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

Aarya Jadhav- 20685514

May 14, 2025

# 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.

# Introduction

Money laundering continues to be a major issue for financial insti- tutions and regulatory bodies worldwide. Despite increasing reg- ulatory 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).

Conventional AML systems are based on rule-based techniques that trigger alerts according to predetermined thresholds and pat- terns. Although straightforward to set up, these systems lack flex- ibility 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 post- poned inquiries into suspicious activities.

These difficulties highlight the need for more flexible moni- toring approaches. Reinforcement learning (RL)—which enables agents to learn the best actions via feedback—holds considerable promise for AML applications. In contrast to supervised learning that relies on labeled data, reinforcement learning can discover efficient detection strategies by navigating the transaction envi- ronment, managing false positives and negatives.

This study—AML Guardians—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. We believe that these algorithms will ex- hibit varied performance in different financial settings with unique risk characteristics.

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 various financial contexts.

# Technology used

## Reinforcement Learning Framework

Reinforcement learning offers a mathematical structure for ad- dressing sequential decision-making challenges, where an agent

learns to make the best decisions by interacting with its environ- ment. The agent obtains feedback through rewards or penalties depending on its actions, and it seeks to optimize the total reward throughout time.

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, 𝛾) (1)

* **States (S)**: Features of transactions and account profiles in- cluding transaction amount, frequency, geographic risk, cus- tomer 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 pat- terns.
* **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.

## Q-Learning Algorithm

Q-learning is an off-policy reinforcement learning technique that determines the value of performing a particular action in a speci- fied state. The "Q" in Q-learning represents the "quality" of the ac- tion 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.

The update rule for Q-learning is:

𝑄 (𝑠, 𝑎) ← 𝑄 (𝑠, 𝑎) + 𝛼 𝑟 + 𝛾 max 𝑄 (𝑠′, 𝑎′) − 𝑄 (𝑠, 𝑎) (2)

*𝑎*′

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 applica- tions 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 un- cover hidden money laundering trends that conservative algo-

action records, with about 5

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 learn- ing agents.

To extract deeper behavioral insights, the dataset was modeled as a transaction network using the NetworkX library, where ac- counts 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:

deg(𝑣)

rithms may overlook.

For our implementation, I used a tabular Q-learning method

𝐶*𝐷* (𝑣) =

𝑁 − 1

(4)

that utilized an epsilon-greedy strategy to select actions. The ep- silon value began at 0.9 (90% random actions) and decreased expo- nentially to 0.1 throughout the training phase, facilitating consid- erable exploration in the initial stages and subsequently improving the use of acquired strategies.

## SARSA Algorithm

* 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 of-

ten a node appeared on the shortest paths between other nodes,

defined as:

∑︁

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-

𝐶*𝐵*

(𝑣) =

*𝑠*≠*𝑣*≠*𝑡*

𝜎*𝑠𝑡* (𝑣)

𝜎*𝑠𝑡*

(5)

learning, which adjusts its value evaluations using the highest po- tential value in the subsequent state without considering the pol- icy, SARSA adjusts based on the specific action chosen by the cur- rent policy.

The update rule for SARSA is given by:

. .

𝑄 (𝑠, 𝑎) ← 𝑄 (𝑠, 𝑎) + 𝛼 𝑟 + 𝛾𝑄 (𝑠′, 𝑎′) − 𝑄 (𝑠, 𝑎) (3) Where:

* 𝐶*𝐵* (𝑣): Betweenness centrality of node 𝑣
* 𝜎*𝑠𝑡* : Number of shortest paths from 𝑠 to 𝑡
* 𝜎*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 = 1 ∑︁ 𝐴 − 𝑘*𝑖* 𝑘 *𝑗* 𝛿 (𝑐 , 𝑐 ) (6)

* **Q(s,a)** is the current estimate of the Q-value.
* 𝛼 is the learning rate.

2𝑚

*𝑖, 𝑗*

*𝑖 𝑗*

2𝑚

*𝑖 𝑗*

* **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 ac- tions 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.

# Methodology

## 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 trans-

* 𝑄: Modularity score
* 𝐴*𝑖 𝑗* : 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
* 𝛿 (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.

## Building the environment

* + 1. **Standard Environment** In the Standard Environment, the anti-money laundering (AML) framework is designed to repli- cate a stable, rule-consistent setting for agents to learn and detect illegal financial activities based on established patterns. This envi- ronment features a fixed fraud framework with uniform transac- tion 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 be- tweenness centrality, calculated using NetworkX for efficiency.

At each timestep, both agents make classification decisions, either as binary indicators or continuous confidence scores, de- pending 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 un- der stable conditions.

This fixed setup serves as a controlled foundation for evaluat- ing learning efficiency, classification accuracy, and policy conver- gence of different agent-architecture-algorithm pairs. By main- taining stability, the Standard Environment enables a precise com- parison of Q-learning and SARSA without the interference of changing fraud patterns, setting the stage for more dynamic eval- uation in the Gamified Environment.

* + 1. **Gamified environment** The Gamified Environment en- hances the typical AML simulation by introducing dynamic com- plexity, adaptive fraud types, and performance-based reward ad- justments to create a more unpredictable, real-world financial sce- nario. 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 ad- versarial situations. A time\_step counter controls difficulty pro- gression, increasing the difficulty\_factor every ten steps and pun- ishing delayed or inaccurate detection to encourage swift adapta- tion.

Fraud patterns evolve through a rotational selection pro- cess, randomly sampling from various laundering typologies like smurfing and structuring at set intervals. This dynamic fraud be- havior mimics real-world money laundering networks. Addition- ally, the reward system incentivizes early detection, with transac- tions 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 land- scape 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 compari- son of agent performance in both the static Standard Environment and the dynamic Gamified Environment, examining the robust- ness and generalization of Q-learning and SARSA in AML tasks.

## SeEing up agents

* + 1. **Base Agent** A generalized BaseAgent class was created to provide a uniform basis for agent behavior throughout the ex- perimental pipeline. This modular agent abstraction incorporates the fundamental reinforcement learning principles shared by Q- learning and SARSA algorithms, facilitating consistent initializa- tion, policy selection, and Q-value updating. The reason for imple- menting 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 consoli-

dating key control logic—like epsilon-greedy action selection, Q- table management, and temporal-difference learning rules—into one algorithm-independent class, the system achieves architec- tural clarity and minimizes logic duplication across different agent types. Additionally, this abstraction enables efficient compara- tive testing of Q-learning and SARSA techniques, since both ap- proaches 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.

* + 1. **Transaction Monitoring Agent (TMA)** The Transac- tion Monitoring Agent (TMA) is an agent based on 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 charac- teristic 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 algo- rithms. 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 proce- dures, 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.

In studies, reinforcement learning has demonstrated poten- tial in automating transaction oversight in anti-money launder- ing (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 characteris- tics, this agent can provide insights into how automated systems can utilize data to make decisions usually managed by human ex- perts. This method not only aids in optimizing AML procedures but also presents opportunities for more flexible and scalable re- sponses in fraud detection.

* + 1. **Network Analysis Agent (NAA)** 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. Unlike the TMA, which analyzes individual trans- actions, the NAA processes a "sliding window" of transactions to identify patterns and connections over time.

Key to the NAA is the window logic, which aggregates graph- based features for all transactions within the window. Rather than computing these features in real-time using libraries like Net- workX, 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 rep- resentation of the network’s characteristics.

The NAA uses features like degree centrality (the number of connections a node has), clustering coefficient (the interconnec- tivity of a node’s neighbors), and betweenness centrality (the fre- quency 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 de- cay to reduce exploration and improve decision-making over time. Research has shown that network-based characteristics can un-

* + - * 𝜖*𝑡* is the exploration rate at time step 𝑡 ,
      * 𝜖0 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 explo- ration, allowing for an assessment of the agents’ performance ac- cording to the policies they have acquired. These validation runs monitor essential metrics such as reward, F1 score, precision, re- call, and accuracy, providing a better understanding of the agents’ performance in real-world scenarios. Precision, recall, and F1 score are given by:

cover fraud patterns that traditional methods might miss. By an- alyzing multiple transactions at once, the NAA can identify sus- picious behavior emerging from the relationships between trans- actions, making it a powerful tool for detecting hidden fraud in evolving transaction networks.

## Training loop

The training loop includes several essential elements to enhance

Where:

Precision = 𝑇 𝑃

𝑇 𝑃 + 𝐹𝑃

Recall = 𝑇 𝑃

𝑇 𝑃 + 𝐹 𝑁

Precision · Recall Precision + Recall

F1 Score = 2 ·

(10)

(11)

(12)

the performance of reinforcement learning agents and enable monitoring during the training process. Initially, TensorBoard log- ging 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 train- ing. This allows the training process to restart if halted and guar- antees that the top-performing models are saved, thereby avoiding any loss of advancement. Early stopping is an essential technique applied to avoid overfitting. 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:

If max(reward*𝑡* −*𝑛*, . . . , reward*𝑡* ) ≤ threshold, stop training.

(7)

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

1

* 𝑇 𝑃 is the number of true positives, and
* 𝐹 𝑁 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 deter- mining 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.

# Experiments conducted

The experiments were conducted in two phases. In the first phase, both Q-learning and SARSA agents were trained in a controlled environment with fixed transaction patterns. The purpose was to evaluate how well the agents performed in consistent conditions. Each agent was trained separately with both algorithms, and dif- ferent reward structures and settings were tested to find the most effective approach. A random search method proved to be the best way to fine-tune the settings, allowing for a broader exploration of possible options.

During the initial testing, the agents often received negative

rewards, which indicated that they were being too cautious or

given as:

Where:

𝛼*𝑡* = 𝛼0 · 1 + 𝜆 · 𝑡 (8)

not taking enough meaningful actions. To address this, the re- ward system was adjusted to better highlight successful outcomes and refine the penalties for incorrect actions. This change encour- aged the agents to take more decisive actions and improved their

* 𝛼*𝑡* is the learning rate at time step 𝑡 ,
* 𝛼0 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 strate- gies, exploration diminishes; however, a basic level of exploration is upheld to guarantee ongoing adaptation. This epsilon decay is given by:

𝜖*𝑡* = 𝜖0 · decay\_factor*𝑡* (9)

Where:

decision-making.

Based on the results from the controlled environment, the best- performing algorithm and settings were applied to the next phase, which involved a more dynamic environment. This phase intro- duced changing transaction patterns, time limits, and increased complexity to more closely simulate real-world money laundering activities. The goal was to test how well the agents could adapt to changing conditions while maintaining accuracy and reducing false positives.

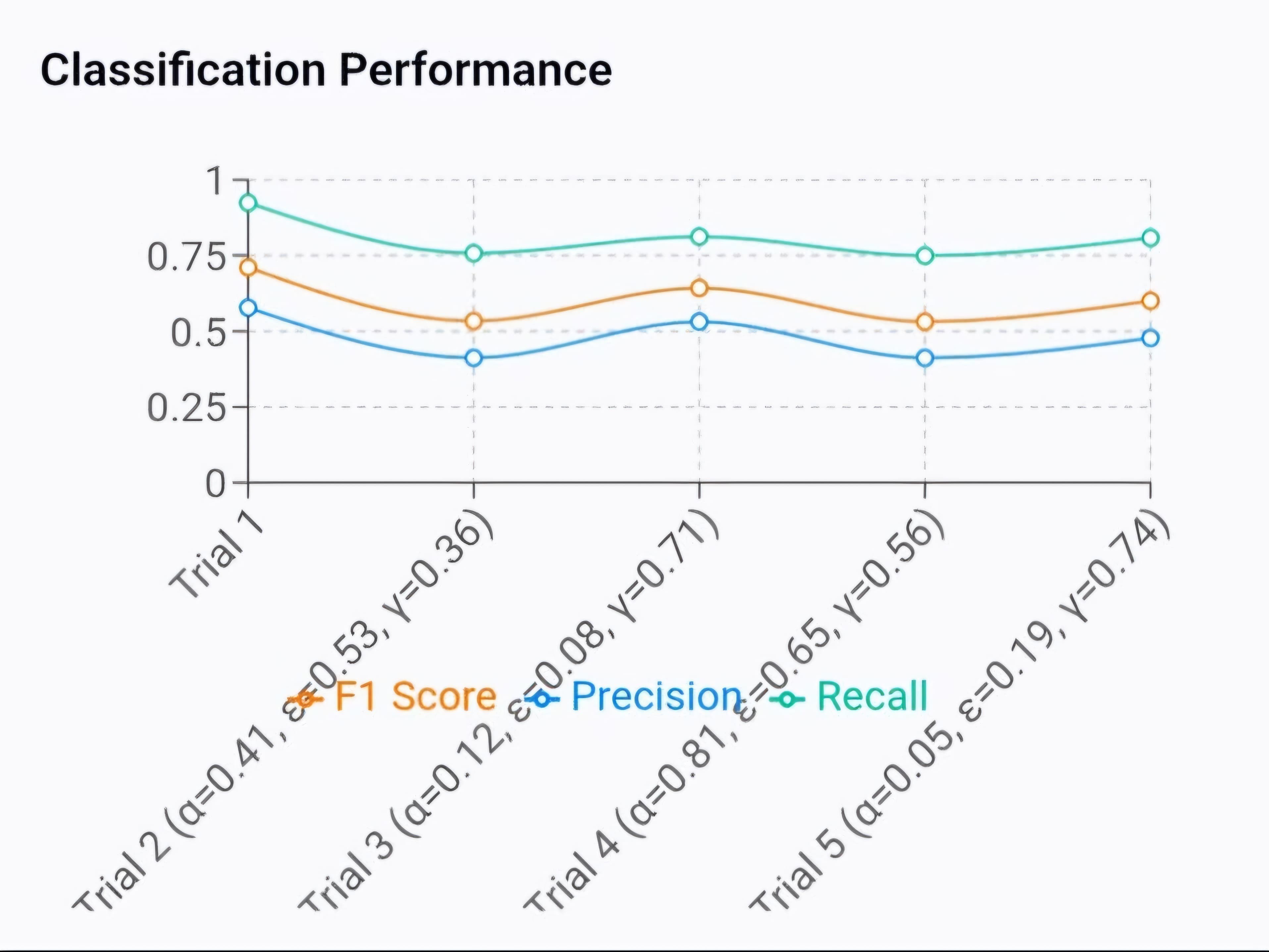
However, due to computational limitations, it was not possi- ble to extend the training episodes beyond a certain length. This restriction limited the agents’ ability to undergo longer training

cycles, which might have provided deeper insights into their long- term performance. Despite this, the experiments were designed to make the best use of available resources while still providing valu- able data on how the agents performed in both stable and dynamic environments.

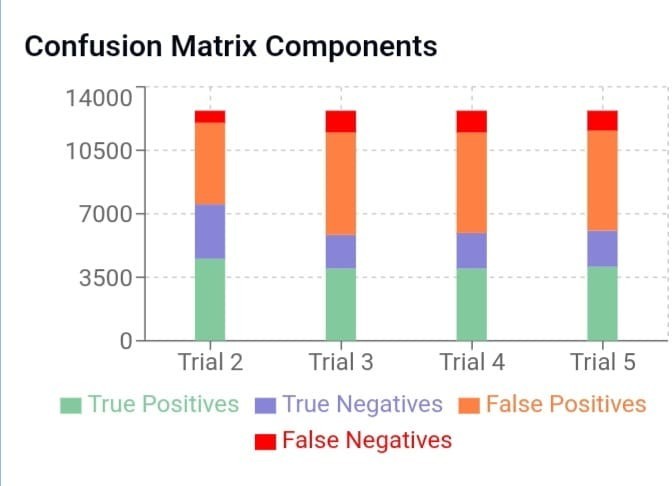
# Results

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 **??**, **??**,

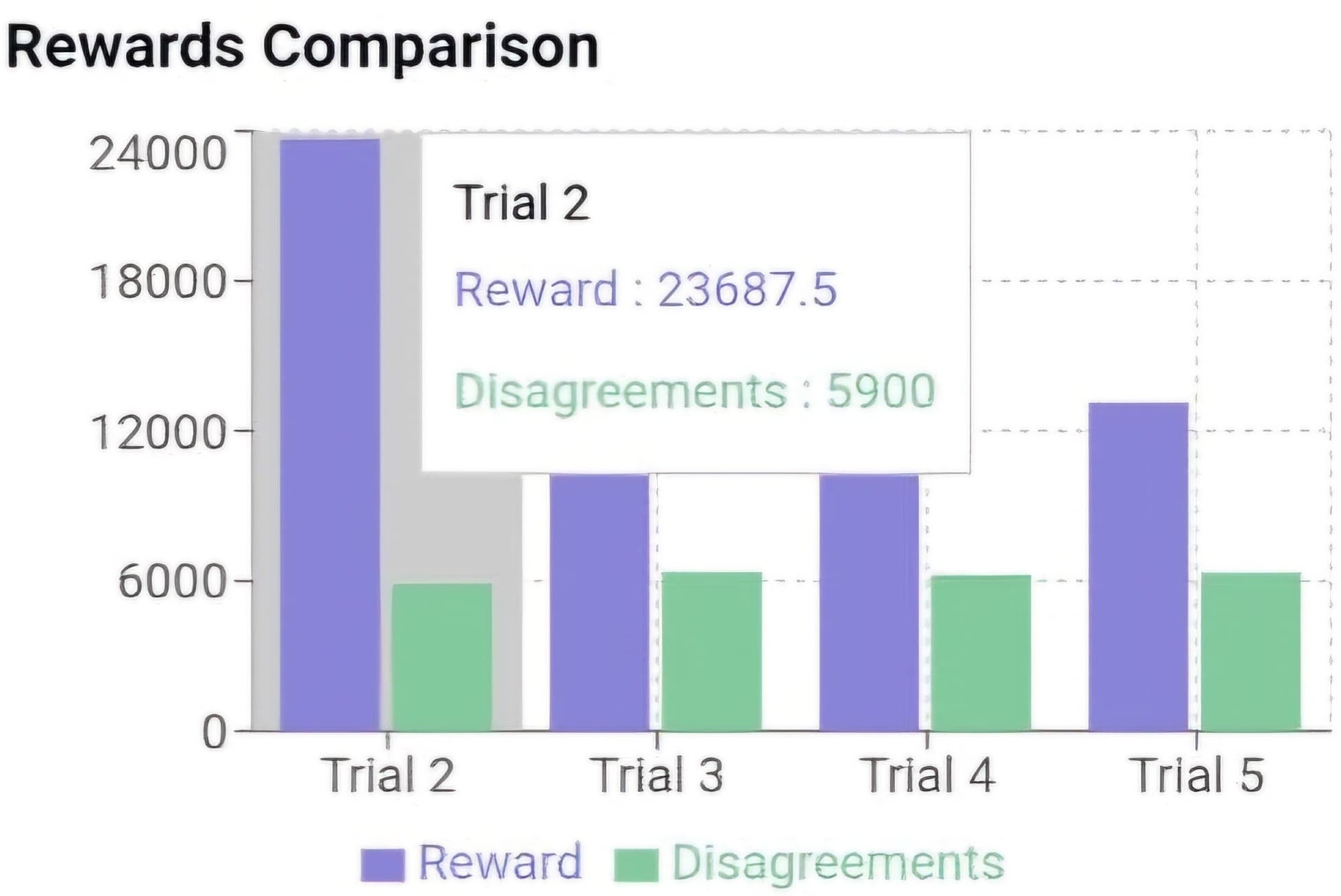
**??**, and **??**, the output results in the standard environment are pre- sented. These illustrate how the agents performed under stable and predictable conditions.



**Figure 3.** Confusion Matrix for SARSA in Standard Environment

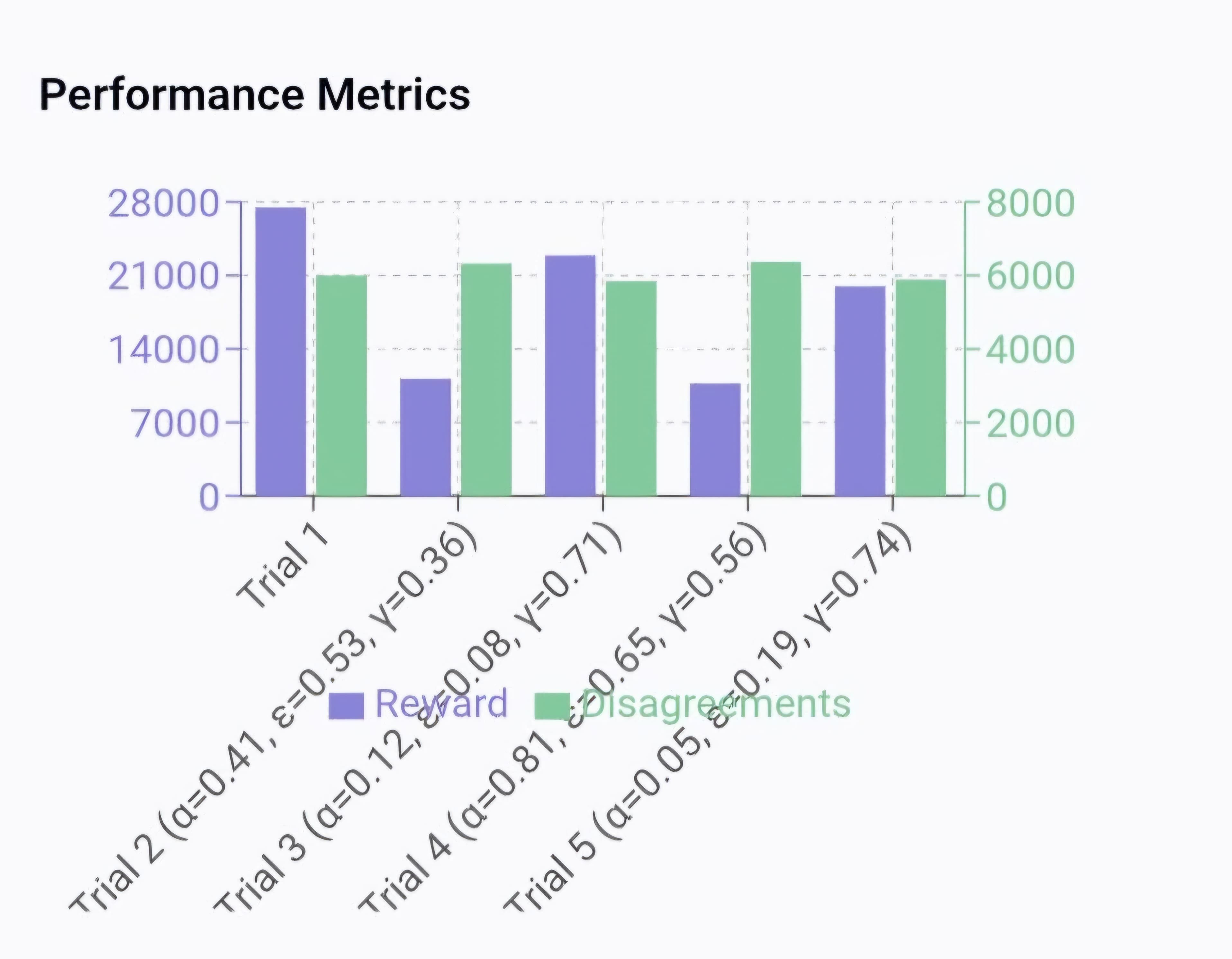


**Figure 1.** Confusion Matrix for Q-Learning in Standard Environment



**Figure 2.** Rewards for Q-Learning in Standard Environment

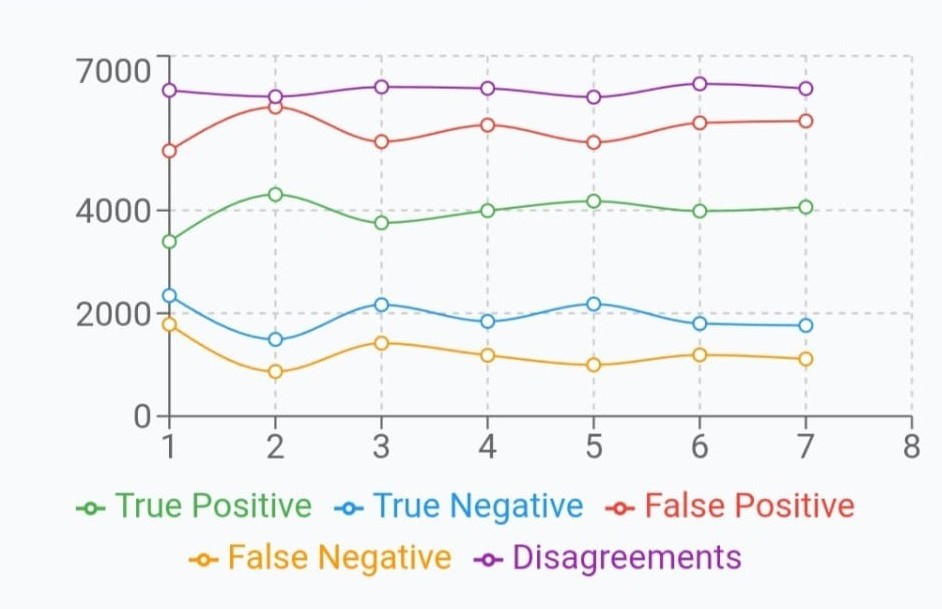
**Figure 4.** Rewards for SARSA in Standard Environment

When tested in the gamified environment, however, both agents’ performance dropped. Even with the best settings from 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. The minimal exploration that had worked well in the first phase might have actually hin- dered them, as they didn’t try new actions to adjust.Figures **??** and

**??** , 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 6.** Rewards for SARSA in Gamified Environment



**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 high- lighted 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 under- performed when faced with more complex challenges. The find- ings emphasize the importance of developing approaches that can handle change, rather than relying solely on strategies that work in simpler settings.

# 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 gami- fied environment, where rapid shifts and added pressure disrupted learning and weakened agent performance.

To address these gaps, future versions could introduce strate- gies that support adaptability. 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. Tweak- ing 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.

# Conclusion

The study shows that reinforcement learning provides effective so- lutions for enhancing anti-money laundering detection systems. Both Q-learning and SARSA algorithms showed satisfactory re- sults 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, attain- ing around 74% accuracy versus Q-learning’s 70%, especially when using cautious learning parameters. Nevertheless, this perfor- mance 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, flexi- bly modifying exploration variables, and improving reward strate- gies 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 sce- narios 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 stan- dards by customizing reinforcement learning applications for par- ticular financial situations and integrating strategies to manage environmental shifts