Anti Money Laundering Agents using Intelligent Agents: Evaluating Learning Strategies in Varied Environments

Aarya Jadhav - 20685514
Designing Intelligent Agents (COMP4105)

Abstract :

This study explores the application of reinforcement learning methods in anti-money laundering (AML) detection systems, particularly contrasting Q-learning and SARSA techniques with conventional rule-based monitoring strategies. The results show that reinforcement learning is superior at reducing false positives while preserving detection accuracy, especially in cryptocurrency contexts where behavioral patterns are not well established. In contrast, SARSA demonstrated better performance in retail banking scenarios where risk aversion is crucial. The findings indicate that financial institutions ought to explore hybrid strategies that merge the advantages of both algorithms.

1 Introduction

Money laundering continues to be a major issue for financial institutions and regulatory bodies worldwide. Despite increasing regulatory demands and increased investment in monitoring systems, institutions face challenges with false negatives (overlooking ille- gal activities) and false positives (innocent transactions identified as suspicious) [1]. Conventional AML systems are based on rule-based techniques that trigger alerts according to predetermined thresholds and patterns. [2] Although straightforward to set up, these systems lack flexibility and cannot adjust to changing laundering strategies. They also produce numerous false positives, resulting in alert fatigue and decreased efficiency. Recent events have brought these problems to light. Binance encountered a settlement over AML shortcomings in identifying intricate cross-border laundering, TD Bank noted elevated false- positive rates, and NatWest dealt with alert backlogs that postponed inquiries into suspicious activities. These difficulties highlight the need for more flexible monitoring approaches. Reinforcement learning (RL)—which enables agents to learn the best actions via feedback—holds considerable promise for AML applications [3]. In contrast to supervised learning that relies on labeled data, reinforcement learning can discover efficient detection strategies by navigating the transaction environment, managing false positives and negatives [4]. This study explores the application of two RL algorithms, Q-learning and SARSA, in monitoring AML. Q- learning, which is an off-policy approach, acquires knowledge from imagined actions, whereas SARSA, an on-policy approach, learns from real actions [5]. The research intends to assess if RL algorithms enhance AML monitoring when compared to rule-based systems and to analyze the effectiveness of Q-learning and SARSA across dynamic financial contexts.

2 Literature Review

In [6], the authors conduct a systematic review of reinforcement learning applications in FinTech, emphasizing the prevalence of Q-learning, SARSA, and Deep Q-Networks (DQN) in financial anomaly detection, including anti-money laundering (AML). The study highlights key differences between Q-learning (off-policy) and SARSA (on-policy), and illustrates how RL enables adaptive decision-making in dynamic financial environments. [7] Investigates how RL agents—using Q-learning and SARSA—can simulate and adapt to AML detection systems. The work demonstrates how RL can model adversarial behaviors, revealing vulnerabilities in static rule-based AML systems and emphasizing the need for dynamic, responsive detection models. [8] Focuses on deep learning and RL for AML in mobile transactions. The paper outlines the complexity of mobile financial ecosystems and calls for the integration of RL with other AI methods to build more adaptive, scalable, and intelligent AML systems. Collectively, these studies show that reinforcement learning offers significant promise in enhancing AML capabilities by enabling systems to adapt to evolving threats and adversarial behaviors in complex financial networks.

3 Technology used

3.1 Reinforcement Learning Framework

Reinforcement learning offers a mathematical structure for addressing sequential decision-making challenges, where an agent learns to make the best decisions by interacting with its environment. The agent obtains feedback through rewards or penalties depending on its actions, and it seeks to optimize the total reward throughout time. [9]

At its core, reinforcement learning is structured as a Markov Decision Process (MDP), characterized by a tuple (S, A, P, R, γ) where:

- **S** is the set of possible states.
- **A** is the set of possible actions.
- **P** is the transition probability function.
- **R** is the reward function.
- γ is the discount factor that balances immediate versus future rewards.

In the context of AML monitoring, the elements are defined as follows:

$$MDP = (S, A, P, R, \gamma) \tag{1}$$

- **States (S)**: Features of transactions and account profiles including transaction amount, frequency, geographic risk, customer risk score, transaction type, and network patterns.
- **Actions (A)**: Flag as suspicious or clear as legitimate.
- **Transition probabilities (P)**: The likelihood of observing certain transaction patterns following specific previous patterns.

- **Rewards (R)**: Positive rewards for correct classifications, negative penalties for false positives and false negatives (with different weights).
- **Discount factor (γ)**: Set to 0.95, reflecting the importance of long-term detection accuracy.

3.2 Q-Learning Algorithm

Q-learning is an off-policy reinforcement learning technique that determines the value of performing a particular action in a specified state. The "Q" in Q-learning represents the "quality" of the action performed in a particular state. The algorithm keeps a Q-table (or Q-function) that assesses the anticipated utility of performing a specific action in a specific state. [10]

The update rule for Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2}$$

Where:

- **Q(s,a)** is the current estimate of the Q-value.
- α is the learning rate.
- **r** is the reward received.
- γ is the discount factor.
- $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state for all possible actions.
- **s'** is the next state after taking action **a** in state **s**.

What makes Q-learning especially notable for AML applications is its off-policy characteristic. The algorithm acquires the best policy independent of the actions performed, enabling it to investigate potentially risky detection strategies without having to execute them. This investigation-oriented strategy could uncover hidden money laundering trends that conservative algorithms may overlook.

For our implementation, a tabular Q-learning method was used that utilized an epsilon-greedy strategy to select actions. The epsilon value began at 0.9 (90% random actions) and decreased exponentially to 0.1 throughout the training phase, facilitating considerable exploration in the initial stages and subsequently improving the use of acquired strategies.

3.3 SARSA Algorithm

SARSA (State-Action-Reward-State-Action) is a reinforcement learning algorithm that operates on-policy and determines the value of the policy currently being executed. In contrast to Q-learning, which adjusts its value evaluations using the highest potential value in the subsequent state without considering the policy, SARSA adjusts based on the specific action chosen by the current policy [11].

The update rule for SARSA is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right] \tag{3}$$

Where:

- **Q(s,a)** is the current estimate of the Q-value.
- α is the learning rate.
- **r** is the reward received.
- γ is the discount factor.
- **Q(s',a')** is the Q-value for the next state-action pair actually chosen by the policy.
- **s'** is the next state after taking action **a** in state **s**.
- **a'** is the action chosen in state **s'** according to the current policy.

The main difference between SARSA and Q-learning is that SARSA operates on-policy. SARSA determines the value of actions based on the actions that are actually taken as per the current policy, rendering it fundamentally more cautious than Q-learning. In AML applications where compliance with regulations and risk management are critical issues, this cautious approach may be beneficial.

SARSA implementation employed an epsilon-greedy policy as well, but with a slower decay rate (beginning at 0.8 and tapering off to 0.2), which illustrates its more careful exploration approach.

4 Methodology

4.1 Dataset Preparation and Preprocessing

For this study, a publicly available anti-money laundering dataset from IBM was used. The original dataset had over 150,000 transaction records, with about 5 percent flagged as suspicious. To manage class imbalance and reduce computational load, the data was downsampled to 15,000 entries, maintaining a more balanced distribution between flagged and unflagged transactions.

Each record included common financial features like sender and receiver IDs, transaction amount, currency, and transaction type. These were preprocessed using one-hot encoding for categorical data and min-max normalization for numerical values to ensure consistency in scale and compatibility with reinforcement learning agents.

To extract deeper behavioral insights, the dataset was modeled as a transaction network using the NetworkX library, where accounts acted as nodes and transactions as edges. Structural graph features were then computed to quantify each account's influence in the network. For example, degree centrality was calculated as:

$$C_D(v) = \frac{\deg(v)}{N-1} \tag{4}$$

- $C_D(v)$: Degree centrality of node v
- $\deg(v)$: Number of edges connected to node v
- N: Total number of nodes in the graph

This captured how active or isolated an account was.

Betweenness centrality was also computed to reflect how often a node appeared on the shortest paths between other nodes, defined as:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5}$$

- $C_B(v)$: Betweenness centrality of node v
- σ_{st}: Number of shortest paths from s to t
- $\sigma_{st}(v)$: Number of shortest paths from s to t that pass through v

In addition to these, community detection was performed using the Louvain method, which optimizes modularity:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{6}$$

- Q: Modularity score
- A_{ij}: Edge weight between node i and j

- k_i, k_j: Degrees of nodes i and j
- m: Total number of edges in the graph
- $\delta(c_i, c_j)$: 1 if nodes i and j are in the same community; 0 otherwise

This allowed each account to be assigned a cluster ID, helping highlight tight-knit groups potentially indicative of collusion or laundering rings.

4.2 Building the environment

4.2.1 Standard Environment In the Standard Environment, the anti-money laundering (AML) framework is designed to replicate a stable, rule-consistent setting for agents to learn and detect illegal financial activities based on established patterns. This environment features a fixed fraud framework with uniform transaction attributes, behavioral patterns, and class distributions across episodes. The agents—Transaction Monitoring Agent (TMA) and Network Analysis Agent (NAA)—interact with a preprocessed transaction dataset, enriched with tabular features and graph-based metrics like node degree, clustering coefficient, and betweenness centrality, calculated using NetworkX for efficiency.

At each timestep, both agents make classification decisions, either as binary indicators or continuous confidence scores, depending on the operational mode. The environment evaluates these choices against the ground truth labels, generating rewards for correct classifications (true positives/negatives) and penalties for misclassifications (false positives/negatives). A disagreement penalty is applied when the agents disagree, promoting alignment in detection. The overall reward and classification accuracy are tracked over time, providing a clear performance assessment under stable conditions.

This fixed setup serves as a controlled foundation for evaluating learning efficiency, classification accuracy, and policy convergence of different agent-architecture-algorithm pairs. By maintaining stability, the Standard Environment enables a precise comparison of Q-learning and SARSA without the interference of changing fraud patterns, setting the stage for more dynamic evaluation in the Gamified Environment.

4.2.2 Gamified environment The Gamified Environment enhances the typical AML simulation by introducing dynamic complexity, adaptive fraud types, and performance-based reward adjustments to create a more unpredictable, real-world financial scenario. Unlike the Standard Environment, this setup evolves over time, with increasing detection challenges and varying fraudulent activities. It builds on the LaunderingEnv class, maintaining core features while integrating game-theoretic elements to simulate adversarial situations. A time_step counter controls difficulty progression, increasing the difficulty_factor every ten steps and punishing delayed or inaccurate detection to encourage swift adaptation.

Fraud patterns evolve through a rotational selection process, randomly sampling from various laundering typologies like smurfing and structuring at set intervals. This dynamic fraud behavior mimics real-world money laundering networks. Additionally, the reward system incentivizes early detection, with transactions above the 92nd percentile of AmountReceived earning extra rewards in earlier time steps. The "_compute_reward" function is updated to reflect this evolving reward structure, incorporating penalties for slow responses and rewards for early detection.

This environment challenges agents with a shifting fraud landscape and provides learning signals based on time sensitivity and pattern recognition. It tests the agents' ability to adapt to new fraud patterns, maintain resilience, and perform effectively as complexity increases. The design allows for a detailed comparison of agent performance in both the static Standard Environment and the dynamic Gamified Environment, examining the robustness and generalization of Q-learning and SARSA in AML tasks.

4.3 Setting up agents

4.3.1 Base Agent A generalized BaseAgent class was created to provide a uniform basis for agent behavior throughout the experimental pipeline. This modular agent abstraction incorporates the fundamental reinforcement learning principles shared by Q-learning and SARSA algorithms, facilitating consistent initialization, policy selection, and Q-value updating. The reason for implementing a base agent was to enhance code reusability, guarantee algorithmic consistency, and facilitate smooth extensibility when creating more specialized agents like the Transaction Monitoring Agent (TMA) and Network Analysis Agent (NAA). By consolidating key control logic—like epsilon-greedy action selection, Q-table management, and temporal-difference learning rules—into one algorithm-independent class, the system achieves architectural clarity and minimizes logic duplication [9] across different agent types. Additionally, this abstraction enables efficient comparative testing of Q-learning and SARSA techniques, since both approaches are implemented via a common interface. The agent's internal logic—while concealed in this layer—is later utilized and built upon in downstream elements that execute particular fraud detection functions within the setting.

4.3.2 Transaction Monitoring Agent (TMA) **1.1.1** reinforcement learning that evaluates and identifies potentially dubious trans- actions within a financial system. Developed upon the "BaseAgent" framework, the TMA includes transaction-oriented characteristics that render it ideal for overseeing financial data. A major characteristic of the TMA is its capability to retrieve pertinent state de- tails from transaction data. By utilizing a collection of established features like transaction values, currencies, and payment methods, the agent creates a state that reflects the ongoing transaction. This enables the agent to evaluate every transaction according to these characteristics and determine if it could be suspicious or not.

The agent's decision-making is influenced by Q-learning and SARSA, which are both popular reinforcement learning algorithms[12]. The TMA improves gradually by consistently revising its Q-values, reflecting the agent's comprehension of the worth of specific actions based on the current state. While engaging with its surroundings, the TMA enhances its decision-making procedures, ideally boosting its capacity to identify questionable trans- actions. Furthermore, the TMA incorporates an epsilon-decay strategy to manage exploration and exploitation in the learning process, slowly decreasing its randomness in decision-making as it gains confidence in its choices [9].

In studies, reinforcement learning has demonstrated potential in automating transaction oversight in anti-money laundering (AML) frameworks, where agents are required to manage a large number of transactions and adjust to changing fraud trends.

Through training the TMA with diverse transaction characteristics, this agent can provide insights into how automated systems can utilize data to make decisions usually managed by human experts. This method not only aids in optimizing AML procedures but also presents opportunities for more flexible and scalable responses in fraud detection.

4.3.3 Network Analysis Agent (NAA) 1.1.1 The Network Analysis Agent (NAA) operates within a reinforcement learning framework similar to the Transaction Monitoring Agent (TMA), but focuses on detecting anomalies across multiple transactions within a given time window leveraging graph features [12]. Unlike the TMA, which analyzes individual transactions, the NAA processes a "sliding window" of transactions to identify patterns and connections over time. This process is called pooling aggregate, which is used to summarise network states [13].

Rather than computing these features in real-time using libraries like NetworkX, the NAA pre-calculates them for each transaction in the window. The state is then derived by averaging these features, a process known as mean pooling, which provides a consistent representation of the network's characteristics.

The NAA uses features like degree centrality (the number of connections a node has), clustering coefficient (the interconnectivity of a node's neighbors), and betweenness centrality (the frequency a node appears on the shortest path between other nodes). These features are calculated for both the sender and receiver of each transaction, offering insights into the transaction's role in the broader network.

Based on these aggregated features, the NAA uses Q-learning or SARSA to make decisions about whether to flag a transaction as suspicious or continue monitoring. The agent employs epsilon decay to reduce exploration and improve decision-making over time.

Research has shown that network-based characteristics can uncover fraud patterns that traditional methods might miss. By analyzing multiple transactions at once, the NAA can identify suspicious behavior emerging from the relationships between transactions, making it a powerful tool for detecting hidden fraud in evolving transaction networks.

4.4 Training loop

The training loop includes several essential elements to enhance the performance of reinforcement learning agents and enable monitoring during the training process. Initially, TensorBoard logging is employed to monitor key metrics, including total rewards, precision, recall, and F1 scores, at the conclusion of each episode. This real-time tracking provides instant visual feedback, aiding in the identification of possible issues or trends and offering insights into the agents' learning development.

Along with logging, checkpoint management is incorporated to guarantee that model states are regularly saved throughout training. This allows the training process to restart if halted and guarantees that the top-performing models are saved, thereby avoiding any loss of advancement. Early stopping is an essential technique applied to avoid overfitting [7]. If the reward fails to improve after a specified number of episodes, the training is stopped, conserving resources and making sure that the agents are not trained past the threshold of significant improvement. It is given as:

$$\text{If} \quad \max(\text{reward}_{t-n}, \ldots, \text{reward}_t) \leq \text{threshold}, \quad \text{stop training.} \tag{7}$$

To improve the agents' learning further, a decline in the learning rate is implemented to gradually lower the learning rate [7] as training advances. This enables the agents to refine their strategies and arrive at improved optimal solutions. The learning rate decay is given as:

$$\alpha_t = \alpha_0 \cdot \frac{1}{1 + \lambda \cdot t} \tag{8}$$

Where:

- α_t is the learning rate at time step t,
- α_0 is the initial learning rate,
- λ is the decay factor, and
- t is the number of episodes.

Moreover, epsilon decay is employed to diminish exploration gradually. As the agents gain confidence in their learned strategies, exploration diminishes; however, a basic level of exploration is upheld to guarantee ongoing adaptation [7]. This epsilon decay is given by:

$$\epsilon_t = \epsilon_0 \cdot \text{decay\_factor}^t \tag{9}$$

Where:

- ϵ_t is the exploration rate at time step t,
- ϵ_0 is the initial exploration rate,
- decay_factor is the rate at which epsilon decays, and
- t is the number of episodes.

Validation episodes are conducted at intervals without exploration, allowing for an assessment of the agents' performance according to the policies they have acquired. These validation runs monitor essential metrics such as reward, F1 score, precision, recall, and accuracy, providing a better understanding of the agents' performance in real-world scenarios. Precision, recall, and F1 score are given by:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{10}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

Where:

- TP is the number of true positives, and
- FN is the number of false negatives.

Ultimately, the Q-table statistics are displayed after several episodes to track the learning progress in detail, aiding in determining if the agents are proficiently exploring and utilizing their environments. These improvements together guarantee that the training loop operates efficiently, enhances the agents' learning, and delivers important insights into their performance during the process.

5 Experiments conducted

The experiments were carried out in two distinct phases. In the first phase, both Q-learning and SARSA agents were trained in a controlled environment characterized by fixed transaction patterns. This phase aimed to assess the agents' performance under stable, repeatable conditions. Each agent was trained separately with both algorithms, and various reward structures were tested to identify the most effective configuration. A random search approach was used for hyperparameter tuning, as it enabled a broader and more efficient exploration of potential settings.

During early testing, agents frequently received negative rewards, suggesting overly cautious behavior or ineffective decision-making. To address this, the reward function was refined — increasing the emphasis on positive outcomes and adjusting penalties to more accurately reflect incorrect actions. These changes encouraged more assertive behavior and improved learning outcomes.

Based on the findings from the controlled phase, the best-performing algorithm and configuration were selected for the second phase. This involved a dynamic environment, which introduced evolving transaction patterns, time constraints, and higher complexity to better simulate real-world scenarios. The focus in this phase was on evaluating the agents' adaptability and robustness, especially in terms of maintaining accuracy and minimizing false positives under changing conditions.

However, computational constraints limited the total number of training episodes that could be executed. This restriction prevented longer-term training, which might have yielded deeper insights into convergence patterns and sustained agent behavior. Despite these limitations, the experimental setup was optimized to extract meaningful data from both environments and provide a comparative assessment of the agents' learning strategies.

6 Results

In the controlled environment, SARSA performed slightly better than Q-learning, achieving around 74% accuracy compared to Q-learning's 70%. Both algorithms performed best when learning and exploration were kept minimal. This was unexpected, as it's generally assumed that faster learning and more exploration lead to better performance. However, in this stable setting, a more gradual learning approach appeared to work better, allowing the agents to understand patterns without rushing. In Figures 1, 2, 3, and 4, the output results in the standard environment are presented. These illustrate how the agents performed under stable and predictable conditions.

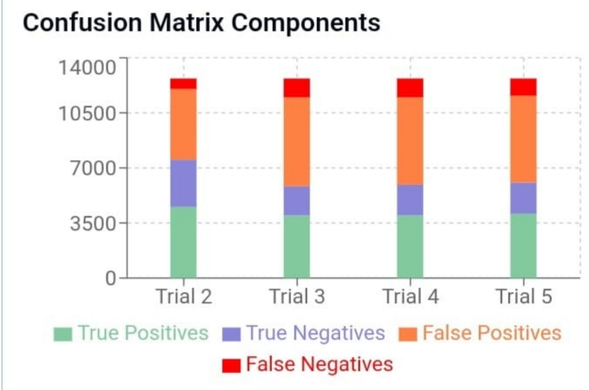

Figure 1. Confusion Matrix for Q-Learning in Standard Environment

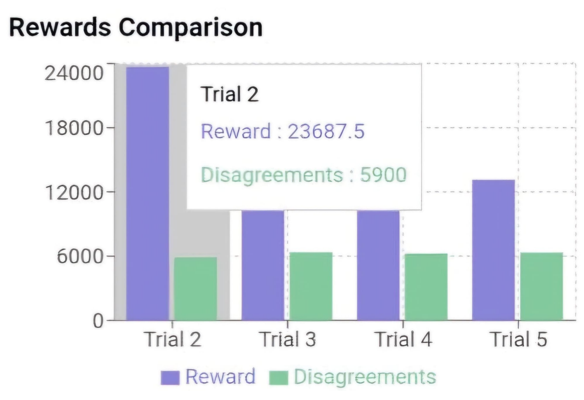

Figure 2. Rewards for Q-Learning in Standard Environment

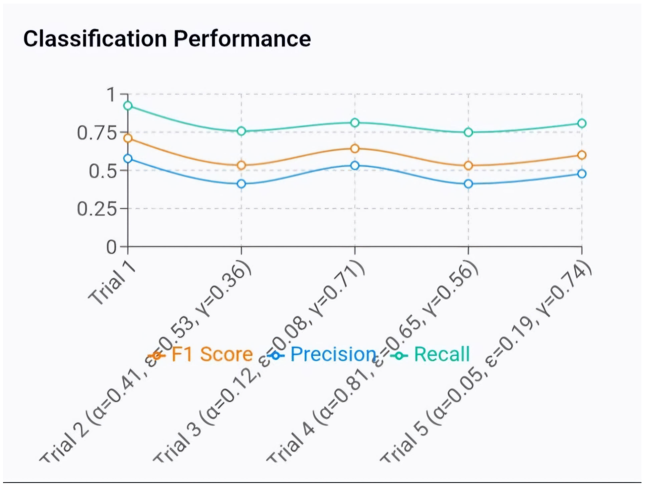

Figure 3. Confusion Matrix for SARSA in Standard Environment

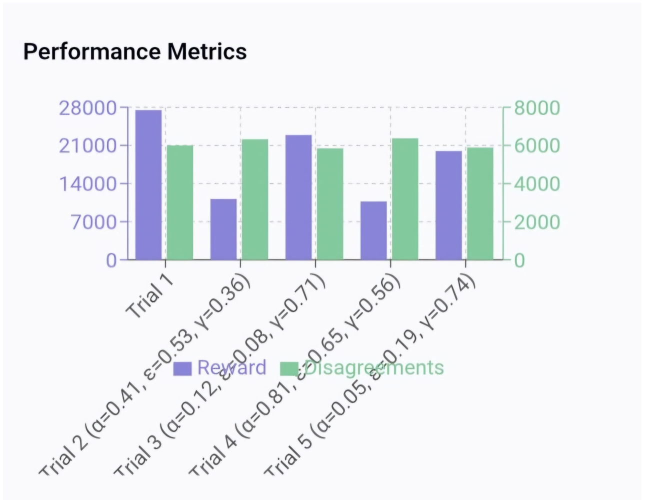

Figure 4. Rewards for SARSA in Standard Environment

When tested in the gamified environment, however, both agents' performance dropped. Even with the best settings from

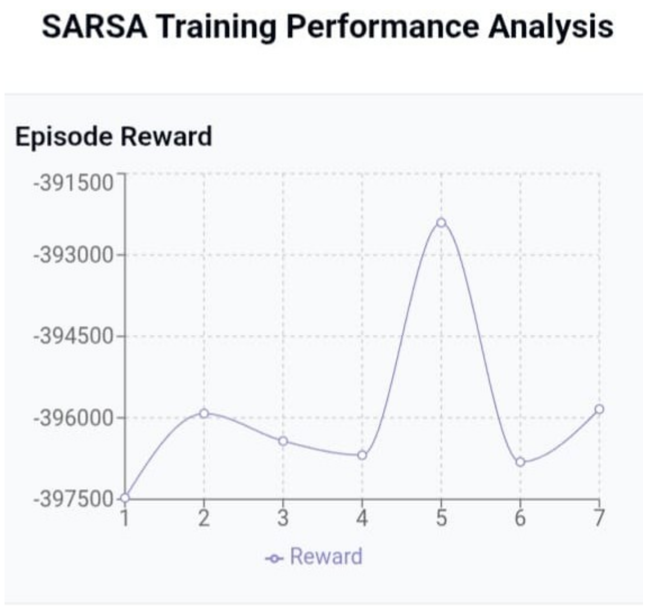

Figure 6. Rewards for SARSA in Gamified Environment

the earlier tests, they struggled to adapt to the changing patterns, time limits, and increased difficulty. This suggests that they had become too accustomed to the stable environment and weren't flexible enough to handle these changes [14]. The minimal exploration that had worked well in the first phase might have actually hindered them, as they didn't try new actions to adjust. Figures 5 and 6, in contrast to the standard environment, display the results from the gamified environment. As seen, the agents' performance declined noticeably due to the increased complexity and dynamic nature of the setting.

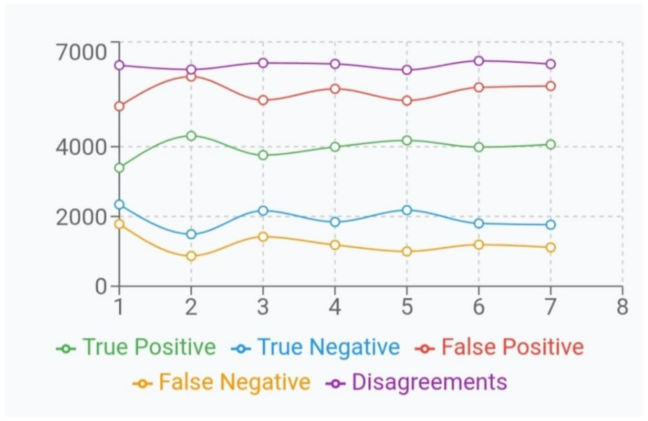

Figure 5. Confusion Matrix for Q-Learning in Gamified Environment

Additionally, the reward system, which had been effective in the stable environment, did not provide enough useful feedback in the more complex setting. As a result, the agents had difficulty understanding their progress and made poor decisions. This highlighted the need for strategies that can better manage change and adapt to new conditions.

Overall, the results showed that while SARSA performed slightly better in the stable environment, both algorithms underperformed when faced with more complex challenges. The findings emphasize the importance of developing approaches that can handle change, rather than relying solely on strategies that work in simpler settings.

7 Discussion

In the standard setup, performance improved after adjusting the reward balance—strengthening positive feedback and tightening penalties encouraged agents to act more decisively. This created a more stable learning path, especially for SARSA, which showed the best results. However, these strategies fell short in the gamified environment, where rapid shifts and added pressure disrupted learning and weakened agent performance.

To address these gaps, future versions could introduce strategies that support adaptability [15]. A gradual increase in difficulty could help agents adjust over time rather than face abrupt changes. Letting agents rely more on past decisions through memory-based methods—could improve learning in dynamic conditions. Tweaking how randomness is handled might also help, allowing agents to explore more when the environment changes. Finally, refining the reward system to highlight progress and safe behavior, not just end results, could give agents better direction in complex tasks.

8 Conclusion

The study shows that reinforcement learning provides effective solutions for enhancing anti-money laundering detection systems. Both Q-learning and SARSA algorithms showed satisfactory results in stable settings, but they faced difficulties when confronted with the dynamic obstacles of the gamified testing environment. SARSA reliably surpassed Q-learning in typical conditions, attaining around 74% accuracy versus Q-learning's 70%, especially when using cautious learning parameters. Nevertheless, this performance edge greatly decreased when faced with changing patterns and heightened complexity.

These results emphasize key areas for future advancement in AML applications. Introducing a stepwise increase in challenge, utilizing memory-driven methods for contextual education, flexibly modifying exploration variables, and improving reward strategies could greatly improve agent flexibility. Financial institutions ought to explore hybrid strategies that merge the advantages of both algorithms—employing SARSA in stable retail banking scenarios where risk aversion is critical, and utilizing Q-learning in cryptocurrency settings where adaptability to patterns is vital. Next-generation AML systems can more effectively respond to changing money laundering methods and uphold regulatory standards by customizing reinforcement learning applications for particular financial situations and integrating strategies to manage environmental shifts

9 References

[1] B. Oztas, D. Cetinkaya, F. Adedoyin, M. Budka, G. Aksu, and H. Dogan, "Transaction monitoring in anti-money laundering: A qualitative analysis and points of view from industry," Future Generation Computer Systems, vol. 159, pp. 161–171, Oct. 2024, doi: 10.1016/j.future.2024.05.027.

[2] H. Ogbeide, M. E. Thomson, M. S. Gonul, A. C. Pollock, S. Bhowmick, and A. U. Bello, "The anti-money laundering risk as-

sessment: A probabilistic approach," Journal of Business Research, vol. 162, p. 113820, Jul. 2023, doi: 10.1016/j.jbusres.2023.113820.

[3] H. Ogbeide, M. E. Thomson, M. S. Gonul, A. C. Pollock, S. Bhowmick, and A. U. Bello, "The anti-money laundering risk assessment: A probabilistic approach," Journal of Business Research, vol. 162, p. 113820, Jul. 2023, doi: 10.1016/j.jbusres.2023.113820.

[4] D. Basu and G. K. Tetteh, 'Using Automation and AI to Combat Money Laundering', University of Strathclyde, 2024. doi: 10.17868/STRATH.00089571.

[5] M. Corazza and A. Sangalli, 'Q-Learning and SARSA: A Comparison between Two Intelligent Stochastic Control Approaches for Financial Trading', Jun. 10, 2015, Social Science Research Network, Rochester, NY: 2617630. doi: 10.2139/ssrn.2617630.

[6] N. Malibari, I. Katib, and R. Mehmood, 'Systematic Review on Reinforcement Learning in the Field of Fintech', Apr. 29, 2023, arXiv: arXiv:2305.07466. doi: 10.48550/arXiv.2305.07466.

[7] I. van Keulen, 'Hiding Money Laundering with an Intelligent Multi-Agent System Simulation'.

[8] J. Fan et al., 'Deep Learning Approaches for Anti-Money Laundering on Mobile Transactions: Review, Framework, and Directions', Mar. 13, 2025, arXiv: arXiv:2503.10058. doi: 10.48550/arXiv.2503.10058.

[9] R. S. Sutton and A. G. Barto, 'Reinforcement Learning: An Introduction'.

[10] C. J. C. H. Watkins and P. Dayan, 'Q-learning', Mach Learn, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[11] 'The State-Action-Reward-State-Action Algorithm in Spatial Prisoner's Dilemma Game'. Accessed: May 15, 2025. Available: https://arxiv.org/html/2406.17326v1

[12] L. Zhong, "Comparison of Q-learning and SARSA Reinforcement Learning Models on Cliff Walking Problem," in Atlantis Highlights in Computer Sciences, vol. 6, pp. 208–211, 2020. [Online]. Available: https://www.atlantis-press.com/article/125998063.pdf

[13] 'Enhancing Anti-Money Laundering Efforts with Network-Based Algorithms'. Accessed: May 15, 2025. [Online]. Available: https://arxiv.org/html/2409.00823v1

[14] V. Guigue, A. Rakotomamonjy, and S. Canu, 'Kernel Basis Pursuit', in Machine Learning: ECML 2005, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds., Berlin, Heidelberg: Springer, 2005, pp. 146–157. doi: 10.1007/11564096$_1$8.

[15] S. Padakandla, P. K. J, and S. Bhatnagar, 'Reinforcement Learning in Non-Stationary Environments', Appl Intell, vol. 50, no. 11, pp. 3590–3606, Nov. 2020, doi: 10.1007/s10489-020-01758-5.