Performance Analysis of a Reinforcement Learning-Based Negative Selection Algorithm

Dariga Shokayeva - s1158777, Daniel Duhnev - s1158773
Natural Computing at Radboud

Final Report

KEYWORDS

negative selection algorithm, detector generation, self/non-self discrimination, reinforcement learning

Code is available at: GitHub Repository

1 INTRODUCTION

The Negative Selection Algorithm (NSA) is a bio-inspired method for anomaly detection that models the immune system's ability to distinguish "self" from "non-self" patterns[15]. In this project, NSA is applied to anomaly detection using a Unix system calls dataset[8]. It is also applied to text classification tasks, using the AG News[1, 13] and Twitter Sentiment140 datasets[11]. For the AG News experiment, "self" sequences are drawn from a single news category, and detectors are generated to avoid matching these normal patterns. In the case of the Twitter sentiment data, the algorithm is trained to distinguish between binary "positive" and "negative" sentiment tweets.

Traditional NSA can be inefficient and lacks adaptability, especially when generating diverse and effective detectors. To address this, reinforcement learning (RL) is integrated into the detector generation process. The RL agent observes features of the current detector set — such as diversity, coverage, and false positive rate - and adaptively selects among different generation strategies, including random, mutation-based, and coverage-optimized approaches. The agent receives feedback based on anomaly detection performance calculated via the AUC score and learns to optimize detector placement and configuration through trial and error.

It was hypothesised that combining NSA with dynamic, data-driven detector management using RL would improve detection accuracy. This study evaluates whether RL enhances detection performance and discusses the additional computational costs incurred compared to a baseline NSA approach.

2 PROBLEM STATEMENT

While NSA has demonstrated effectiveness in various tasks, it can be inefficient in detector generation and may struggle to adapt to changing environments. Reinforcement Learning (RL), with its ability to learn from trial and error, offers a promising way to address these limitations. This work integrates RL into the NSA detector-generation process to enhance adaptability and efficiency. Specifically, we propose an RL-enhanced framework that dynamically selects among multiple generation strategies - random, mutation-based, or coverage-optimized — based on metrics such as coverage, diversity, and false-positive rate.

The main goal of this study is to evaluate whether this RL-driven approach improves anomaly or classification performance (measured by AUC) when applied to a range of datasets, including Unix system calls (in a security context), AG News (in a multi-class text setting), and Twitter Sentiment140 (in a binary sentiment task). Additionally, we aim to quantify the computational cost of RL-enhanced NSA and assess the trade-off between performance gains and the associated increase in runtime or resources. Furthermore, we seek to identify the ranges of NSA parameters n and r where RL consistently provides a benefit and to determine the scenarios in which this added complexity is worthwhile.

By addressing these goals, this study aims to clarify how RL can make NSA more effective and adaptable for both anomaly detection and text classification, while also considering the practical implications for scalability and computational resources.

3 DATASETS ANALYSIS

In this study, we identified three publicly available datasets suitable for evaluating the proposed RL-enhanced NSA framework across different classification scenarios. The

first dataset selected was the Unix system calls data provided by Stephanie Forrest's group at the University of New Mexico[8], which has been studied and analysed as part of the Natural Computing Master's course at Radboud University. It included a total of 1000 normal examples of system calls which were used for training, as well as a total of 1600 test examples of normal and anomalous calls used for testing.

The second and third datasets focus on natural language processing. On the one hand, the Twitter Sentiment140 dataset was chosen[23] representing a text classification task involving noisy, real-world social media data. It includes 1.6 million tweet examples, each capped at 140 characters — the maximum tweet length at the time. The average tweet length was identified as 14 words or 78 characters[11]. A total of 8,000 randomly selected positive samples were used for training. Additionally, 4,000 samples, evenly split between positives and negatives, were selected for testing. The same dataset was used across all experiments to ensure a fair comparison.

Lastly, we selected the AG News dataset[1, 13] for its well-structured, multi-class news categorization, allowing us to assess the generalizability of our approach across a broader range of textual domains. The dataset is divided into four categories: 'world' 'sports', 'business', and 'science', and it comprises 120,000 and 7,600 evenly distributed training and testing samples. The selected category 'World' includes 30,000 training sequences. These three datasets collectively support a comprehensive evaluation across both security-focused and natural language processing tasks.

4 RELATED WORK

Stephanie Forrest's foundational work on NSAs[9] established their use as one-class classifiers for distinguishing between self and non-self patterns in Unix system call sequences. The experiments demonstrated that short-range correlations in system calls form a stable and efficient signature of *normal* behaviour, capable of detecting several real-world intrusions involving processes such as sendmail by identifying deviations from this learned baseline. Our study aims to extend this approach by using reinforcement learning to dynamically generate and manage detectors that better cover the self-space of the data. The strategy involves training an RL agent that learns to place detectors more strategically by observing performance metrics such as coverage and false-positive rates, in turn improving responsiveness to novel anomalies while retaining computational efficiency.

Furthermore, NSA has been extended to include adaptive mechanisms, such as integrating optimization algorithms[22] or hypercube detectors for better coverage[12]. However, these approaches address adaptability and efficiency within NSA itself, without leveraging RL. The specific integration of NSA and RL remains under-explored and may offer an improvement to tasks utilising the algorithm in various contexts.

On the other hand, RL has been successfully applied to various anomaly detection tasks, especially in cybersecurity and network intrusion detection, demonstrating strong performance across multiple benchmarks[2, 20]. RL has also been successfully implemented in natural language processing tasks. For example, Natural Language Reinforcement Learning (NLRL) redefines RL concepts in a language-based representation space and leverages large language models like GPT-4 to improve interpretability and sample efficiency in sequential decision problems[16]. Moreover, deep learning-based text classification has achieved state-of-the-art results across diverse benchmarks[17] with RL techniques allowing for adaptive policy learning in interactive or sequence-oriented NLP scenarios.

Finally, numerous methods have been proposed for Twitter sentiment classification. For instance, a hybrid stacked ensemble technique was proposed to combine diverse classifiers for handling noisy tweet text and improving robustness[10]. It has also been demonstrated that optimized textual preprocessing pipelines and ensemble ML models can achieve high accuracy in social media news categorization and sentiment analysis[14]. A hybrid gated attention recurrent network (GARN) was developed for Sentiment140, with term-weighting feature extraction and meta-heuristic feature selection integrated to capture temporal dependencies and salient patterns in tweets[19]. However, to date, none of these approaches combine RL with Negative Selection Algorithms (NSA), indicating that this research direction remains under-explored.

5 BACKGROUND IN REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a branch of machine learning in which an agent interacts with an environment, learning to make sequential decisions with the objective of maximizing cumulative reward over time[21]. This section introduces the fundamental RL concepts relevant to the implementation.

5.1 Core Concepts

Agent: The learner or decision-maker.
Environment: The system with which the agent interacts. At each time step, the agent observes the state of the environment, selects an action, and receives a reward and a new state as feedback as presented in Figure 1. **State**: The current

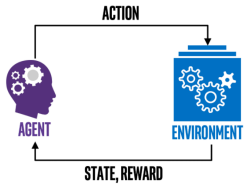


Figure 1: RL Interaction Cycle Overview

situation or configuration of the environment as perceived by the agent.
Action: A choice made by the agent that influences the state of the environment.
Reward: A scalar feedback signal indicating the immediate benefit of the action taken.
Policy: A mapping from states to actions, defining the agent's behavior at any given time. The goal of RL is to learn an optimal policy that maximizes the expected cumulative reward.
Episode: A sequence of agent-environment interactions from an initial state to a terminal state. Each episode allows the agent to gather experience and refine its policy [21].

5.2 Epsilon-Greedy Strategy

A central challenge in RL is balancing exploration (trying new actions to discover their effects) and exploitation (selecting actions believed to yield the highest reward).
The **epsilon-greedy strategy** is a simple and widely used method for balancing exploration and exploitation in reinforcement learning. With tunable probability ϵ, the agent selects a random action (exploration), and with probability $1 - \epsilon$, it chooses the action with the highest estimated reward (exploitation). This approach helps the agent discover better actions while still leveraging what it has learned so far[5].

5.3 Q-Learning

Q-learning is a value-based RL algorithm that estimates the expected cumulative reward (the Q-value) for taking a given action in a given state and following the optimal policy thereafter. The Q-values are updated iteratively using the following rule [21]:

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

where α is the learning rate, γ is the discount factor, r is the reward, s' is the next state, and a' is the next action.
The learning rate (α) controls how much the agent updates its value estimates in response to new experiences. A higher learning rate means faster adaptation but can cause instability, while a lower learning rate leads to slower, steadier updates. The discount factor (γ) ranging 0 to 1 determines how much the agent values future rewards versus immediate ones. A value close to 1 encourages long-term planning, while a value near 0 focuses the agent on short-term gains.

6 METHODOLOGY

6.1 Research Objective

The primary objective of all experiments is to systematically compare the classification performance of standard NSA and RL-based NSA (NSA+RL) across multiple datasets. All methodological choices, parameter settings, and evaluation procedures are designed to ensure a fair and direct comparison between these two approaches.

6.2 Tools and Implementation

Both standard Negative Selection Algorithm and Negative Selection Algorithm with RL-based detector set generation are implemented in Python, utilizing libraries such as *scikit-learn* for computing AUC and generating ROC curves, and *matplotlib* for visualizations. The *random* library is used for detector generation, while file handling uses Python's built-in *os* module. The *numpy* library is extensively used for calculations in Q-learning.

6.3 Pre-Processing

Our approach begins with preprocessing to convert raw input data into sequences suitable for NSA analysis. This process involves reading the input data, extracting sequences, and segmenting each sequence into overlapping character n-grams of fixed length.

The Unix system calls data were stored across multiple files in two folders, namely *snd-cert* and *snd-unm*. The training data comprise normal behaviour sequences, while the test files contain both normal and anomalous system call traces, with corresponding label files specifying 'anomalous' (1) or 'normal' (0) outcomes. The preprocessing function reads each file line-by-line and outputs cleaned sequences with leading and trailing spaces removed. Since these sequences are of variable length, each is segmented into fixed-length chunks by generating every possible substring of a specified length n. This conversion is essential to align the variable-length data with the fixed-length requirements of the Negative Selection Algorithm (NSA).

For the AG News dataset, preprocessing includes reading labelled text from a pandas DataFrame, converting all text to lowercase, and stripping leading and trailing whitespaces. The "World" category was chosen as the self class, and the training data were filtered to retain only those sequences. All other categories were labelled as 'non-self'.

The Twitter sentiment data were noisy, and issues such as non-ASCII characters caused problems during training. To address this, the data were preprocessed with a Python cleaning function designed to handle noisy, informal text. Each entry was first checked for type, with non-string values converted to empty strings to avoid errors. HTML entities were decoded, and URLs and user mentions were replaced with '<url>' and '<user>' placeholders to abstract away specific links or usernames. Hashtags were normalized by converting '#tag' into '<hashtag_tag>' tokens, preserving topical cues without treating them as arbitrary symbols. The regex pattern [^A-Za-z0-9<>\s_] was then applied to replace any remaining unwanted symbols with spaces, leaving only letters, digits, whitespaces, underscores, and angle brackets. Extra whitespaces were collapsed, and the text was lowercased for consistency. Finally, the cleaned tweet strings were segmented into fixed-length chunks via a sliding-window approach of length n, aligning with NSA's fixed-length input requirement. These steps mitigate the high variability and noise in social media text and provide stable, normalized sequences for detector generation.

Across all datasets, training data represent normal behaviour, while test sets include both normal and anomalous sequences with labels indicating 'anomalous' (1) or 'normal' (0). Because sequence lengths vary, they are segmented into all possible substrings of length n to ensure compatibility with NSA's fixed-length input constraint.

6.4 Detector Generation in Plain NSA Algorithm

6.4.1 Overview. In the first NSA phase, we extract all substrings of a specified length n that appear in the training data and then generate candidate detectors by random sampling from the same alphabet. On a high level, each candidate detector is validated to ensure it does not occur in the training set or is not too similar to existing detectors to ensure stronger self-discrimination. Those that satisfy the criteria are collected until a desired number of detectors is reached or until new candidates consistently fail to add diversity based on a threshold value (refer to Section 6.4 for details).

6.4.2 Special Control Considerations. When a candidate is generated, the detector generation implementation can stochastically choose a base sequence from the training data and uses a controlled mutation to produce a candidate. This allows exploring detector space more intelligently compared to pure random sampling. The implementation also calculates a coverage impact by checking how many existing detectors are similar (within the Hamming threshold r) to the candidate. By requiring that the ratio of candidate coverage to the number of existing detectors is below 0.15, it ensures that new detectors are not redundant and genuinely add diversity, thus preventing clusters of highly similar detectors that would leave gaps in the non-self space. A stability check monitors the progression of detector generation and terminates early if no significant diversity is being added to keep it computationally efficient.

The general logic of plain NSA detector generation is presented in pseudocode below (refer to Algorithm 1). The method to mutate a substring that is used in the detector generation is presented separately in Algorithm 2.

6.4.3 Performance Issues with Random Generation. As we learnt from the experiments with the first unsuccessful NSA classifier implementation, randomly generating detectors without such control can lead to significant performance issues. Without filtering, the candidate pool could include detectors that match 'self' patterns. If detectors inadvertently match normal (self) behaviour, the classifier may produce false negatives by misclassifying normal sequences as anomalous, thereby degrading overall accuracy. Equally importantly, not using filtering can result in highly redundant detectors, meaning they are almost identical to one another. This redundancy reduces the overall coverage of the 'non-self' pattern space, causing blind spots where anomalies could go undetected.
These limitations highlighted the critical importance of detector set quality for NSA performance and motivated the exploration of reinforcement learning strategies to adaptively optimize detector generation and address these shortcomings.

6.5 Framing Detector Generation as a RL Problem

6.5.1 Overview. To optimize the detector generation process for the Negative Selection Algorithm (NSA), we framed it as a reinforcement learning (RL) problem using an epsilon-greedy Q-learning approach. In this setup, the RL agent interacts with the environment by selecting among different detector generation strategies - such as random, mutation-based, or coverage-optimized methods - as discrete actions. The environment responds by updating the detector set

Algorithm 1 Generate Detectors with Coverage Monitoring

Require: Training sequences train_sequences, substring length n, Hamming distance threshold r, target number of detectors num_detectors, coverage sample size coverage_samples
Ensure: Set of generated detectors

Extract all self substrings of length n and store in all_self
Extract character alphabet from train_sequences and store in alphabet
Initialize empty set detectors, empty list coverage_history
while |detectors| < num_detectors **do**
  if Random() < 0.3 **then**
    Select random base sequence base from train_sequences
    Generate candidate by calling MutateSubstring(base, n, alphabet)
  else
    Generate candidate by choosing n random characters from alphabet
  end if
  if candidate ∉ all_self **then**
    Compute new_coverage = number of detectors within Hamming distance r of candidate
    if detectors empty **or** new_coverage/|detectors| < 0.15 **then**
      Add candidate to detectors
      Append |detectors| to coverage_history
    end if
  end if
  if |coverage_history| > coverage_samples **and** std. dev. of last coverage_samples < 5 **then**
    break {Stability check}
  end if
end while
return detectors

Algorithm 2 MutateSubstring(base, n, alphabet)

Require: Base string base, substring length n, character set alphabet
Ensure: Mutated substring of length n

if |base| < n **then**
  return random string of length n sampled from alphabet
end if
start ← random integer in $[0, |\text{base}| - n]$
substring ← substring of base from start to start + n
Initialize mutated as empty string
for each character c in substring **do**
  With probability 0.4: select random character from alphabet, else keep c
  Append chosen character to mutated
end for
return mutated

and returning a new state, which is characterized by features like coverage, diversity, false positive rate, and uniformity. The agent receives a reward based on the improvement in anomaly detection performance, measured by the AUC (Area Under the Curve) score on the validation data. Through iterative episodes, the agent learns to select the most effective generation strategies by maximizing cumulative rewards, thereby discovering an optimal policy for constructing high-quality, diverse, and effective detector set. After each episode, if the newly generated detector set yields a higher AUC than any previous set, it is stored as the current best. This RL-based formulation is supposed to address the strong dependency of NSA performance on the quality of the detector set and enable adaptive, data-driven optimization.

6.5.2 Action Space. The action space in this RL formulation is defined as a set of discrete detector generation strategies, each corresponding to a distinct method for constructing detector sets. The implementation specifies three actions:

- **Random Detector Generation:** This strategy samples detector candidates uniformly at random from the full alphabet derived from the training data. Each candidate is checked to ensure it does not match any substring in the self set before being added to the detector pool.
- **Mutation-Based Detector Generation:** This strategy selects a random base sequence from the self (normal) data and applies controlled mutations to a substring of length n. Each character in the substring has a fixed probability (e.g., 0.4) of being replaced by a randomly chosen character from the alphabet. The resulting candidate is included as a detector only if it does not match any self substring.
- **Coverage-Optimized Detector Generation:** This strategy combines mutation and random sampling with an anti-clustering heuristic. Candidates are generated either by mutating self substrings or by random sampling, and are only accepted if they do not match any self substring and if their similarity (measured by Hamming distance within a threshold r) to existing detectors is below a specified anti-clustering threshold. This ensures that new detectors contribute to greater coverage and diversity in the detector set.

At each RL decision episode, the agent selects one of these three actions. The chosen strategy is then used to generate and update the detector set for that episode. This choice of action space aims to enable systematic exploration and exploitation of different detector generation approaches, allowing the RL agent to learn which strategies yield the most effective and diverse detector sets for classification tasks.

6.5.3 State Space. In the RL-based NSA detector generation framework, the state space is constructed to capture the essential properties of the current detector set, enabling the agent to make informed decisions about which generation strategy to apply next. Each state is represented by a feature vector comprising the following technical components:

- **Diversity:** The average Hamming distance between randomly sampled pairs of detectors, normalized by the n-gram length. This quantifies the variability within the detector set, with higher values indicating greater diversity.
- **Non-Self Coverage:** An estimate of how well the detector set covers the non-self space. This is calculated by generating random strings not present in the self set and measuring the fraction that are within the Hamming threshold r of at least one detector.
- **Self-Coverage (False Positive Rate)**: The fraction of sampled self substrings that are incorrectly matched by any detector within the Hamming threshold, serving as an estimate of potential false positive rate.
- **Uniformity:** The uniformity of the detector distribution, measured as one minus the normalized standard deviation of Hamming distances between detector pairs. This feature penalizes clustering and promotes even coverage of the non-self space.

Each feature is discretized into 10 bins per feature. This discretized state representation minimises complexity and allows the RL agent to efficiently update the Q-table during training. By encoding coverage, diversity, false positive rate, and uniformity, the state space provides a comprehensive summary of the detector set's effectiveness and guides the agent toward policies that improve classification performance.

6.5.4 Rationale for State Features. These state features presented above are necessary because they provide immediate, granular feedback on the structure and quality of the detector set, which directly impacts NSA performance. Relying only on reward (AUC score) improvements is insufficient, as AUC offers delayed and sparse feedback and cannot capture intermediate issues like redundancy, clustering, or insufficient coverage. Structural features allow the agent to detect and correct these issues during training, support more stable policy learning, and improve sample efficiency by reducing reliance on computationally expensive full evaluations.

6.6 Classification

Classification remains the same in both NSA and RL-based NSA algorithms. It is done by splitting each test sequence into overlapping n-grams of length n and then comparing each chunk against the detector set. If no detector matches a chunk within the Hamming distance threshold r, that chunk is considered 'unmatched'. The anomaly score for a sequence is calculated as the fraction of unmatched chunks relative to the total number of chunks. This score represents how anomalous a sequence is based on its deviation from normal patterns.

6.7 Performance Evaluation and Visualization

Performance evaluation focuses on determining whether RL-based NSA provides measurable improvements over standard NSA, using consistent metrics and datasets to enable a robust comparison. For both the standard NSA and the RL-based NSA, performance is primarily evaluated using the Area Under the Curve (AUC) metric, which provides a robust measure of the classifier's ability to distinguish between normal and anomalous sequences. An AUC of 1 indicates perfect discrimination, while an AUC of 0.5 reflects random guessing. Receiver Operating Characteristic (ROC) curves[4, 7] are also used to visualize classifier performance by plotting sensitivity (true positive rate) against 1-specificity (false positive rate) across various thresholds, offering insights into the trade-off between detection sensitivity and false alarms.

The evaluation process is different between the two approaches. In the standard NSA implementation, a single detector set is generated, and anomaly scores for all test sequences are computed and aggregated to produce overall AUC results and ROC plots for the dataset.

In contrast, the RL-based NSA introduces an episodic evaluation framework. Multiple detector sets are generated across RL episodes, each guided by learned policies. After each episode, the AUC is computed for the current detector set, and these values are tracked over time to monitor learning progress and policy improvement. Visualization in the RL-based approach includes comparative ROC curves for both the baseline NSA and the RL-optimized NSA.

6.8 Optimization of Parameters

Parameter optimization is essential for maximizing the performance of both standard NSA and RL-based NSA

across diverse classification datasets. In the standard NSA, optimization focuses on grid search over the fixed-length chunk size (n), Hamming distance threshold (r), and the maximum number of detectors. The search is conducted by evaluating predefined sets of n and r values, generating detectors for each combination, and selecting the configuration that yields the highest AUC on a validation set. When searching for the optimal number of detectors, the values of n and r are fixed to those previously found to be optimal, ensuring that only one variable is adjusted at a time. For both NSA and NSA+RL, for all experiments, the number of detectors to be generated is fixed. The particular number is chosen among a preselected options, namely 2k, 5k, 10k and 15k, with the NSA performance under each detector set size being tested on snd-cert dataset. The smallest number of detectors with viable performance is then used for all NSA and NSA+RL methods on all datasets. This simplified approach is motivated by the need to manage computational resources in more complex experimental settings presented in RL approach, while still maintaining competitive detection performance.

In contrast to standard NSA, RL-based NSA introduces a significantly broader hyperparameter space. Beyond n and r, the RL-based approach requires tuning additional parameters such as the learning rate (α), discount factor (γ), exploration rate (ϵ), and anti-clustering threshold. The RL agent iteratively generates detector sets using different strategies, with each episode evaluated based on AUC improvement.

In practice, due to the larger search space and higher computational cost, hyperparameters in RL-based approach are tuned experimentally by varying one parameter at a time while keeping others fixed, rather than performing an exhaustive search. This approach helps identify settings that improve learning stability and detection performance, but the optimal configuration may not always be found due to runtime constraints. As a result, parameter selection in RL-based NSA is guided by iterative experimentation and practical feasibility, rather than fully systematic optimization.

6.9 Pseudocode RL Algorithm Implementation

To clarify the structure of our NSA+RL framework, we present the core algorithm below in Algorithm 3.

6.10 Experimental Setup and Parameter Optimization

For all experiments, both standard NSA and RL-based NSA were evaluated on three datasets: snd-cert, Twitter

Algorithm 3 Reinforcement Learning-based Detector Batch Optimization

Initialize parameters: n, r, episodes, batch size, max batches, target detectors, action size, α, γ, ϵ
Extract alphabet from training sequences
Generate set of "self" substrings from training data
Initialize detector_batches ← empty list
Initialize batch_counter ← 0, best_auc ← 0, best_batches ← empty
for *episode* = 1 **to** episodes **do**
  state ← extract feature vector from detector_batches
  Epsilon-greedy action selection:
  if random() < ϵ **then**
    action ← random integer in [0, *action_size*)
  else
    action ← $\arg\max_a Q[state, a]$
  end if
  Generate new detector batch based on *action*:
  if *action* = 0 **then**
    new_batch ← generate batch with random substrings
  else if *action* = 1 **then**
    new_batch ← generate batch by mutating training substrings
  else
    new_batch ← generate coverage-optimized batch against current detectors
  end if
  Batch management:
  if size of detector_batches < max_batches **then**
    Add (*batch_counter*, *new_batch*, *action*) to detector_batches
    batch_counter ← *batch_counter* + 1
  else
    (*shouldReplace*, *worstIdx*, *improvement*) ← evaluate replacement condition
    if *shouldReplace* **then**
      Replace batch at *worstIdx* with (*batch_counter*, *new_batch*, *action*) in detector_batches
      batch_counter ← *batch_counter* + 1
    end if
  end if
  Enforce total detector count ≤ target_detectors via sampling
  Combine all batches: *combined_detectors* ← $\bigcup$ detector_batches
  auc ← compute AUC on validation/test data
  Calculate reward: *reward* ← $f(auc, best\_auc)$
  next_state ← extract features from updated batches
  Update Q-table: $Q[state, action] += \alpha(reward + \gamma \max_a Q[next\_state, a] - Q[state, action])$
  if *auc* > *best_auc* **then**
    best_auc ← *auc*
    best_batches ← copy of detector_batches
  end if
  Decay exploration: $\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \times decay\_rate)$
end for
return Q-table, detectors combined from *best_batches*

sentiment, and AG News. Across all datasets, the primary parameters optimized were the fixed-length chunk size (n) and the Hamming distance threshold (r). For both NSA and NSA+RL, a grid search was performed over predefined sets of n and r values, with the optimal configuration selected based on the highest AUC achieved on a validation set.

For the snd-cert and Twitter datasets, optimization was limited to n and r in both NSA and NSA+RL. Other hyperparameters, such as the number of detectors or RL-specific settings, were kept fixed at values found to perform well on snd-cert data due to computational constraints and the relatively smaller scale of these datasets.

In contrast, the AG News dataset, being the largest and most complex of the three, allowed for more extensive experimentation. In the AG News experiments, tuning focused on three core RL hyperparameters: the learning rate (α), discount factor (γ), and exploration rate (ϵ). These parameters were selected because they directly influence the Q-learning process. The learning rate (α) controls how rapidly the agent incorporates new information. The discount factor (γ) determines how much the agent prioritizes long-term versus immediate rewards, shaping its strategy for optimizing detector generation over multiple episodes. The exploration rate (ϵ) dictates the likelihood of the agent trying new actions rather than exploiting known good strategies, ensuring sufficient exploration of the action space and helping to avoid suboptimal policies.

Adjusting these parameters individually, while keeping other RL and NSA settings fixed, allowed for a targeted assessment of their specific impact on NSA+RL performance and learning dynamics. This approach was necessary given the high computational cost of RL-based NSA, which made exhaustive hyperparameter search impractical. The chosen values for α, γ, and ϵ reflect commonly used ranges and were selected to balance convergence speed, stability, and overall classification accuracy on the AG News dataset.

6.11 Challenges in Statistical Analysis & Alternative Visualization

Due to the limited number of data points available in this study, a comprehensive statistical analysis would be inconclusive and potentially misleading for making general observations. Therefore, as advised, a scatter plot is instead used to visually represent the available data points, allowing for an presentation of results without over-interpreting patterns that may not be statistically robust.
Additionally, the RL + NSA approach used involves several tunable hyperparameters, the optimal settings of which remain uncertain in our specific context. Given the small sample size, it is particularly difficult to assess whether the chosen hyperparameters are appropriate or if alternative settings might significantly impact the outcomes, further limiting the generalizability of the results presented later in the report.

6.12 RL Approach Expected Behavior Validation

During RL-based detector training on snd-cert dataset, the adjusted script logs key features of the detector set at each episode—including diversity, coverage, false positive rate, and AUC performance—creating a detailed history of the learning process. These feature values are saved to CSV and used to automatically generate plots showing how detector quality and classification performance evolve over time. The resulting visualizations (line plots and correlation heatmaps) confirm whether the RL agent is successfully adapting detector generation strategies, helping to interpret agent learning dynamics and the impact of RL-driven adaptation on anomaly detection performance.

6.13 Consistency Validation

To ensure the reliability of the results, all critical experiments were repeated to check for consistency and rule out the influence of random variation or unknown artifacts in detector generation or classification. For the standard NSA approach, each experiment was rerun three times, and the reported results include the mean and standard deviation of the AUC across these independent runs to reflect both typical performance and observed variability.

For the RL-based NSA, each experiment was repeated at least once; however, it was observed that the RL-based approach exhibited considerable variability between runs, with AUC scores sometimes differing substantially across repetitions. This variability is consistent with findings in reinforcement learning literature, where high variance and sensitivity to initialization, exploration, and stochasticity in training are well-documented challenges [3][6]. The increased computational cost and longer runtimes of RL-based NSA further limited the number of repetitions feasible in practice. As such, results for RL-based NSA should be interpreted with an understanding of this inherent variability.

6.14 Computational Resources and Runtime

All experiments were conducted locally on multi-core CPUs without GPU acceleration. The standard NSA approach typically completed a full experiment including detector generation, classification, and evaluation, within several minutes to under half an hour, depending on dataset size and parameter settings. In contrast, the RL-based NSA approach was substantially more computationally demanding: each RL run, consisting of 50 episodes of detector set generation and evaluation, often required several hours to a full day to complete on the same hardware. This significant increase in runtime is due to the iterative nature of RL training and repeated evaluation steps.

7 RESULTS

7.1 Optimization of the Total Number of Detectors

Basing experiments on snd-cert.1.test test data, we explored the changes in AUC values depending on a number of total generated detectors on a fixed set of n and r values using standard NSA algorithm. Table 1 summarizes the mean and standard deviation AUC values for experiments repeated 3 times under varying number of detectors set sizes of 2000, 5000, 10000 and 15000 detectors. The highest AUC values for each detector count are shown in bold.

		AUC Value Variation on Detectors #			
n	r	2k	5k	10k	15k
4	1	0.92 ± 0.02	0.93 ± 0.03	0.92 ± 0.01	$\mathbf{0.97 \pm 0.02}$
4	2	0.61 ± 0.01	0.61 ± 0.02	0.61 ± 0.01	0.60 ± 0.03
5	1	0.92 ± 0.03	0.92 ± 0.01	0.93 ± 0.02	0.94 ± 0.02
5	2	0.92 ± 0.02	0.91 ± 0.03	0.85 ± 0.01	0.86 ± 0.02
6	1	$\mathbf{0.93 \pm 0.02}$	0.93 ± 0.02	0.92 ± 0.03	0.94 ± 0.01
6	2	0.92 ± 0.02	0.93 ± 0.01	0.93 ± 0.02	0.94 ± 0.03
7	1	0.92 ± 0.01	$\mathbf{0.93 \pm 0.02}$	$\mathbf{0.94 \pm 0.02}$	0.93 ± 0.02
7	2	0.90 ± 0.02	0.91 ± 0.01	0.92 ± 0.03	0.93 ± 0.02

Table 1: NSA parameter optimization results (mean ± std) for varying detector counts on the *snd-cert.1.test* dataset (100 sequences).

All tested detector counts yielded high median AUC values, generally close to or above 0.9, indicating robust performance across parameter settings. For subsequent experiments, a detector count of 2k was selected to ensure efficient runtimes for both NSA and NSA+RL. While this detector count delivers strong detection performance on the relatively structured Unix syscall data, where a smaller alphabet and more constrained patterns allow 2,000 detectors to achieve high AUC values (above 0.9), its sufficiency for more complex and noisy datasets like Twitter Sentiment140 and AG News remains uncertain.

For the richer, more variable textual data where the alphabet size and sequence diversity are larger, it is possible that a higher number of detectors could improve coverage and detection effectiveness. Due to computational and time limits, this study explores experiments conducted with the smallest detector set size sufficient for good performance on the initial Unix syscall data.

7.2 Unix syscalls snd-cert dataset

7.2.1 NSA+RL AUC Results. Due to RL requiring greater computational resources, 2,000 detectors were chosen for the RL experiments. This decision was made because the NSA performed well with 2,000 detectors on the snd-cert dataset, and RL experiments were considerably more time-intensive than non-RL ones. It was observed that an RL run for the snd-cert dataset took on average about 35 minutes to execute for 50 episodes, with total execution times typically varying by no more than a minute between runs. Table 2 summarises the results of these RL experiments.

α	γ	ϵ	AUC	
			Run 1	**Run 2**
0.1	0.8	0.1	0.93	0.92
0.2	0.8	0.1	0.94	0.95
0.1	0.9	0.1	0.94	0.92
0.1	0.8	0.2	0.94	0.93

Table 2: RL-enhanced NSA AUC on Unix syscalls (*snd-cert*), using 2 000 detectors and 50 episodes. Each row is one $(\alpha, \gamma, \epsilon)$ setting with two independent runs.

7.2.2 Feature Evolution During NSA+RL training. To confirm whether this NSA+RL approach operates as expected, we present the set of plots in Figure 2 that illustrates the evolution of key detector set features over the course of RL-based NSA training episodes. Each subplot visualizes how a specific quantitative property - such as detector diversity, coverage, false positive rate, distance uniformity, batch diversity, method balance, classifier performance (AUC), and the total and unique number of detectors—changes as the RL agent learns to adapt detector generation strategies. Trend lines have been added to highlight general trajectories across episodes.

The feature evolution plots indicate that the RL approach is functioning as expected. Over the course of training episodes, several key properties of the detector set—such as diversity, coverage, and batch diversity—show clear trends,

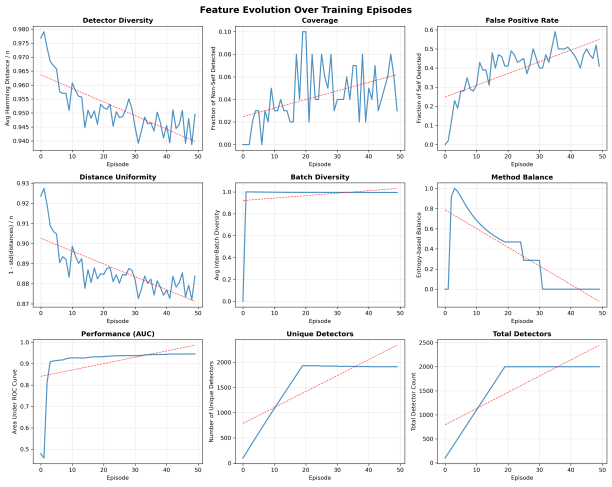


Figure 2: Feature Evolution over 50 episodes of NSA+RL training for the Unix syscalls snd-cert dataset.

reflecting the RL agent's adaptive strategy selection. For example, detector diversity and distance uniformity tend to decrease gradually, while batch diversity and method balance are initially high but stabilize as the policy converges. The AUC performance metric rises quickly and then plateaus, demonstrating that the agent is able to discover and maintain effective detector sets. Importantly, these patterns, along with the rolling action selection probabilities, confirm that the RL agent is exploring different strategies and adjusting its behavior in response to feedback, resulting in the intended adaptation of detector set quality over time.

7.3 Twitter Sentiment140 Dataset

For the Twitter Sentiment140 dataset, we conducted an extensive grid search to identify the optimal values of n and r that maximize the AUC for the NSA classifier. Table 3 shows a summary of the best results found across a number of different combination. By systematically evaluating combinations of these parameters, we were able to select the configuration that provided the highest performance on the more noisy, and informal text dataset.

We used the results from the best value pairs n=8 and r=2 as a baseline against our RL enhanced approach. Each run from the grid search took no longer than 30 minutes to run, with the average run estimated to be 27 minutes. Multiple jobs were spun up to run different ranges of values in parallel and speed up the total runtime.

$n \backslash r$	1	2	3	4
9	0.56±0.01	0.60±0.01	0.60±0.01	0.58±0.02
8	0.58±0.01	**0.61±0.02**	0.59±0.01	0.51±0.02
7	0.59±0.02	0.59±0.02	0.51±0.01	0.49±0.02
6	0.59±0.02	0.56±0.02	0.50±0.02	0.49±0.03

Table 3: Grid search AUC results for the NSA classifier ran against the Twitter sentiment dataset for different (n, r) pairs with a constant detector count of 2000, showing mean ± standard deviation over three runs.

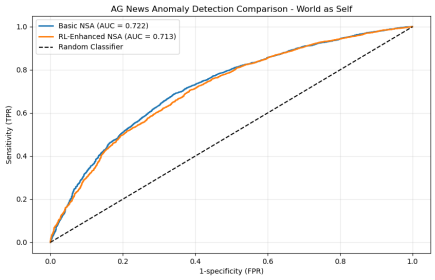
α	γ	ϵ	AUC	
			Run 1	Run 2
0.1	0.8	0.1	0.59	0.60
0.2	0.8	0.1	0.64	0.59
0.1	0.9	0.1	0.62	0.60
0.1	0.8	0.2	0.58	0.62

Table 4: RL-enhanced NSA AUC on Twitter Sentiment140 with 2 000 detectors, 50 episodes, and fixed $n = 8$, $r = 2$. Each row corresponds to one RL hyperparameter setting with two independent AUC runs.

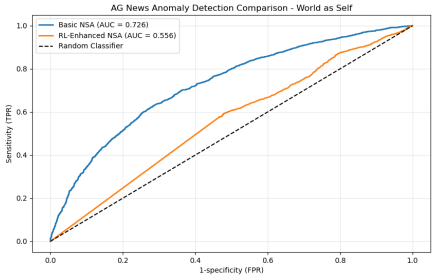
7.4 AG News Dataset

To identify the optimal chunk size (n) and Hamming distance threshold (r) for AG News, the same approach was used as for the Unix Syscall and Twitter Sentiment data. More specifically, a grid search was performed using 8,000 training samples and 2,000 testing samples. For each combination of n and r, detectors were generated from the training subset, and classification performance was evaluated on the testing subset using the AUC metric. $n = 6$ and $r = 2$ were selected based on this grid search, where the simple NSA method yielded an AUC of 0.724 ± 0.002.

Similarly, to evaluate our proposed RL approach, 4 different experiments with varying RL parameters of learning rate, discount factor, and exploration rate were ran. Table 5 presents the AUC results for RL-based NSA on AG News under various RL hyperparameter settings, with $n = 6$ and $r = 2$ fixed throughout. The AUC scores from two independent runs are listed to illustrate the greater variability observed across repetitions (as also shown through ROC curves in Figure 3). On average a full RL run took around 24 hours to complete, which demonstrates the tremendous difference in computational resources and time required to run the algorithm.



(a) 1st run



(b) 2nd run

Figure 3: High variability in NSA+RL AUC scores: NSA vs NSA+RL ROC curves in two runs under the same settings

α	γ	ϵ	AUC	
			Run 1	Run 2
0.1	0.8	0.1	0.71	0.56
0.2	0.8	0.1	0.62	0.72
0.1	0.9	0.1	0.70	0.73
0.1	0.8	0.2	0.72	0.67

Table 5: RL-enhanced NSA AUC on AG News with 2 000 detectors, 50 episodes, and fixed $n = 6$, $r = 2$. Each row corresponds to one RL hyperparameter setting with two independent AUC runs.

Runtimes for experiments on the AG News dataset, when executing plain NSA and RL-enhanced NSA sequentially, took on average a full day to complete for 50 episodes. The plain NSA algorithm was relatively fast, averaging 28 minutes per run and never exceeding 30 minutes. In contrast, most of the total execution time (roughly 24 hours) was due to RL training. This highlights the significant computational resources required to execute the RL algorithm and underscores its infeasibility when applied to large datasets with upward of 10,000 samples.

7.5 Visual Representation of Data Points

Figure 4 provides a consolidated view of classifier performance across our three evaluation domains. For each dataset, the blue circles mark the mean AUC of the standard NSA method (with standard deviation error bars), while the orange diamonds represent individual AUC scores from eight independent RL-enhanced runs under different hyperparameter settings. This layout highlights that as the dataset size increases, the RL-driven detector optimisation exhibits substantial variability, sometimes matching or slightly exceeding the NSA mean, but often falling below it for bigger datasets. By plotting both methods together, we can directly compare their stability and peak performance, and assess whether the additional computational effort of RL yields commensurate gains.

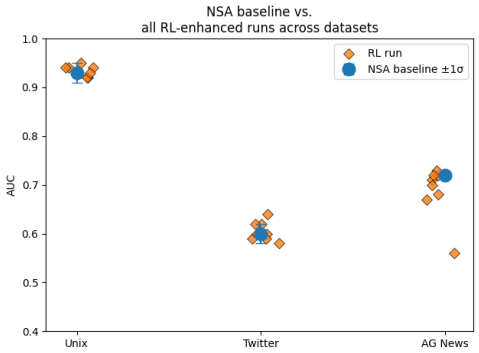


Figure 4: Comparison of NSA baseline AUC (blue circles ± 1σ) versus individual RL AUC runs (orange diamonds) on Unix syscalls, Twitter Sentiment140, and AG News datasets.

7.6 Results Comparison

To gain further insight into whether our RL approach resulted in a performance improvement over simply running NSA, we plotted the best results we obtained across all experiments for each dataset in Figure 5. RL results were significantly more volatile and required more hyperparameter tuning. This comparison highlights the best-case rather than mean result from our experiments and helps us discuss and determine whether the RL approach significantly higher computational load justifies any improvement in the algorithm.

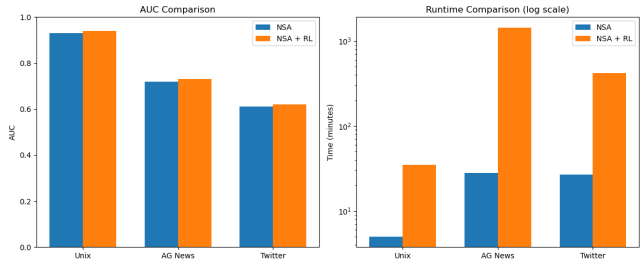


Figure 5: Comparison of AUC and runtime between baseline NSA and RL-enhanced NSA across the single best results for Unix system calls, AG News, and Twitter datasets. Left shows AUC results, while right shows runtime on a log scale.

The comparison highlights not only the differences in AUC scores between the two methods but also the additional computational effort required to achieve these results. This view provides a summary of the trade-offs between performance and training time when applying RL to enhance NSA.

8 DISCUSSION

As shown in Table 1, NSA performs well at distinguishing anomalous Unix system calls. Through a grid search over n, r, and the number of detectors, we achieved a maximum AUC of about 0.97 on the *snd-cert.1* test set with $n = 4$, $r = 1$, and 15k detectors. Out of 72 configurations, 25 (about 35%) reached an AUC of 0.9 or higher, reflecting the classifier's ability to perform strongly when properly tuned. Using this configuration, we achieved an AUC of about 0.93 on the combined *snd-cert* test set. Overall, these results demonstrate that a well-tuned NSA can effectively identify anomalous behaviour in Unix system calls.

For the RL-enhanced NSA on the Unix dataset (Table 2), to make the computations feasible we used 2,000 detectors and 50 episodes per run. The two independent runs for each parameters yielded AUCs comparable to or slightly above the baseline NSA at the same detector count. We determine that RL detector generation strategies may only marginally boost detection performance when computational resources permit.

In contrast, on the Twitter Sentiment140 dataset, the NSA baseline achieved mean AUCs around 0.60 (Table 3), while RL-enhanced runs did not consistently improve over the baseline. This indicates that, for noisy text data like tweets, the simple NSA with optimized (n, r) is as effective as the more complex RL-based detector adaptation. Overall, NSA performed poorly on the Twitter data, and neither the baseline, nor the RL-enhanced algorithm was able to produce a classifier capable of distinguishing between positive and negative sentiment tweets.

Similarly, on AG News, NSA baseline yields stable AUC while RL-enhanced runs show greater variability. Given the very large runtime difference between plain NSA and RL-enhanced NSA, as well as the overall runtime needed for a single RL run, the modest or inconsistent gains do not justify the additional computational cost and an RL-enhanced algorithm is not recommended.

Taken together, these findings highlight that while RL can adapt detector-generation strategies to yield slight gains in domains with clear structure, its benefits are limited for large-scale or highly variable text data under the current formulation. The trade-off between performance improvement and computational expense is especially pronounced. NSA alone achieves competitive AUCs quickly for structured data like Unix system calls, whereas RL demands orders of magnitude more resources for marginal or non-significant gains on text tasks.

Future work could explore more efficient RL formulations or hybrid approaches that combine NSA with lighter-weight optimization methods. Additionally, refining reward signals beyond AUC per episode or incorporating domain-specific features may improve RL effectiveness. However, given the current results, plain NSA with careful parameter tuning remains more practical and efficient choice for large or noisy datasets.

9 LIMITATIONS

This study's RL-enhanced NSA approach is significantly more computationally demanding than standard NSA, making it less practical for large datasets or real-time scenarios. The RL method also showed notable variability in results across runs, reflecting the instability common in RL and making consistent improvements difficult to guarantee. Due to high computational cost, hyperparameter optimization for RL was limited, potentially preventing the discovery of optimal settings. An additional limitation is that the detector count was fixed across experiments to balance runtime and comparison fairness; however, this fixed count may be suboptimal for more complex or variable datasets, potentially restricting coverage and performance.
The design of the RL action and state spaces was kept simple for tractability, but this may have limited the agent's ability to learn more nuanced or effective detector generation strategies. Additionally, the use of tabular Q-learning, while straightforward, is not well-suited for larger or continuous state spaces and may have limited the scalability and expressiveness of the RL approach.

On noisy text data like tweets, statistical analysis found no significant performance difference between standard NSA and RL-enhanced NSA, suggesting that the adaptive benefits of RL are outweighed by the variability in natural language and the high cost of RL exploration. Finally, all experiments were conducted on a limited set of datasets and hardware, so results may not generalize to other domains or larger-scale applications.

10 CONCLUSION

This study investigated whether integrating Reinforcement Learning (RL) into the Negative Selection Algorithm (NSA) could enhance its adaptability and detection performance across a range of datasets. The results show that while RL-assisted NSA (NSA+RL) can outperform plain NSA under certain conditions, these gains come at a substantial computational cost and are not uniformly observed across all datasets.

For the snd-cert dataset, RL improved AUC scores but at the cost of longer runtimes. However, for the Twitter Sentiment140 and AG News datasets, the improvement was less pronounced and often fell within the range of variance, reflecting the algorithm's limited adaptability in more complex and noisy scenarios. Furthermore, RL's training process demanded significantly more computational resources and time - up to nearly 24 hours for AG News, which undermines its practicality for large-scale applications. Furthermore, the grid search and RL hyperparameter tuning highlights that RL's benefits are strongly influenced by proper configuration and may diminish if those parameters are not well-tuned.

Overall, this study shows that RL has little potential to enhance NSA by adding adaptability to detector generation. Its benefits were found to be inconsistent and context-dependent, while also taking considerable computational costs.

11 AUTHOR CONTRIBUTIONS

In Table 6, we present a summary of each author's contributions.

Dariga	Conducted an initial literature review. Implemented the initial NSA algorithm and the modular RL implementation. Produced initial results for the Unix system dataset and saw promising results. Modified the scripts to preprocess and work with the AG News dataset and ran experiments for it. Worked on writing and editing multiple sections in the report.
Daniel	Conducted a further literature review to discuss the field more broadly and how our proposed approach relates to it. Produced further results for the Unix system dataset. Modified the scripts to preprocess and work with the Twitter dataset and ran experiments for it. Worked on writing and editing multiple sections in the report.

Table 6: Author Contributions

REFERENCES

[1] Aman Anand. 2020. AG News Classification Dataset. https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset. Kaggle version of the AG News dataset, labeled into four classes: World, Sports, Business, and Science/Technology.

[2] K. Arshad and A. et al. Muneer. 2022. Deep Reinforcement Learning for Anomaly Detection: A Systematic Review. *International Journal of Computational Intelligence Systems* (2022). https://ieeexplore.ieee.org/document/9956995 Review of 32 DRL-based anomaly detection papers (2017–2022).

[3] Johan Bjorck, Carla P. Gomes, and Kilian Q. Weinberger. 2021. Is High Variance Unavoidable in RL? A Case Study in Continuous Control. *CoRR* abs/2110.11222 (2021). arXiv:2110.11222 https://arxiv.org/abs/2110.11222

[4] Andrew P Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30, 7 (1997), 1145–1159. https://doi.org/10.1016/S0031-3203(96)00142-2

[5] Vahide Bulut. 2022. Optimal path planning method based on epsilon-greedy Q-learning algorithm. *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 44, 3 (2022), 106.

[6] Kaleigh Clary, Emma Tosch, John Foley, and David Jensen. 2019. Let's play again: Variability of deep reinforcement learning agents in atari environments. *arXiv preprint arXiv:1904.06312* (2019).

[7] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27, 8 (2006), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

[8] Stephanie Forrest. [n.d.]. System Calls Dataset. Dataset stored in the syscalls folder and originally collected many years ago by Stephanie Forrest's group at the University of New Mexico. Available at https://www.cs.unm.edu/.

[9] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. 1996. A sense of self for unix processes. In *Proceedings 1996 IEEE symposium on security and privacy*. IEEE, 120–128.

[10] Babacar Gaye, Dezheng Zhang, and Aziguli Wulamu. 2021. A Tweet Sentiment Classification Approach Using a Hybrid Stacked Ensemble Technique. *Information* 12, 9 (2021), 374. https://doi.org/10.3390/info12090374

[11] Alec Go, Richa Bhayani, and Lei Huang. 2009. *Twitter Sentiment Classification using Distant Supervision.* Technical Report. Stanford University. https://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf Original paper describing the Sentiment140 dataset.

[12] Ming Gu, Dong Li, Jia Liu, Wangweiyi Shan, and Shulin Liu. 2024. A negative selection algorithm with hypercube interface detectors for anomaly detection. *Applied Soft Computing* 154 (2024), 111339.

[13] Antonio Gulli and Alessio Signorini. 2005. AG's Corpus of News Articles. http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html. Collected from over 2000 news sources by the ComeToMyHead academic search engine.

[14] Mahmudul Hasan, Tanver Ahmed, Md Rashedul Islam, and Md Palash Uddin. 2024. Leveraging Textual Information for Social Media News

Categorization and Sentiment Analysis. *PLOS ONE* 19, 7 (2024), e0307027. https://doi.org/10.1371/journal.pone.0307027

[15] Rob J. de Boer Judith N. Mandl Inge M. N. Wortel, Can Keşmir and Johannes Textor. 2020. Is T Cell Negative Selection a Learning Algorithm? *Cells* 9, 3 (2020), 690. https://doi.org/10.3390/cells9030690

[16] Yuangang Li et al. 2024. NLRL: Natural Language Reinforcement Learning. *arXiv preprint arXiv:2402.07157* (2024). Redefines RL mechanisms in natural language embedding spaces.

[17] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2020. Deep Learning Based Text Classification: A Comprehensive Review. *arXiv preprint arXiv:2004.03705* (2020). https://arxiv.org/abs/2004.03705

[18] Mike Nkongolo Wa Nkongolo. 2023. News Classification and Categorization with Smart Function Sentiment Analysis. *International Journal of Intelligent Systems* (2023). https://doi.org/10.1155/2023/1784394

[19] Nikhat Parveen, Prasun Chakrabarti, Bui Thanh Hung, and Amjan Shaik. 2023. Twitter Sentiment Analysis Using Hybrid Gated Attention Recurrent Network. *Journal of Big Data* 10 (2023). https://doi.org/10.1186/s40537-023-00726-3

[20] Saba Sanami and Amir G. Aghdam. 2025. Calibrated Unsupervised Anomaly Detection in Multivariate Time-series using Reinforcement Learning. *arXiv preprint arXiv:2502.03245* (2025). https://arxiv.org/abs/2502.03245

[21] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[22] Geying Yang, Lina Wang, Rongwei Yu, Junjiang He, Bo Zeng, and Tian Wu. 2023. A Modified Gray Wolf Optimizer-Based Negative Selection Algorithm for Network Anomaly Detection. *International Journal of Intelligent Systems* 2023, 1 (2023), 8980876. https://doi.org/10.1155/2023/8980876 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/8980876

[23] Μαριος Μιχαηλιδης. 2017. Sentiment140 Dataset with 1.6 Million Tweets. https://www.kaggle.com/datasets/kazanova/sentiment140. Kaggle dataset uploaded by KazAnova.