Deep Reinforcement Learning Using Double Deep Q-Network Applied to Anomaly Detection

João Antônio Fernandes Oldenburg*, Ênio dos Santos Silva*
*Instituto Federal de Santa Catarina (IFSC)
Av. Abrahão João Francisco, 3899, Ressacada District
ZIP Code: 88307-303, Itajaí, Santa Catarina, Brazil
E-mails: joao.oldenburg@gmail.com, enio.silva@ifsc.edu.br

***Abstract*—The digital transformation and the exponential growth of data generated daily have driven a global data-driven landscape, where companies increasingly seek to extract strategic value from large datasets. In this context, machine learning techniques and neural networks have become essential tools for efficiently processing and interpreting data. Anomaly detection, crucial for identifying unusual patterns, plays a fundamental role and is a highly relevant research area. To investigate and identify anomalous behaviors in time series, this work explores the application of deep neural networks combined with reinforcement learning techniques. Among the studied approaches, the Double Deep Q-Network (DDQN) stands out as an effective method in scenarios requiring the generalization of large datasets. Using the "Numenta Anomaly Benchmark" (NAB) dataset, this study investigates the effectiveness of DDQN for anomaly detection. The results highlight not only the academic contribution of this approach but also its practical impact on developing robust solutions for critical problems in dynamic and complex environments.**

***Index Terms*—Deep reinforcement learning, anomaly detection, artificial intelligence, deep neural networks, time series.**

I. Introduction

According to the Global Artificial Intelligence (AI) Adoption Index 2022 [1], AI has been increasingly integrated into business models across industries worldwide, driving substantial transformations in the global industrial landscape. The adoption of this technology spans a wide range of domains, including employee performance analysis, hiring profiling, data analytics, and decision-making optimization. This evolution highlights a significant shift in corporate markets and operational processes that had prevailed until recent years [1].

In this context, even well-established and mathematically grounded fields, such as anomaly detection, are undergoing significant transformation. Traditional statistical techniques—some of the earliest methods used to identify anomalies [2]. The task of recognizing patterns that deviate from expected behavior, referred to as anomaly detection, has become increasingly critical in today's industrial landscape [2], [3]. This growing importance is driving the development of more effective solutions and paving the way for substantial advancements in the field. As a result, anomaly detection continues to benefit from this increasing demand and, particularly when combined with machine learning techniques, plays a pivotal role in the current state-of-the-art.

Many contemporary anomaly detection systems aim to identify deviations from expected behavior; however, they often misclassify normal instances as anomalous events, leading to what is known as false positives [4]. Addressing the challenge of anomaly detection with reduced false positive rates has become particularly critical in sectors such as finance and healthcare. In this context, machine learning-based anomaly detection algorithms have been widely adopted across a range of applications. Notable examples include intrusion detection in cybersecurity, failure identification in streaming environments, monitoring of anomalous network traffic to expose compromised datasets, and fraud detection in the financial sector, particularly within banking and financial technology (fintech) institutions [5].

In the aforementioned applications, various machine learning strategies have been employed, including supervised learning, unsupervised learning, and reinforcement learning (RL). In particular, RL is an ML paradigm in which an agent interacts with an environment to learn optimal policies by maximizing cumulative rewards over time. Deep Reinforcement Learning (DRL), an extension of RL, enhances this framework by leveraging deep neural networks to handle high-dimensional input spaces. Currently, DRL represents the state of the art in reinforcement learning due to its effectiveness on large-scale datasets and its broad applicability across diverse domains, including anomaly detection, games, robotics, transportation, natural language processing, healthcare, computer vision, and finance [5]. In recent years, the application of DRL to anomaly detection has outperformed other deep learning techniques, particularly using of Deep Q-Networks (DQN), a rapidly emerging subfield within the DRL framework [5].

In the current landscape, characterized by the continuous and large-scale generation of data, the challenge of accurately and rapidly identifying anomalous patterns has become increasingly critical [6]. In this context, the present research aims to apply advanced DRL techniques to enhance anomaly detection in heterogeneous and dynamic datasets. Thus, this research work discusses the state-of-the-art in DRL and proposes a framework for anomaly detection available at https://github.com/joaoOldenburg/Anomaly_detection_with_deep_Q_network. To this end, the study explores and investigates AI and ML technologies, specifically DRL using DQN applied to anomaly detection. Among the approaches

evaluated, the Double Deep Q-Network (DDQN) stands out as an effective method, particularly in scenarios that demand strong generalization capabilities over large-scale datasets. Thus, using the Numenta Anomaly Benchmark (NAB) dataset, this study assesses the performance of the DDQN model for accurate and efficient anomaly detection.

II. Theoretical Foundation

Anomalies in time series refer to observations that significantly deviate from the expected temporal behavior of the data. In this context, statistical methods play a key role in identifying anomalies by detecting data points that deviate from specific statistical properties of a dataset, such as mean, median, mode, and quantiles. Based on well-established mathematical models widely supported in the literature, statistical techniques offer high reliability in well-behaved scenarios [7]. However, in time series characterized by complex seasonal patterns, more advanced approaches are required. These approaches typically involve decomposing the data into multiple trend components to better capture seasonal variations and improve the effectiveness of anomaly detection.

A. Conditional Variance and Standard Deviation

Conditional variance (σ^2) is a statistical measure that captures the expected uncertainty of a variable conditioned on historical data and other observed variables [8]. It is widely employed in finance, particularly in time series analysis, due to its ability to reflect the dynamic nature of asset returns. This measure is essential for modeling and forecasting investment risks, especially under heteroskedastic behavior [8].

A commonly used approach to model conditional variance is the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model. The GARCH framework is popular for its effectiveness in capturing variance persistence.The standard GARCH specification is given by:

$$\sigma_t^2 = \omega + \alpha_1 y_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{1}$$

where σ_t^2 denotes the conditional variance at time t; ω, α_1, and β_1 are model parameters; and y_{t-1} is the return at time $t-1$.

Conditional volatility models are effective in detecting structural breaks or unexpected events, such as large jumps or crashes in financial time series. The GARCH model, in particular, is capable of highlighting such deviations due to their divergence from expected volatility patterns.

Volatility models such as GARCH offer a solid foundation for risk forecasting and anomaly detection, acting as a preliminary processing stage that generates discriminative features for machine learning applications addressed in this study.

B. Anomaly Detection Using Reinforcement Learning

This section provides an introductory overview of the application of RL to anomaly detection. Figure 1 illustrates the standard RL framework. In this setting, an agent interacts with an environment by observing a state S_t and taking an action A_t, which yields a reward or penalty R_t. Through repeated interactions, the agent learns to improve its decision-making policy over time.

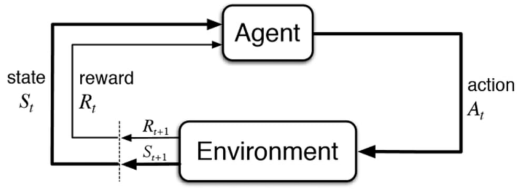

Figure 1. Illustration of a Reinforcement Learning framework.

To illustrate the application of RL to anomaly detection systems, a specific architecture is shown in Figure 2.

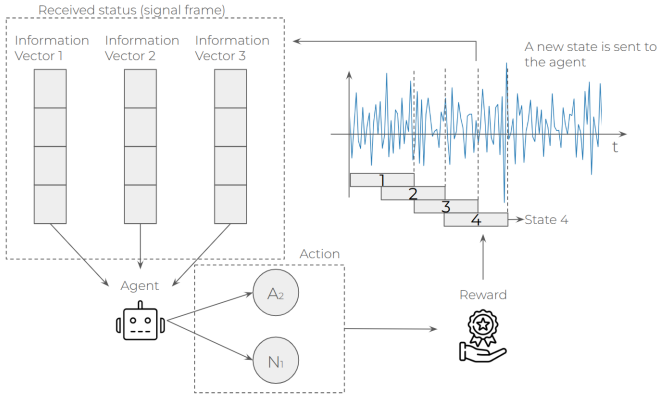

Figure 2. Operational diagram of the reinforcement learning-based anomaly detection system.

In this scenario, the agent selects between two possible actions: N_1 (normal) and A_2 (anomalous). The process begins with the input signal $x(t)$, which is segmented into frames $\mathbf{o}_t$. To enhance the discriminative power for decision-making, each segment can be preprocessed to extract additional features such as the temporal derivative $\partial \mathbf{o}_t / \partial t$ and the conditional standard deviation σ_t. These elements compose the environment's state vector at time t:

$$\mathbf{S}_t = \left[\mathbf{o}_t, \frac{\partial \mathbf{o}_t}{\partial t}, \sigma_t \right].$$

The decision-making environment is formally defined as:

1) **Environment:** Anomaly detection process;
2) **State space $\mathbf{S}_t$:** Vector representing the observed frame and its statistical attributes;
3) **Action space A_t:** Binary classification of the segment, i.e., $A_t = [N_1, A_2]$.

It is assumed that each state vector $\mathbf{S}_t$ contains sufficient statistical information to describe the characteristics of the corresponding input segment $\mathbf{o}_t$. The RL agent is thus trained to map each state to one of the two actions: classifying $\mathbf{o}_t$ as either normal (N_1) or anomalous (A_2). This formulation enables the agent to make informed decisions regarding the presence of anomalies.

The reward mechanism guides the learning process by assigning positive feedback to correct decisions and penalizing incorrect ones [9]. This feedback loop reinforces optimal behaviors, allowing the agent to adjust its policy based on cumulative experience. Consequently, RL operates as a semi-supervised learning method, combining exploratory actions with iterative refinement [9]. By aiming to maximize cumulative reward over time, the agent learns to optimize anomaly detection performance.

C. Reinforcement Learning with Q-Learning

Reinforcement learning is formally modeled as a Markov Decision Process (MDP). Then, the environment is modeled as a time series, and the agent's actions correspond to anomaly detection decisions. Under a first-order MDP assumption, the selected action a_t depends solely on the current state s_t, as it is assumed that all relevant historical information $(t-1, t-2, \ldots)$ is encoded within the present state s_t [10].

For anomaly detection in time series data, each frame is treated as a distinct state s. A reward value r is assigned to each state based on the accuracy of the anomaly detection outcome. Thus, the environment is structured as a reward-based decision process, in which state transitions and action choices determine the cumulative return.

The anomaly detection process unfolds over sequential iterations (or time steps). At each step, the agent observes a state s_t and selects an action a_t based exclusively on that state. The decision-making process is guided by the Markov property, ensuring that transitions depend only on the current state and chosen action.

This RL framework, grounded in Q-learning, is illustrated in Figure 1, where the agent's learning process evolves through exploration and reward feedback within the MDP environment.

In this context, the agent's objective is to traverse the time series and identify as many anomalies as possible while minimizing false positives. To achieve this, the agent must learn a policy π that selects the optimal action a at each state s_t, leading to a future state s_{t+1}. The current state s_t is assumed to contain sufficient statistical information for estimating (or predicting) the next state s_{t+1} [10].

A policy π can be evaluated by the expected cumulative reward from an initial state s_0 up to the current state s_t, as represented by the value function $V^\pi(s)$ in Eq. (2). Here, γ denotes the discount factor applied to past rewards r_{t-i} associated with previous states s_{t-i}, $\forall i \in \mathbb{Z}$.

$$\begin{aligned} V^\pi(s) &= r_t + \gamma r_{t-1} + \cdots = E\left\{ \sum_{i=0}^{\infty} \gamma^i r_{t-i} \,\middle|\, s_0 = s, \pi \right\} \\ V^\pi(s) &= \sum_{a_t} \pi(s_t, a_t) \sum_{s_t, a_t, s_{t+1}} [R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1})] \end{aligned} \tag{2}$$

Additionally, the action-value function is defined as

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) V^*(s_{t+1}),$$

where $Q^*(s,a)$ denotes the expected return of executing action a in state s under the optimal policy π^*. In reinforcement learning, estimating the optimal Q-function (Optimal Q-function in Q-learning Q^*) is equivalent to solving the corresponding MDP [10].

D. Deep Reinforcement Learning

Traditional RL methods face critical limitations when operating in environments with large or continuous state spaces. In a MDP, each state must be explicitly represented, which often leads to a combinatorial explosion as the number of states increases [11]. This renders both storage and computation impractical, particularly in high-dimensional scenarios such as multivariate time series or high-frequency data streams [11].

Here, RL agents struggle to generalize learned behaviors to unseen states, as the policy function $\pi(a|s)$ and value function $V^\pi(s)$ must be defined or estimated for each individual state. This results in high data requirements and extended training times to adequately explore the state space. Moreover, conventional RL may fail to effectively capture complex dependencies among states, leading to suboptimal performance.

To address these challenges, Deep Reinforcement Learning (DRL) leverages deep neural networks to abstract and generalize state representations. Rather than explicitly encoding each state, DRL employs neural architectures to approximate key functions such as the state-value function $V(s)$, the policy $\pi(a|s)$, and the action-value function $Q(s,a)$ [12]. These networks are capable of learning compact, high-level features from raw input data, enabling more efficient policy learning and improved generalization [12].

E. DQN Algorithm – Deep Q-Network

Deep Q-Network (DQN) is a reinforcement learning algorithm that integrates Q-learning with deep neural networks to approximate the action-value function Q [11]. The function $Q(s,a)$ estimates the expected return of executing action a in state s, and is learned through a neural network that replaces the traditional tabular representation.

To enhance training stability, DQN introduces two key techniques: a *target network*, which provides a slowly updated reference for stable temporal-difference targets, and *experience replay*, which stores past transitions and samples them randomly to break temporal correlations. These improvements enable DQN to learn effective policies in high-dimensional and complex environments with large state-action spaces [11].

F. DQN Optimization

Despite its success, Deep Q-Network (DQN) suffers from issues such as temporal overfitting and training instability. Temporal overfitting occurs when consecutive updates to the Q-values are highly correlated due to sequential training transitions, such as (s_t, a_t, r_t, s_{t+1}), $(s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2})$, and so on [12]. These transitions share structural similarities, which can lead the Q-network to overfit specific patterns, resulting in suboptimal convergence [12].

Another issue is the overestimation of Q-values, where the maximum selected value across actions tends to exceed the

true expected return. This bias stems from the maximization step in the update rule, which often favors overestimated actions due to inherent estimation variance [12].

To mitigate these problems, several enhancements have been proposed, including the target network, Double DQN (DDQN), and experience replay. Each technique contributes uniquely to stabilizing and improving DQN performance [12].

G. Target Network

The target network technique addresses instability by decoupling the bootstrapped target computation from the rapidly updating main Q-network. A separate network, referred to as the target network, is periodically updated with the weights of the primary Q-network and used to compute the target values [12]. This reduces learning oscillations and provides a more stable reference for temporal-difference updates.

Formally, the target Q-value is computed as:

$$Q^{\pi}_{\text{tar}}(s,a) = r + \gamma \max_{a'} Q^{\pi}_{\text{target}}(s',a'), \tag{3}$$

where Q^{π}_{target} is the action-value function of the target network.

H. Double DQN

Overestimation bias in standard DQN can degrade learning performance [12]. Double DQN (DDQN) addresses this by decoupling action selection from action evaluation. In DDQN, the main network selects the action with the highest predicted value, while the target network is used to evaluate its value. This separation reduces overestimation and leads to more accurate value estimation [12].

The modified target value in DDQN is given by:

$$Q^{\pi}_{\text{tar}}(s,a) = r + \gamma Q^{\pi}_{\text{target}}\left(s', \arg\max_{a'} Q^{\pi}(s',a')\right) \tag{4}$$

This architecture, illustrated in Fig. 3, improves stability by ensuring that action evaluation is based on a delayed, and thus more stable, estimate.

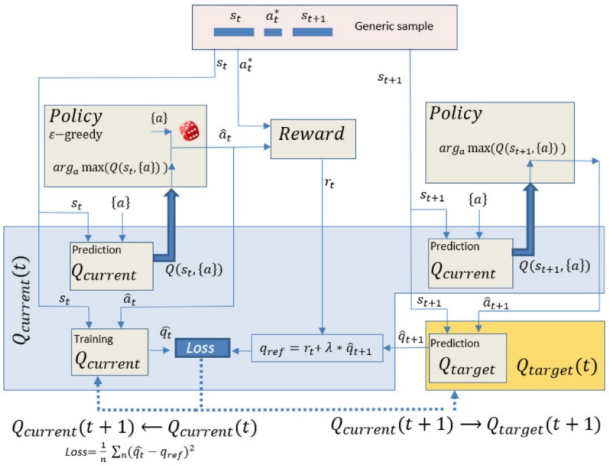

Figure 3. Optimized architecture: Double DQN. Adapted from [13].

I. Experience Replay

Experience replay addresses temporal correlation by storing past transitions (s,a,r,s') in a replay buffer. During training, random minibatches are sampled from this buffer to update the Q-network, which breaks the correlation between consecutive transitions and enables better generalization [12].

The buffer retains the k most recent experiences. When full, the oldest entries are discarded to make room for new ones. During training, one or more minibatches are drawn uniformly and random, then used to update the Q-network parameters.

The experience replay update procedure can be described as follows:

1) Store each transition (s,a,r,s') into the replay buffer.
2) Sample a random minibatch of transitions.
3) Compute the target Q-values for each sampled transition.
4) Update the Q-network using the computed targets.

By learning from a more diverse and decorrelated set of experiences, experience replay significantly enhances training efficiency and stability.

These three techniques—target network, Double DQN, and experience replay—are critical for improving the robustness and convergence of DQN-based models across a variety of RL environments [12].

III. Computer Simulation

A. Dataset Description

The experiments were conducted using the *Numenta Anomaly Benchmark* (NAB), a widely recognized dataset designed to evaluate anomaly detection systems. NAB provides a publicly available repository that simplifies data acquisition and preprocessing, allowing researchers to focus on the development and evaluation of anomaly detection algorithms. It includes diverse and well-documented training and validation datasets, making it a reliable resource for performance benchmarking in anomaly detection tasks [14].

The NAB dataset was introduced in [14] along with an evaluation methodology. It consists of 58 time series signals from real-world and synthetic applications, ranging from 1,000 to 22,000 samples, totaling approximately 365,550 samples. These signals are categorized into the following subsets:

- **Artificial with anomalies**: Six synthetic time series with injected anomalies.
- **Artificial without anomalies**: Five synthetic time series with no anomalies.
- **Real AWS CloudWatch**: Seventeen series from Amazon CloudWatch measuring metrics such as CPU usage and byte input/output.
- **Real known cause**: Seven real-world signals with anomalies of known origin.
- **Real traffic**: Seven traffic-related series collected by the Minnesota Department of Transportation, including vehicle occupancy, speed, and travel time.
- **Real Tweets**: Ten series based on Twitter volume for different companies.

- **Real AdExchange**: Five series containing metrics such as CPC (cost-per-click) and CPM (cost-per-thousand impressions).

Anomaly labels in NAB were manually annotated through visual inspection, following a protocol that imposes the following constraints:

- No anomalies are labeled within the first 15% of each signal, which is used to characterize normal behavior.
- If an anomalous pattern persists beyond 10% of the total signal length, it is no longer considered anomalous after this threshold.
- Anomaly windows are generated around identified anomaly points using:

$$\text{Window Size} = 0.1 \times \frac{\text{Signal Length}}{\text{Number of Anomalies}} \tag{5}$$

This windowing approach can affect sample-level performance evaluation since normal samples within anomaly windows may be incorrectly labeled as anomalous [15].

B. Train/Test Data Partitioning

All data used in this study were sourced from the NAB benchmark. Table I presents the datasets used for training and testing. For synthetic signals (prefixed by 'art'), the *art_daily_jumpsup* signal was used for training, while the remaining artificial datasets were reserved for testing. For real-world signals, training and testing partitions were manually defined, with the first 50% of each series used for training and the remaining 50% for testing.

Table I
DATASETS USED IN THE EXPERIMENTS

Signal Name	Points	Anomalies	Anomaly Windows
art_daily_flatmiddle	4032	402	1
art_daily_jumpsdown	4032	402	1
art_daily_jumpsup	4032	402	1
art_daily_nojump	4032	402	1
Twitter_volume_AMZN	15831	1580	4
RealAdExchange_results	1643	164	3
RealAWSCloudWatch	15831	1580	3
RealTraffic_speed	1127	116	4

C. Evaluation Methodology

The model evaluation follows the F1-score metric, in line with the protocol widely adopted in subsequent benchmark studies such as [16]. The evaluation is based on detecting anomalies in temporal windows, with ground truth labels provided by expert annotation.

Classifications are assessed as follows:

- **True Positive (TP)**: A predicted anomaly window overlaps with a labeled anomaly window.
- **False Positive (FP)**: A predicted anomaly window does not overlap with any labeled window.
- **False Negative (FN)**: A labeled anomaly window is not overlapped by any predicted anomaly window.

Fig. 4 illustrates the classification criteria.

The evaluation metrics are computed as follows:

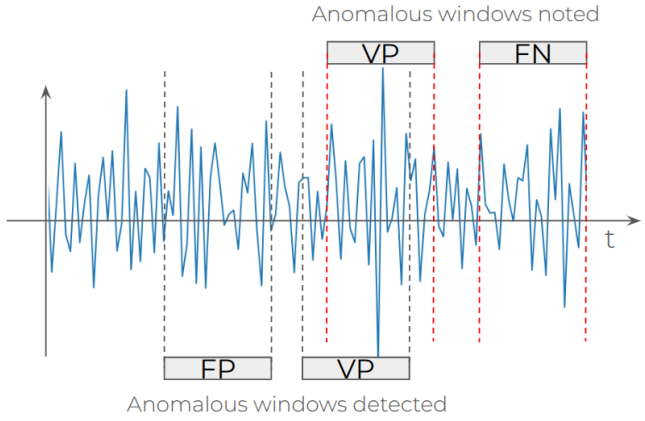

Figure 4. Illustration of TP, FP, and FN classification in anomaly window evaluation.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

This evaluation approach is based on an adapted version of the NAB scoring method, as discussed in [17]. These adaptations aim to reduce evaluation complexity while maintaining performance assessment reliability. Despite known limitations in the NAB documentation, the dataset remains one of the most commonly used benchmarks for anomaly detection in time series analysis.

D. Time Window Construction

To capture the temporal dynamics of the input data, a sliding window strategy was adopted. Each time window was defined as 10% of the total length of the corresponding time series described in Section III-B. These windows include the current observation and its preceding values, providing sufficient temporal context for anomaly prediction. This technique segments the signal into fixed-size subsequences, which are then used as input features. The use of time windows is critical for modeling temporal dependencies and recurring patterns.

E. Conditional Standard Deviation

The conditional standard deviation was computed using the `ARCH` library. Specifically, a GARCH model was employed to capture time-varying volatility patterns often present in financial and non-stationary time series. This model effectively characterizes heteroskedastic behavior, enhancing the extraction of temporal features relevant for anomaly detection.

F. Temporal Derivative

To highlight abrupt changes and significant variations in the signal, the temporal derivative was computed using the `gradient` function from the `NumPy` library. Differentiation provides a measure of how rapidly a signal evolves,

serving as a critical component in identifying dynamic shifts or transient anomalies. This preprocessing step enhances the model's sensitivity to sudden variations, thereby supporting more discriminative learning and improved anomaly detection performance.

G. Data Normalization

Data normalization is essential for ensuring that all input features are scaled uniformly, thus facilitating model convergence and learning efficiency. We applied the `MinMaxScaler` function from the `scikit-learn` library to rescale values to the range [0, 1], as expressed in Equation 9:

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{9}$$

This transformation prevents features with larger magnitudes from dominating the learning process.

H. State and Action Space

In the proposed DRL anomaly detection system, the definition of the state and action space plays a critical role in learning performance. The state space consists of temporally contextualized features derived from the time series. Each state $\mathbf{S}_t$ includes normalized representations of the conditional standard deviation σ_t, temporal derivative $\partial \mathbf{o}_t / \partial t$, and the current observation $\mathbf{o}_t$ over a window $[\mathbf{o}_{t-2}, \mathbf{o}_{t-1}, \mathbf{o}_t]$. Thus, each state vector is represented as $\mathbf{S}_t = [\sigma_t, \partial \mathbf{o}_t / \partial t, \mathbf{o}_t]$.

The action space is binary, with actions $A_t = [N_1, A_2]$ representing the decision to classify the current observation as normal or anomalous, respectively. The agent selects actions based on the current state S_t using a policy π approximated by a neural network. The exploration–exploitation trade-off is managed by exploration rate and decay rate hyperparameters that gradually shift the agent's behavior from exploration to exploitation as training progresses. This balance is crucial for enabling the agent to detect anomalies consistently.

I. Reward Function

The reward function is designed to guide the agent toward correct anomaly classification by assigning positive rewards for true positives and penalties for false positives or missed detections. Specifically:

- A reward of $+10$ is assigned if an anomaly is correctly detected.
- A penalty of -10 is applied if an existing anomaly is missed.
- A penalty of -2 is applied for false alarms (i.e., incorrect anomaly detection).
- A small positive reward χ is granted for correctly identifying a normal sample, where χ is a tunable parameter.

These reward signals are used in conjunction with the Bellman update (Eq. 4) and stored in a replay memory buffer. This mechanism enables the agent to learn from past experiences and refine its policy for more accurate anomaly detection over time.

J. Model Definition: Double Deep Q-Network (DDQN)

The anomaly detection model adopted in this work is based on the Double Deep Q-Network (DDQN) architecture. This model enhances the standard DQN by employing two identical neural networks: the primary (online) network and the target network [12]. The primary network is used to select actions, while the target network evaluates the corresponding Q-values. This dual-network strategy reduces overestimation bias and improves training stability. The training process, which incorporates experience replay, is summarized in Algorithm 1:

Algorithm 1: Training Using Experience Replay

for *each transition* (s_t, a_t, r_t, s_{t+1}) *in minibatch* **do**
  Predict Q-values using the primary network
  Select action a' with highest Q-value from s_{t+1} using primary network
  Evaluate $Q(s_{t+1}, a')$ using target network
  Update Q-value using:
  $Q^{\pi}_{\text{target}}(s_t, a_t) \leftarrow r_t + \gamma Q^{\pi}_{\text{target}}(s_{t+1}, \arg\max_{a'} Q^{\pi}(s_{t+1}, a'))$
  Train the primary network on the updated targets
end

The model integrates an experience replay buffer that stores past transitions (s, a, r, s'), allowing for randomized minibatch sampling. This reduces temporal correlation between samples and introduces diversity into the training process, promoting more efficient learning and robust convergence.

For each minibatch, multiple training iterations are performed to improve the agent's learning from past experiences.

K. Neural Network Architecture

The neural network employed in the DRL model is intentionally lightweight to emphasize the RL mechanics. The architecture consists of two fully connected layers with 64 neurons each, using the ReLU activation function. The output layer has a linear activation with a number of neurons equal to the number of possible actions. The model is trained using the mean squared error (MSE) loss function and the Adam optimizer with a learning rate of 0.001. This architecture ensures computational efficiency and quick integration within the DDQN framework.

L. Complete Algorithm

The complete training procedure for the DDQN-based anomaly detection model is outlined in Algorithm 2. The key hyperparameters used are summarized in Table II.

IV. Analysis and Discussion of Results

The data subsets employed in this study, listed in Table I, were used to evaluate the performance of the proposed DDQN-based anomaly detection algorithm, following the evaluation protocol outlined in the previous section. It is important to note that the initial 10% of each time series was excluded from the final performance assessment. This exclusion is

Table II
HYPERPARAMETERS USED IN TRAINING

Hyperparameter	Value
Window size	10% of signal
State dimension	3× Window size
Training iterations	150 (default)
Discount factor γ	0.9
Initial exploration rate	1.0
Decay rate	0.8
Replay memory size	10,000
Batches per epoch	2
Training iterations per batch	3
Target network update frequency	20 epochs

Algorithm 2: DDQN-based for Anomaly Detection

for *epoch* $= 1$ *to* N_{epochs} **do**
  for *each time step in series* **do**
    Retrieve current state s_t and next state s_{t+1}
    Select action $a_t = \arg\max_a Q(s_t, a)$
    if *random number* $\leq$ *exploration rate* **then**
      Select random action a_t
    end
    Compute immediate reward r_t based on prediction correctness
    if *replay memory is full* **then**
      Remove oldest transition
    end
    Store transition (s_t, a_t, r_t, s_{t+1}) in memory
  end
  for *each batch in replay memory* **do**
    for *each training iteration* **do**
      Sample minibatch
      Compute target Q-values:
      $Q^{\pi}_{\text{target}}(s_t, a_t) = r_t + \gamma Q^{\pi}_{\text{target}}(s_{t+1}, \arg\max_{a'} Q^{\pi}(s_{t+1}, a'))$
      Train primary network with targets
    end
  end
  if *epoch is multiple of target update frequency* **then**
    Update target network weights
  end
  if *exploration rate* $>$ *minimum threshold* **then**
    Decrease exploration rate
  end
end

necessary because the model utilizes a sliding window of 10% for input processing, which inherently prevents anomaly detection during this initial phase. This procedure ensures the reliability and accuracy of the performance metrics discussed and illustrated in this section.

To contextualize the achieved results, the benchmarks provided by [16] are adopted for comparison. This reference compiles a comprehensive set of anomaly detection models evaluated using the NAB dataset, including state-of-the-art approaches developed by reputable institutions.

Table III presents the $F1$-score results obtained from the evaluated subsets (NAB-Art, NAB-AdEx, NAB-AWS, NAB-Traf, and NAB-Tweets). As defined in Eq. (8), the $F1$-score ranges from 0 to 1, where values near 0 indicate poor performance and values close to 1 indicate ideal performance.

The DDQN-based model proposed in this work demonstrates competitive performance across multiple subsets when compared to other state-of-the-art methods. Notably, in the NAB-Art and NAB-Tweets subsets, the proposed approach outperforms existing models, highlighting its effectiveness in detecting temporal anomalies.

An important observation during model evaluation concerns the reward design for true negative predictions. A dynamic reward scheme was adopted, tailored to the empirical characteristics of each dataset. It was observed that the contribution of true negatives to the total cumulative reward becomes more significant in longer signals. Thus, for larger datasets with a higher number of states, the reward for true negatives was reduced to prevent disproportionate reward accumulation. This strategy aligns with the goal of maintaining balance in reward attribution and improving overall model performance.

Table III
PERFORMANCE COMPARISON OF DIFFERENT MODELS ON NAB DATASET (ADAPTED FROM [16])

	NAB Subsets				
Model	**Art**	**AdEx**	**AWS**	**Traffic**	**Tweets**
DDQN[1]	**1.000**	**0.516**	**0.423**	**0.471**	**0.641 ± 0.005**
TadGAN	0.800	**0.800**	0.644	0.486	0.609
LSTM	0.375	0.538	0.474	**0.634**	0.543
ARIMA	0.353	0.583	0.518	0.571	0.567
DeepAR	0.545	0.615	0.390	0.542	0.542
LSTM-AE	0.545	0.571	**0.764**	0.552	0.542
HTM	0.455	0.519	0.571	0.474	0.526
Dense AE	0.444	0.267	0.273	0.412	0.444
MAD-GAN	0.324	0.297	0.230	0.333	0.057
MS Azure	0.125	0.066	0.173	0.166	0.118

A. Analysis of Real-Time Series Detection

Analyzing the real (non-artificial) time series datasets—those with greater practical relevance—offers a robust perspective on the effectiveness of the proposed detection techniques. These signals typically exhibit irregular behavior, making anomaly identification a complex task often requiring expert interpretation.

To visually illustrate the comparative performance of anomaly detection, Fig. 5 presents the actual anomalies within a Twitter-based time series, as annotated by human experts. These annotations serve as a ground truth benchmark to evaluate the accuracy of automated detection models.

Fig. 6 shows the anomalies predicted ($\hat{A}_2$) by the proposed DDQN-based model, which achieved an F1-score of 0.641 ± 0.005, as reported in Table III. By comparing Figs. 5 and 6, one can visually assess the model's accuracy and its ability to detect expert-labeled anomalies. This visual comparison enables an intuitive evaluation of the system's precision, particularly in minimizing false negatives—a capability clearly demonstrated across the analyzed signals.

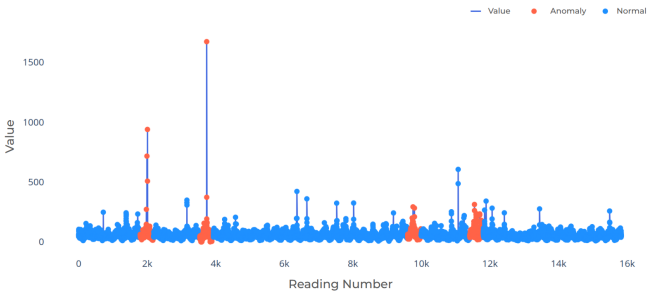

Figure 5. Twitter dataset annotated with ground truth anomalies.

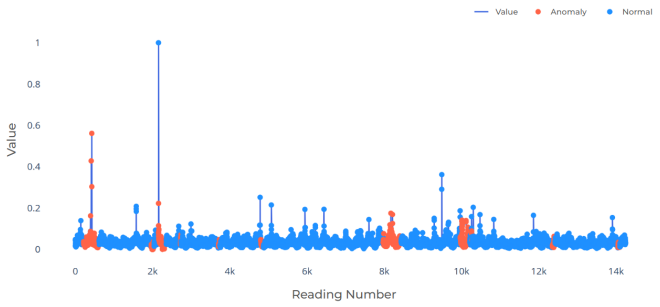

Figure 6. Anomalies detected by the proposed DDQN model.

It is important to highlight that the anomaly detection systems developed in this work were evaluated in highly complex scenarios by leveraging the NAB dataset. The NAB presents realistic challenges due to the presence of noise and irregular patterns inherent to real-world data. This rigorous setup reinforces the significance and applicability of the obtained results in both practical and academic contexts.

The reinforcement learning-based framework proposed herein demonstrates adaptability and learning through environmental interactions. Each correctly classified instance is a step forward in mastering complex patterns. Furthermore, the modular design of the proposed framework provides exceptional flexibility: it not only enables training in controlled environments but also facilitates robust deployment in real-world, noisy, and non-stationary data scenarios.

V. Conclusion and Final Remarks

This study aimed to investigate and apply the Double Deep Q-Network (DDQN) technique for anomaly detection in time series data. The research encompassed the definition and execution of the problem scope, focusing on the implementation of DDQN while addressing the challenges typically associated with anomaly detection, such as limited access to labeled datasets and the inherent complexity of real-world applications. Well-established benchmarks were employed to ensure reproducibility and enable comparative analysis with state-of-the-art approaches. The framework proposed allowed for addressing both theoretical and practical challenges in the application of reinforcement learning for anomaly detection.

The proposed implementation demonstrated competitive performance compared to both traditional and emerging machine learning techniques. Notably, the DDQN model outperformed other methods in detecting anomalies within Twitter datasets, achieving the highest F1-score among evaluated models. These results reinforce the effectiveness and practical potential of the proposed approach.

VI. Acknowledgements

And finally, we would like to thank Murabei Data Science for its financial and computational support.

References

[1] Watson, "Ibm global ai adoption index 2022: New research commissioned by ibm in partnership with morning consult," may 2022, [Online]. [Online]. Available: https://www.ibm.com/downloads/cas/GVAGA3JP

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[3] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, "Bayesian optimization with machine learning algorithms towards anomaly detection," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[4] A. Juvonen, T. Sipola, and T. HäMäLäINen, "Online anomaly detection using dimensionality reduction techniques for http log analysis," *Computer Networks*, vol. 91, pp. 46–56, 2015.

[5] K. Arshad, R. F. Ali, A. Muneer, I. A. Aziz, S. Naseer, and N. S. Khan, "Deep reinforcement learning for anomaly detection: A systematic review," *IEEE Access*, vol. 10, Nov. 2022.

[6] A. V. C. Silva, "Middleware for detecting anomalies in content sharing for blockchain-based systems," Master's thesis, Universidade de São Paulo (USP), 2020.

[7] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Datamining: Data Mining*. Modern sciences, 2009.

[8] K. Boudt, J. Daníelssonb, and S. Laurent, "Volatility models and volatility dynamics." Elsevier, 2013.

[9] M. Yu and S. Sun, "Policy-based reinforcement learning for time series anomaly detection," *Engineering Applications of Artificial Intelligence*, vol. 95, oct. 2020.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. London, UK: MIT press, 2018.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. G. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, jun 2015.

[12] L. Graesser and W. L. Keng, *Foundations of Deep Reinforcement Learning*. Editora Blücher, 2020.

[13] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems," *Expert Systems with Applications*, 2020.

[14] Lavin, Alexander, and A. Subutai, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 38–44.

[15] M. U. BOZZETTO, "Gans for anomaly detection in time series: a case study," 2023, final Course Paper, Universidade Federal do Rio Grande do Sul (UFRGS).

[16] A. Geiger, D. L. andSarah Alnegheimish, and K. V. Alfredo Cuesta-Infante, "Tadgan: Time series anomaly detection using generative adversarial networks," in *IEEE International Conference on Big Data (Big Data)*, 2020.

[17] N. Singh and C. Olinsky, "Demystifying numenta anomaly benchmark," in *Proc. in the International Joint Conference on Neural Networks (IJCNN)*, Anchorage, USA, may 2017.