Scam Token Detection in Memecoin Markets

Suhani Jain, Vardan Vij, Dipit Golechha, Madhavendra Singh

Problem statement:

The primary goal of this project is to explore Deep Q networks to classify if a new Memecoin on the Dex is likely to end in a rug pull scam in **Real time**. We perform a binary classification on memecoin tokens, identifying them as **either legitimate or potential scams** based on their initial characteristics.

A rug pull is a fraudulent maneuver where developers abandon a project after accumulating investor funds, leaving tokens worthless. Over 98% of new tokens on platforms like Uniswap V2 exhibit fraudulent traits.

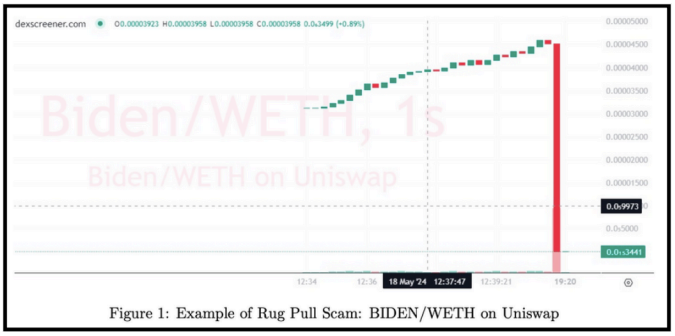


Figure 1: Example of Rug Pull Scam: BIDEN/WETH on Uniswap

Figure 1: An example of a Rug Pull Scam.

Existing tools are mostly *"reactive rather than proactive" (Huynh et al., 2024)* and hence are not able to detect these scam tokens in time, failing to identify scams before liquidity withdrawal occurs.

machine learning algorithm to label tokens as scams. However, the algorithm is only valuable for detecting scams accurately after they have been executed. This paper increases their dataset by 20K

Figure 2: (Mazorra et al., 2022) stating that supervised learning techniques are only valuable AFTER the fact.

The sequential, adaptive nature of rug pulls, where scammers make decisions based on market reactions and timing, suggests that RL is a promising avenue. RL agents learn through interaction and can potentially identify malicious sequential patterns as they emerge, offering a proactive detection capability that supervised methods lack.

Dataset

1. **Initial Real-World Exploration:**

   The process began by exploring real-world data sources via APIs. We used **DexScreener** (for price, volume, liquidity), **Bitquery, Dune and Rugchecker** (for holder analysis, wallet transactions), and **Pump.fun** comments using **Apify** (for social media sentiment), to understand the typical ranges, dynamics, and availability of key features. We decided on these features from what previous literature used. We obtained a dataset from Kaggle that served as a base for our research but lacked features that we wanted to use as our state space, for which we used API calls.

Phase 1: Initial Approach - Static State Classification (Bandit Analogue)

To establish a performance baseline and assess feature importance in a simplified setting, we initially treated the problem as analogous to a multi-armed bandit. Here the state is **static**; it captures only the *current* token's features without historical context within the episode.

State Space: The state (observation) is a **numerical vector** representing the features of a single Memecoin token sample **at time t**. The state represents the information the agent receives to make its classification decision. It's a snapshot of a single token's characteristics **after normalization**. It is randomly selected.

The specific component of the vector are: `Liquidity_lock, top5_holder_pct, top10_holder_pct, volume_spike_ratio, comment_velocity, comment_spam_ratio, hype_score, bundled_rug_selloff, dev_wallet_reputation,wallet_clustering_hh`

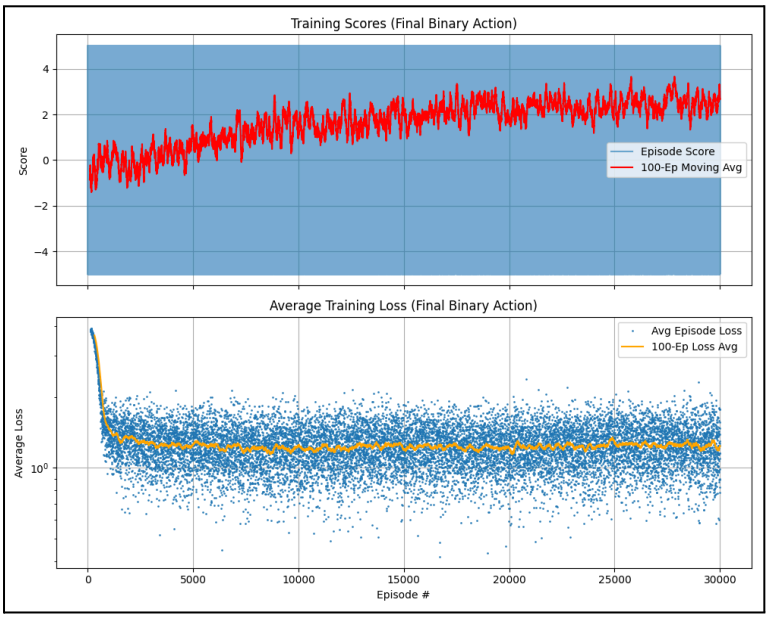
Action space: 0 - Legitimate, 1 - Scam

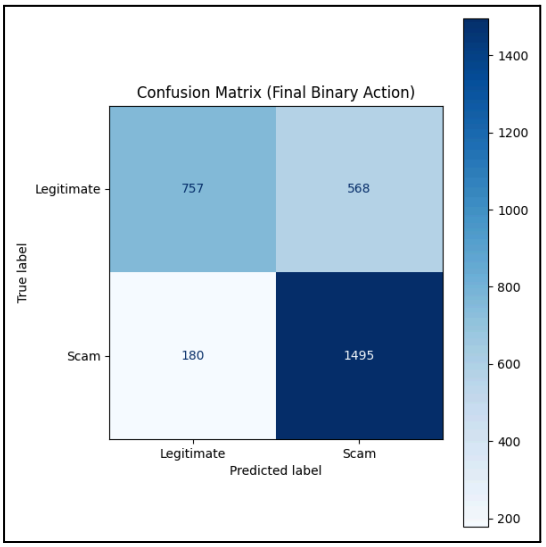
Reward function:

1. Classify Legitimate ($a_t = 0$):
   - $r_t = 2.0$ if the true label is 0 (Legit)
   - $r_t = -5.0$ if the true label is 1 (Scam - False Negative).
2. Classify Scam ($a_t = 1$):
   - $r_t = 5.0$ if the true label is 1 (Scam)
   - $r_t = -5.0$ if the true label is 0 (Legit - False Positive).

Upon taking an action, the agent receives an **immediate reward.** This immediate feedback loop, processed sequentially over the fixed episode length (100 steps), allows the agent to learn from cumulative outcomes.

- **Act**: Select Legit/Scam action via epsilon-greedy on local Q-network's output for current static state.
- **Store**: Receive immediate reward, next state; store transition (s, a, r, s', d) in replay buffer.
- **Learn**: Sample batch from buffer. Update local Q-network by minimizing TD error against Double DQN target (using both local and target networks). Soft-update target network.
- **Repeat**: Continue for many episodes, decaying epsilon.





(fig 1- left, fig 2- right)

The Double DQN agent successfully learned within the static framework, demonstrated by converging loss and increasing average scores *(Fig. 1)*. The resulting ~75% accuracy confirmed baseline predictive capability using the selected features. However, the performance characteristics *(Fig. 2)* reveal a crucial limitation: while scam recall was high (~89%), precision was only moderate (~72%).

This imbalance, **resulting in substantial false positives (568 Legit classified as Scam)**, indicates the agent learned to identify static patterns often associated with scams but lacked the context to differentiate them from transient, legitimate token behavior. Optimizing for immediate rewards on isolated snapshots likely led to an overly sensitive policy unable to account for temporal evolution.

We learnt that we have to prioritise False positives in the rewards function, otherwise it is not detecting properly. Hence we turned the reward function back to being equal for both FN and FP.

Phase 2: Delayed Rewards - RL formulation

While Phase 1 validated the features and basic learning, its static snapshot approach inherently ignored crucial temporal dynamics (trends, velocity changes) vital for real-world token assessment. Classifying isolated data points misses sequential patterns and fails to leverage RL's core strength in handling delayed consequences over time. Therefore, to capture these essential dynamics and model the token launch lifecycle more realistically, we transitioned to a full time-series RL formulation.

State Space(S)

The state (observation) represents the token's condition **at a specific time_step t within its historical sequence.**

The specific component of the vector for each time step remain the same: `Liquidity_lock, top5_holder_pct, top10_holder_pct, volume_spike_ratio, comment_velocity, comment_spam_ratio, hype_score, bundled_rug_selloff, dev_wallet_reputation,wallet_clustering_hh`

The agent receives these state vectors **sequentially** as the environment progresses through the token's timeline, enabling the policy network to learn from the **evolving temporal dynamics** represented by this sequence of observations.

Action Space (A)

- 0: Observe/Wait (Advances environment to time_step t+1)
- 1: Terminate & Classify Legitimate (Predicted Label = 0).
- 2: Terminate & Classify Scam (Predicted Label = 1)

Reward Function (R)

Delayed reward R(s_t, a_t) issued only at episode end.

- **Intermediate Reward** (r_t for t < T): For all steps before the terminal step T, the reward is zero (r_t = 0). The agent receives no immediate feedback on its assessment during the observation period.
- **Terminal Reward** (When the episode ends (done = True), the agent receives a reward based on its final action (a_T) compared to the true ground truth label (derived from the 'label' column for that 'token_id'). The 'true_label' is 0 for Legit, 1 for Scam.
    - Ifa_T == true_label: Agent's final assessment was correct. $R_{correct}$ = **+1**.
    - If a_T != true_label: Agent's final assessment was incorrect. $R_{incorrect}$ = **-1**.

- If a_T = 0 leads to truncation (max steps reached): R = truncation penalty = **-0.5**

State Transitions & Termination: Implicitly defined by step(a_t):

- If a_t = 0: Transition s_t -> s_{t+1} by incrementing 'time_step'. If max steps are exceeded, episode truncates.
- If a_t = 1 or a_t = 2: Transition s_t -> Terminal State. Episode terminates.

Environment

An episodic environment processing time-series data (token_id, time_step) from pre-processed Parquet files. Each episode corresponds to a single token's history.

Phase 2 Methodology

Objective: Maximize cumulative reward by learning both *what* to classify and *when* to classify confidently.

Algorithm Choice (PPO): PPO was selected for its balance of sample efficiency and training stability, crucial for navigating the potentially complex dynamics of time-series financial data. Its clipped surrogate objective prevents destructive policy updates, facilitating more robust learning compared to simpler policy gradient methods when trying to learn subtle temporal patterns.

Core Architecture:

- **Policy Network (π(a|o)):** An MLP taking the current scaled feature vector (observation o) as input, outputting probabilities for actions (Observe, Classify Legit, Classify Scam).
- **Value Network** : A parallel MLP estimating the expected discounted future return from the current observation, guiding the policy's decisions.

Learning Dynamics - Learning *When* and *What*: The Agent learns implicitly through trial-and-error, driven by maximizing expected discounted reward ($\Sigma\gamma^t * R_t$).

- **Advantage Estimation (GAE)**: Learning hinges on the advantage estimate

  $A \approx \Sigma(\gamma\lambda)^{\delta\_\{t+l\}}$ where $\delta_t = R_t + \gamma V(o_{t+1}) - V((o_t))$. This measures how much better taking a specific action was compared to the baseline expectation (V(o)).

- **Timing Decision** : The agent learns to Observe if the current features suggest that future observations are likely to yield a higher expected return (captured by V) than the immediate reward from classifying.

  The Value Network learns to associate ambiguous or early-stage feature patterns with the potential value of waiting. Conversely, it learns to Classify when observed feature patterns strongly predict a specific outcome, making the expected immediate reward (+1/-1) outweigh the estimated value of continued observation (considering GAMMA and the 'truncation_penalty').

- **Classification Logic (Implicit Pattern Recognition)**: PPO does not learn explicit rules. Instead, through policy gradient updates guided by the advantage, it reinforces connections between specific feature sequences (e.g., patterns indicative of liquidity issues, holder concentration changes, sentiment shifts – potentially signaling a "rug pull") and the action (Classify Scam or Classify Legit) that historically led to positive rewards (+1) from those states.

Reward Shaping: The sparse, delayed reward structure (+1/-1 only upon classification) combined with the truncation penalty (-0.5) is critical. It forces the agent to optimize the timing of its decision, balancing the risk of

misclassification against the penalty for indecision, thereby learning to act only when sufficiently confident based on the observed temporal feature dynamics.

Challenges and Learnings while writing the code:

- Challenge: Agent Never Committing: Initially, without a truncation penalty, the agent learned to always Observe to avoid the risk of a -1 reward from misclassification, resulting in zero progress… **Introduced the *truncation_penalty* (-0.5).** This made infinite observation slightly suboptimal compared to eventually attempting a classification, forcing the agent to learn a decision point.
- Challenge: Agent Guessing Too Early: When the classification reward (+1/-1) was the only non-zero signal, the agent sometimes made hasty classifications on minimal data, essentially guessing… **Keep the Observe action reward at 0.** This ensures there's no immediate incentive to stop observing. The agent must implicitly learn that more observation might lead to a higher probability of the +1 reward later, balancing delay against confidence. The PPO algorithm's value function learns to estimate this future potential.
- Challenge: Difficulty Tuning Patience: Finding the right balance between encouraging timely decisions (via truncation_penalty) and allowing sufficient observation was difficult. Too harsh a penalty caused early guessing; too lenient caused truncation…. **Experimented with different truncation_penalty values.** Set GAMMA (discount factor) appropriately (e.g., 0.99) so future classification rewards aren't overly discounted, encouraging some patience. The *ppo_delayed.py* script centralizes these hyperparameters for tuning.

Results and discussion:

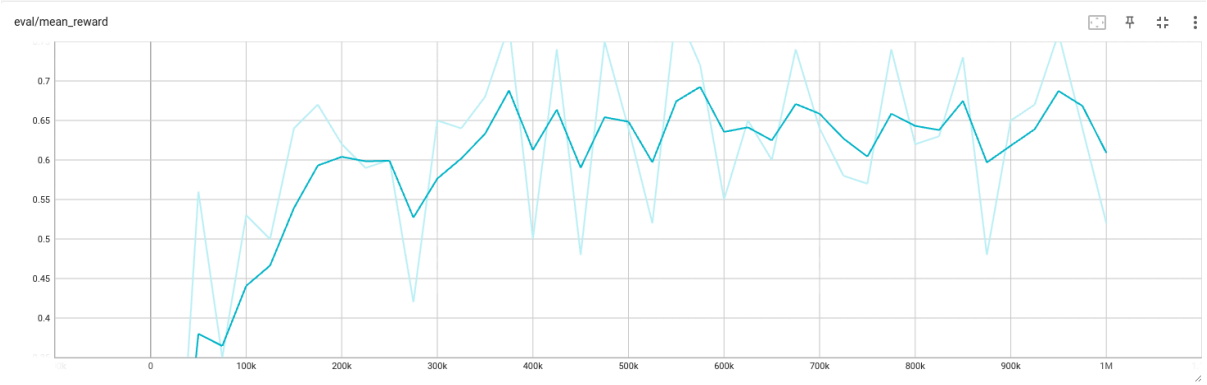


Figure 3: Average rewards per episode during evaluation

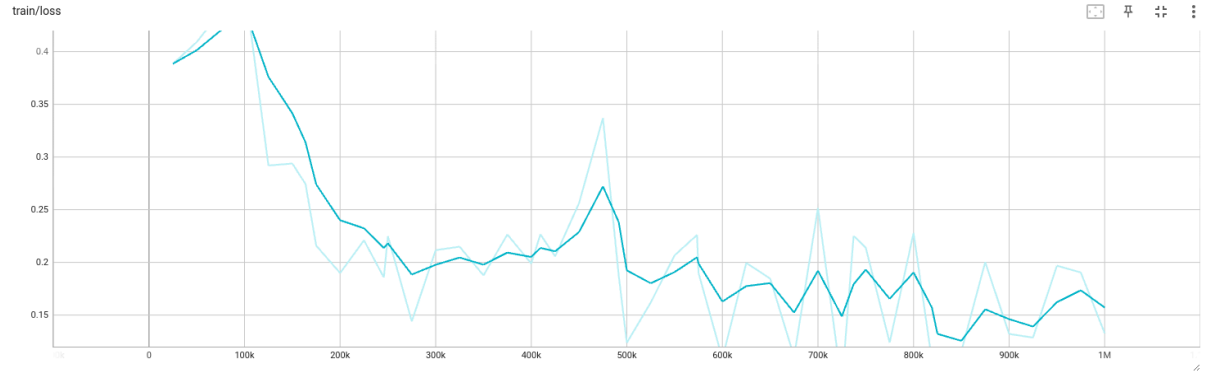


Figure 4: Training loss

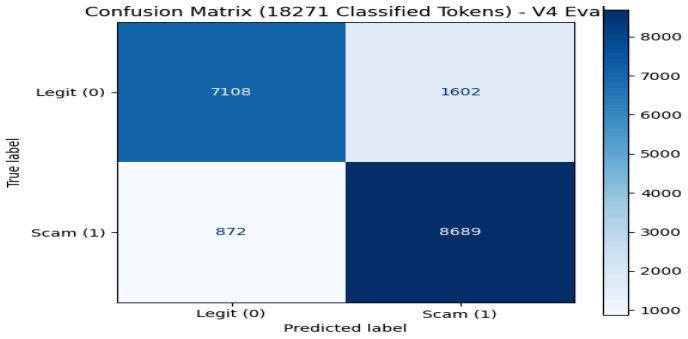


Figure 5: Evaluation metrics

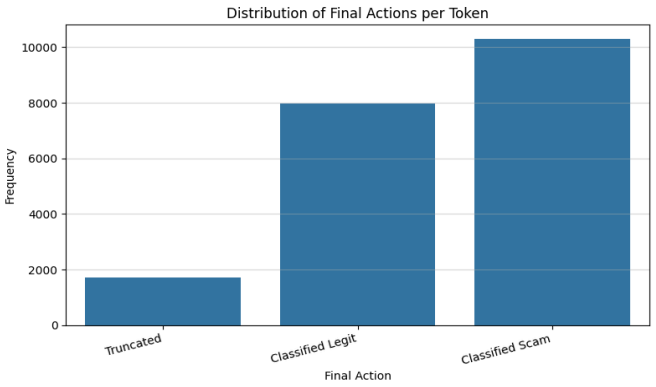


Figure 6: Histogram

The PPO agent successfully learned a functional policy for the timed scam detection task, achieving reasonable performance with potential for refinement.

1. Learning Process & Stability: Training loss decreased substantially before stabilizing with notable variance (around 0.15-0.25). Evaluation reward reflected this, rising significantly before plateauing between 0.6-0.7, also with fluctuations. This indicates convergence to a reasonably effective but potentially suboptimal policy, possibly limited by the problem's complexity or requiring further tuning.

2. Learned Strategy: The agent became highly decisive, rarely truncating (~1700/20k episodes) and usually committing to a classification. A slight, non-excessive bias towards classifying 'Scam' (~10.3k) over 'Legit' (~8k) was observed.

Classification Performance (Conditional): When the agent *did* classify (18,271 instances), its performance was strong, achieving ~86.4% accuracy. It demonstrated high recall for both scams (~90.9%) and legitimate tokens (~81.6%), along with good precision (~89% Legit, ~84% Scam). This indicates a balanced ability to distinguish classes based on observed temporal patterns *once a decision was made*.

4. Conclusion: The agent learned a decisive, time-aware classification strategy that largely avoids truncation and achieves high accuracy (~86%) when classifying. It effectively balances scam detection (~91% recall) with legitimate token identification (~82% recall), demonstrating successful use of temporal features. While the noisy convergence and reward plateau suggest potential for optimization (via tuning or richer data), the current policy represents a balanced and effective approach to the timed detection task.

Contributions

Suhani - Wrote bandit code for phase 1

Madhavendra - Helped in delayed response code for phase 2 and made website

Dipit - Worked on delayed response code for phase 2

Vardan - Worked on phase 1. Wrote PPO code for phase 1 which we didn't end up using.

Bibliography

- Kalacheva, Alisa and Kuznetsov, Pavel and Vodolazov, Igor and Yanovich, Yury, Detecting Rug Pulls in Decentralized Exchanges: The Rise of Meme Coins (October 09, 2024). Available at SSRN: https://ssrn.com/abstract=4981529 or http://dx.doi.org/10.2139/ssrn.4981529
- Mazorra, B., Adan, V., & Daza, V. (2022b). Do not rug on me: Leveraging Machine learning techniques for Automated scam detection. *Mathematics*, *10*(6), 949. https://doi.org/10.3390/math10060949

Links:

Github: https://github.com/suhanijain1/RL_MemeCoin

Video: RL_multi_media.mp4