, D.; Stańczyk, P.; Ramos, S.; Raichuk, A.; Vincent, D.; et al. Acme: A Research Framework for Distributed Reinforcement Learning. *arXiv* **2020**, arXiv:2006.00979.
36. Lapan, M. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*; Packt Publishing Ltd.: Birmingham, UK, 2018.
37. Singh, A.; Yang, L.; Hartikainen, K.; Finn, C.; Levine, S. End-to-End Robotic Reinforcement Learning without Reward Engineering. *arXiv* **2019**, arXiv:1904.07854.
38. Capellier, E.; Davoine, F.; Cherfaoui, V.; Li, Y. Evidential deep learning for arbitrary LIDAR object classification in the context of autonomous driving. In Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 9–12 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1304–1311. Available online: https://hal.science/hal-02322434/file/IV19-Edouard.pdf (accessed on 10 December 2023).
39. Skrinárová, J.; Huraj, L.; Siládi, V. A neural tree model for classification of computing grid resources using PSO tasks scheduling. *Neural Netw. World* **2013**, *23*, 223. Available online: https://www.proquest.com/docview/1418215646/fulltextPDF/AF8F42E64A49412CPQ/1?accountid=49441&sourcetype=Scholarly%20Journals (accessed on 10 December 2023). [CrossRef]
40. Sualeh, M.; Kim, G.W. Dynamic multi-lidar based multiple object detection and tracking. *Sensors* **2019**, *19*, 1474. Available online: https://www.mdpi.com/1424-8220/19/6/1474/pdf (accessed on 10 December 2023). [CrossRef]
41. Kyselica, D.; Šilha, J.; Ďurikovič, R.; Bartková, D.; Tóth, J. Towards image processing of reentry event. *J. Appl. Math. Stat. Inform.* **2023**, *19*, 47–60. Available online: https://sciendo.com/article/10.2478/jamsi-2023-0003 (accessed on 10 December 2023). [CrossRef]
42. Orkphol, K.; Yang, W. Word sense disambiguation using cosine similarity collaborates with Word2vec and WordNet. *Future Internet* **2019**, *11*, 114. Available online: https://www.mdpi.com/1999-5903/11/5/114/pdf (accessed on 10 December 2023). [CrossRef]
43. Appiah, N.; Vare, S. Playing Flappy Bird with Deep Reinforcement Learning. 2018. Available online: http://vision.stanford.edu/teaching/cs231n/reports/2016/pdfs/111_Report.pdf (accessed on 10 December 2023).
44. Li, L.; Jiang, Z.; Yang, Z. Playing Modified Flappy Bird with Deep Reinforcement Learning. 2023. Available online: https://github.com/SeVEnMY/DeepLearningFinal (accessed on 10 December 2023).
45. Hasselt, H. Double Q-Learning. *Adv. Neural Inf. Process. Syst.* **2010**, *23*, 2613–2621. Available online: https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf (accessed on 10 December 2023).
46. Al Rahhal, M.M.; Bazi, Y.; Jomaa, R.M.; AlShibli, A.; Alajlan, N.; Mekhalfi, M.L.; Melgani, F. COVID-19 detection in Ct/X-ray imagery using vision transformers. *J. Pers. Med.* **2022**, *12*, 310. Available online: https://www.mdpi.com/2075-4426/12/2/310 (accessed on 10 December 2023). [CrossRef] [PubMed]
47. Passricha, V.; Aggarwal, R.K. A comparative analysis of pooling strategies for convolutional neural network based Hindi ASR. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *11*, 675–691. Available online: https://link.springer.com/article/10.1007/s12652-019-01325-y (accessed on 10 December 2023). [CrossRef]
48. Mazumder, S.; Liu, B.; Wang, S.; Zhu, Y.; Yin, X.; Liu, L.; Li, J.; Huang, Y. Guided Exploration in Deep Reinforcement Learning. In Proceedings of the International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019; Available online: https://openreview.net/forum?id=SJMeTo09YQ (accessed on 10 December 2023).
49. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *AAAI Conf. Artif. Intell.* **2018**, *32*, 1. Available online: https://ojs.aaai.org/index.php/AAAI/article/download/11796/11655 (accessed on 10 December 2023). [CrossRef]
50. Bao, H.; Dong, L.; Piao, S.; Wei, F. Beit: Bert Pre-Training of Image Transformers. *arXiv* **2021**, arXiv:2106.08254.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.