Extended Abstract

Motivation Over the past few decades, Reinforcement Learning has proved a critical tool in stock market trading (Moody & Saffell, 2001), more recently being applied to large cryptocurrencies as well (Liu et al., 2021). In this paper, we extend its usage to trading recently created small market-cap cryptocurrencies. These coins display significantly higher levels of volatility than traditional assets, making training an RL agent to trade them challenging and often not profitable, and thus leaving the area largely unexplored.

Method We leverage historical transaction data on newly created coins to simulate an online reinforcement-learning trading environment. First, we train LSTM-based feature extractors in a supervised fashion to capture temporal patterns in price. The extracted features are then concatenated with other real-time statistics of the target cryptocurrency (e.g., current price, recent max price, average buy streak, etc.) and fed into a multilayer perceptron (MLP) head, which serves as both the policy and value networks within our PPO framework. We chose Proximal Policy Optimization (PPO) because its clipped objective promotes stable updates in online settings, it is relatively robust to hyperparameter variations, and it has demonstrated strong sample efficiency in financial-trading tasks (Schulman et al., 2017; Deng et al., 2017).

Implementation To ensure our model focused on sufficiently active cryptocurrencies, we first filtered for coins with at least 100 transactions in their first 15 minutes after being launched, using their IDs to gather each coin's full transaction history. We then trained an LSTM network in PyTorch to predict price movements 1, 5, and 10 minutes ahead based on the previous 100 transactions. Next, we built a custom trading environment in Stable Baselines that replays these historical transaction sequences, which is used to train our PPO agent for 10 million time steps. Finally, we split our dataset at the coin level with a 70:30 train-test ratio.

Results Our LSTM models had strong performance, achieving a low MSE in predicting the next minute's movement based on coin price movement data alone. We found that our our additional features did not contribute significantly to short term prediction, their addition in our longer term models was crucial in price forecasting. With these features, our models achieved reasonable accuracy in prediction of the price in 5 and 10 minutes, which are long time horizons relative to the lifetime of most coins (often under 30 minutes).

Next, we utilized these predictions in training our reinforcement learning agent. It initially exhibited unstable training, refusing to trade as doing so would expose it to initial negative reward before it learned how to profitably trade. After we edited the reward function during the initial stages of its learning to encourage buying and selling coins, it stabilized in $5,000,000$ timesteps. It ultimately reached an average training reward per coin of 0.08, equivalent to an 8% return per coin. This generalized well to our test set, where it received a reward of 0.07, or a 7% return.

Discussion Through training several alternative models, we found that a reinforcement learning model struggled to learn future price prediction due to the complexity of this data. Instead, training models offline to do this exhibited the best results. This abstracted one challenging part of trading away from our RL agent, which allowed it to focus on trading strategy and timing.

Moving forward, we plan to enhance the trading environment by incorporating realistic transaction costs such as variable fees and slippage, rather than our current fixed percent fee structure. Additionally, our current agent is constrained to a single buy-sell cycle per coin at a fixed volume, a strategy that we found to be most effected in this high volatility environment. In future work, we will lift this restriction by allowing multiple trade cycles and dynamic volume selection, thereby better capturing real-world trading behaviors. We will eventually move on to live trading too and add live features we are unable to grab with our API.

Conclusion By combining supervised LSTM forecasts with a PPO agent, we trained an RL policy that yields a 7 % return per coin in a simulated small-cap cryptocurrency market. Decoupling short-term price prediction from policy learning proved critical for stability in this high-volatility setting. Future work will focus on refining reward functions to better reflect real-world trading costs, incorporating richer market signals, and live testing.

Reinforcement Learning in Cryptocurrency Trading

Alex Bloom
Department of Computer Science
Stanford University
apbloom@stanford.edu

Michael Liu
Department of Computer Science
Stanford University
mliu1204@stanford.edu

Abstract

We present a combined supervised learning and reinforcement learning model for automated trading of small-cap cryptocurrencies. These assets are characterized by extremely high volatility and short live spans, which explains the lack of academic interest towards them. First, we train LSTM models on historical transaction sequences to predict 1, 5, and 10-minute price movements, then merge these learned representations with real-time statistics (e.g., current and recent peak prices, buy-streak metrics) as inputs to our Proximal Policy Optimization (PPO) agent. In a custom Stable Baselines environment that replays filtered crypto coin histories ($\geq$100 trades in the first 15 minutes), our agent stabilizes by 5 million timesteps and achieves an average simulated return of 8 % per coin in training and 7 % on held-out coins. Decoupling price prediction from policy learning proved essential to stability in this high-volatility setting. Future work will incorporate realistic transaction costs, dynamic trade sizing, and live API integration.

1 Introduction

Automated trading systems have long leveraged reinforcement learning (RL) to optimize decision-making in financial markets, from early applications in equities (Moody & Saffell, 2001) to more recent work on major cryptocurrencies such as Bitcoin and Ethereum (Liu et al., 2021). However, most existing approaches focus on large-cap assets with relatively stable liquidity and price dynamics. In contrast, small-cap cryptocurrencies, specific ones that are launched by individuals and have no value backings, exhibit extreme short-term volatility and fleeting lifespans, posing unique challenges for end-to-end RL: noisy price trajectories hinder value estimation, and sparse trading windows limit sample efficiency.

In this paper, we propose a two-stage framework that decouples price forecasting from policy learning to tame the volatility of small-cap markets. First, we train Long Short-Term Memory (LSTM) networks in a supervised manner to predict 1-, 5-, and 10-minute ahead returns using raw transaction sequences. These learned representations, together with real-time market statistics (e.g., recent high, average buy/sell streaks), form the observation space for a Proximal Policy Optimization (PPO) agent. By filtering for coins with at least 100 trades in their first 15 minutes and replaying their full transaction histories in a custom OpenAI Gym environment, we enable stable online training despite the coins' brief trading horizons.

Our empirical results demonstrate that this decomposition yields substantial benefits: the LSTM forecasters achieve low mean squared error on very short windows, and the combined PPO agent consistently converges to profitable policies, averaging an 8 % return per coin in training and 7 % on held-out test data. This significantly outperforms our preliminary attempts, which involved training a MLP feature extractor that feeds into an LSTM end to end. That end to end model failed to generalize past simple coin trajectories.

Stanford CS224R 2025 Final Report

Despite achieving a positive reward, our model is unlikely to attain the same level of performance in a live trading environment. This discrepancy arises from several simplifying assumptions in our setup, including an oversimplified fee structure, constrained trading cycles, and the absence of dynamic responses from real-world traders and bots. Nevertheless, our work establishes a foundation by outlining modeling strategies and demonstrating initial results for reinforcement learning–based trading in the high-volatility domain of newly minted cryptocurrencies.

2 Related Work

Reinforcement learning has been investigated as a tool for trading market assets for over two decades. Moody and Saffell (2001) were among the first to apply reinforcement learning directly to learn to trade. It showed that reinforcement learning could optimize returns, maximizing Sharpe (a measure of expected returns adjusted for risk) while accounting for fees in trading. Building on this, Deng et al (2016) introduced a novel strategy for training reinforcement learning agents to trade. They used deep learning to extra feature representations of dynamic trading, which were fed into a reinforcement learning agent that then utilized these features in trading.

More recently, reinforcement learning has begun to become applied in cryptocurrency markets. Many studies have focused on Bitcoin and Ethereum. Liu and Tsyvinski (2021) document that cryptocurrency movements depend on a select few factors. Among these include time series movement, legitimizing trading based of off movement data. Fengrui Liu et al (2021) developed a policy to trade Bitcoin at high frequency. It uses LSTM-based price predictors, combining this with PPO for learning a RL agent policy, which is ultimately the architecture we employed. Very recent work has explored alternative architectures. Sarlakifar et al (2025) used a similar framework but switch LSTM networks for xLSTM networks. This has a higher predictive power when trying to capture long-term dependencies, although we found it did not yield improvements when using our relatively short window of historical data.

3 Preliminary Tests

Our initial hypothesis was that an MLP-LSTM model (trained without assist from supervised learning) would succeed in this environment. This model involves a MLP feature extraction head that would process the time series data from the current step, then pass the logits to the LSTM. This was trained end-to-end with no assist from supervised learning.

While still trained using PPO, this model breaks the Markov assumption by maintaining a hidden state that aggregates past observations rather than relying solely on the current observation. This is a highly expressive model for our agent that we believed would allow it fully capture the complicated nature of market movements.

Due to the complexity of training this model, we begun with simplified tasks, which we initially made harder. First, we trained it on a trivial binary output task, where it got a reward for performing action 0 and was punished for action 1 regardless of observation.

Next, we transitioned to a somewhat more complicated time-dependent process. At every moment, the model was shown 1 or 0 and is allowed to take actions 1 or 0. Its goal was to predict whether there was a 1 in the past 5 digits. If it outputted 1 and this was the case, it received a reward of $+0.5$, but if it was erroneous, it received a reward of -1. If it predicted 0 it would receive a -0.2 reward regardless of the true pattern. This converged to a reward of 8.02 within $150,000$ time steps. Running simple code to play the game optimally on repeat, we see the average reward is roughly 8.15, meaning our model achieved near optimal success, with an occasional error.

Our final test was training a model to trade in a highly predictable environment. We generated movement data for coins with 1200 timesteps that exhibited clear patterns. At each timestep, the coin's price had a 0.9 probability of moving in the same direction as it did in the last time step. The magnitude of change in price at each timestep was drawn Uniformly from $(0, 0.01)$. An example coin movement is shown below.

Our model was allowed to buy and sell once, just as it does on our real historical data. Since this problem was more complex than the previous two, it took nearly $1,000,000$ time steps to converge.

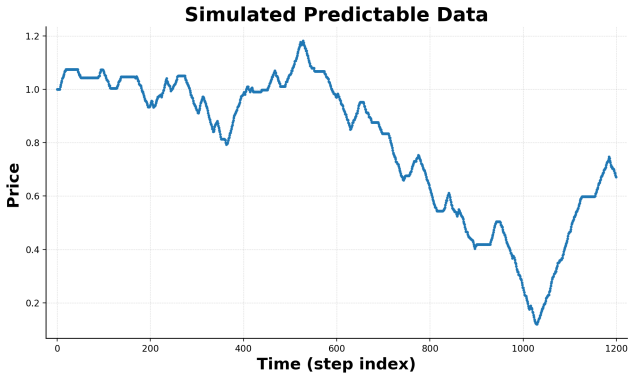

Figure 1: Simulated Predictable Data

Calculating the expected reward of the best strategy is more complex in this scenario, but we observed the action of the agent over dozens of trial coins. It learned to buy immediately after the coin began to increase, and sell immediately after it started decreasing, which is one optimal strategy.

Training this model on our full dataset, however, we found that we were unable to replicate the success on our simplified tests. Observing the steep increase in time steps to convergence as difficulty increased, we hypothesize that with more data and compute, this method of training may be feasible. However, we decided after these experiments to alter our architecture in acknowledgment of our limitations in both compute and data. Rather than training the price prediction and trading policy jointly, we separated them to simplify training.

4 Data

We our training data consists of over 9 million transactions across 8482 coins. We decided to use transaction data instead of OHLCV (Open, High, Low, Close and Volume) data used for most time series models that work with markets. This was because transactions can recreate OHLCV data, making it contain strictly more information that OHLCV data. Because we are working with cryptocurrencies, we have the opportunity to collect all transaction data.

To procure this data set, we went through extensive data cleaning and data collection efforts. After deciding on Solana Tracker API, we started to collect coin mints (unique ID associated with each coin). To optimize our API usage and have noisy training data, we opted to only collect the mints of coins with over 100 transactions with in the first 15 minute of their life time. We had various options for the types of filters we could use, like max market cap, trade volume, holder information, and more. We ultimately decided to only filter on transaction count as it captures almost all high-potential coins without excluding too many. Importantly, because transaction count is observable in real time, applying this filter at deployment introduces no look-ahead bias. Due to the API having more and more unreliable information as the coin ages, something we discovered after testing the search function on coins created at various points in time, we needed to live grab the mints. This is to mimick the distribution of coins the agent would grab at live trading time as close as possible.

After collecting the mints, we allocated a maximum of 100 API calls to each mint to collect transaction data. Each API call can collect 200 transactions. This is important because some popular coins would perform extremely well and have over 100,000 transactions, which would quickly exhaust our API calls and create huge class imbalance.

The transaction data, upon further inspect, were noisy with trivial transactions with volumes less than $\$0.01$. These are clearly not human traders, as the fees would never make such small transactions profitable. Even though we are unsure what these transactions are, it is adding unnecessary noise into an already volatile system, and thus we decided to remove them. However, in case these transactions might hold some information, we decided to keep track of the number of these trivial transactions,

see Table 1 for example.

There were numerous additional data-wrangling challenges. For instance, after filtering out trivial transactions, several coins fell below our 100-transaction threshold, forcing us to decide whether to discard them or adjust our sampling strategy. We also uncovered metadata anomalies: some mints corresponded to tokens created months earlier, which conflicted with our assumption that all coins were newly launched and led to misaligned "grab times." To address many such issues, we devoted a significant portion of the project to cleaning and normalizing the transaction data while continuously debating and refining the desired structure and quality of our final dataset.

5 Trading Environment

We implement a custom Gym environment, `PPOEnv`, that simulates sequential trading of a single small-cap cryptocurrency. At each time step t, the agent observes a 14-dimensional feature vector

$$s_t = \big[s_{t,1}, \ldots, s_{t,14}\big]^\top \in \mathbb{R}^{14},$$

where the components correspond to:

- Time, current on-chain price p_t,
- Average buy/sell streak lengths,
- Lifetime and recent counts of total vs. trivial ("spam") transactions,
- Lifetime and recent maximum prices,
- LSTM-predicted multipliers for +1 min, +5 min, +10 min returns,
- Number of coins held h_t.

The discrete action space is

$$\mathcal{A} = \{\text{BUY}, \text{SELL}, \text{HOLD}\}.$$

If the agent chooses BUY, it spends a fixed amount Q of SOL to acquire $\Delta h = Q/p_t$ coins (unless already holding coins, in which case a penalty fee is charged). If it chooses SELL, it liquidates all h_t coins for $h_t p_t$ SOL (or incurs a penalty if $h_t = 0$). HOLD leaves the portfolio unchanged. We had additional reward signals to guide the model's behavior, outlined below in Figure 2 and its discussion, but is omitted here for notational clarity.

Let c_t denote the agent's cash (in SOL) and h_t its coin holdings at time t. We define a linear transaction-fee function

$$f(v) = \gamma\, v,$$

where $\gamma = 0.01$ in our experiments. The portfolio evolves according to

$$h_{t+1} = \begin{cases} h_t + \dfrac{Q}{p_t}, & a_t = \text{BUY}, \\ 0, & a_t = \text{SELL}, \\ h_t, & a_t = \text{HOLD}, \end{cases} \qquad c_{t+1} = \begin{cases} c_t - Q - f(Q), & a_t = \text{BUY}, \\ c_t + h_t\, p_t - f\big(h_t\, p_t\big), & a_t = \text{SELL}, \\ c_t, & a_t = \text{HOLD}. \end{cases} \tag{1}$$

We define the portfolio value

$$V_t = c_t + h_t\, p_t,$$

and assign the reward as the incremental change in value:

$$r_t = V_{t+1} - V_t. \tag{2}$$

Thus the agent is directly incentivized to increase its portfolio value at each step.

At reset, the environment samples one coin's full transaction history (past and future trades with respect to when the coin's mint was grabbed) and initializes $c_0 = 0,\ h_0 = 0$. On each `step()`, the next transaction is replayed, and the above dynamics and reward are applied. An episode terminates either when the agent executes a successful SELL or when all transactions have been consumed, at which point any remaining coins are forcibly liquidated to realize the final portfolio value.

6 Methods

Instead of expecting our reinforcement learning agent to both implicitly predict price as well as develop a trading strategy, we abstracted away the price prediction to LSTMs, which were trained offline. We trained 3 LSTM models, whose goals were to predict the price 1 minute, 5 minutes, and 10 minutes from a given transaction, using the 100 transactions running up to that point. Our dataset of $9,025,333$ transactions provided a large training set for this, but there was a question as to how we should sample from this dataset. Since in real time, we would not know whether any given transaction was the last transaction for a coin, each one of these transactions was a valid data point to train on.

One way to sample would be simply to pick a transaction at random from the dataset. This initially seems like the clear answer, yet since some coins have tens of thousands of transactions, while others have barely over 100, this would strongly bias our LSTM model to focus its training on larger coins, and it would predict worse on smaller coins at the time of deployment. An alternative strategy would be to pick a coin at random, then select a transaction at random from that coin's set of transactions. However, we ultimately decided to pursue the first option, where we sample a transaction at random, with two main reasons. Firstly, we only had 8482 coins, some of which had very few transactions after we started tracking them. At this size, we would learn to overfit to the transactions in some coins, while never seeing transactions in others. Secondly, the coins we ultimately wanted our agent to learn to trade were the high potential coins, which would generally be very active and have a high number of transactions. For these two reasons, we accepted some level of bias and chose to train on randomly selected transactions.

Once we selected a transaction t, we selected x to be the price and volume of the 100 transactions up to and including transaction t, which had price p. Then, we found target price for one minute prediction was the price p_f of the last transaction before $t + 60$ seconds. If no future transactions were done during that time, we let $p_f = p$, the current price. Our target, y is simply the price change $\frac{p_f}{p} \in [0, \text{inf})$.

Next, we wanted to train a reinforcement learning agent to train based on these predictions of price. The agent would get the chance to buy or sell after each individual transaction. At that time, it would receive a list of features, including the predictions of the LSTM for movement in the next 1, 5, and 10 minutes. Table 1 shows a sample data point (we normalized our data before training but we present a pre-normalized version here to make comprehension easier).

Attribute	Value	Attribute	Value
Time	470	Total Transactions	1240
Price	2.2e-07	Total Spam Trans	102
Avg Buy Streak	9.2	Max Price	8.4e-07
Avg Sell Streak	6.3	Recent Transaction	210
1 Min Prediction	1.1	Recent Spam Trans	12
5 Min Prediction	1.4	Recent Max Price	4.2e-07
10 Min Prediction	1.9	Coins Held	3.2e07

Table 1: Sample data point, price in Solana

We add a large number of additional features. Since our policy no longer has access to the historical movement of the coin, we manually compiled a list of features derived from that which we believed would be useful. The Avg Buy/Sell Streak is computed by first grouping the transaction history into consecutive runs of buy orders and sell orders, and then taking the average length of those runs. Note that the buy and sell streaks do not have the same average length usually; since Pump.Fun uses an automatic market maker, there is only one person on the end of every trade. We also include the total transactions, total spam transactions, and max price of the coin up until the current moment. We categorize a transaction as spam if the quantity bought or sold is trivial ($< \$0.01$). These transactions are bots attempting to make the activity on a coin appear inflated. We also include each of these figures in a "recent" time period, which is the last 5 minutes. Of course, this data point would only

intake transaction data up until the time of this transaction.

The reinforcement learning agent takes this information in after each transaction, and outputs a signal to either buy, sell, or hold (not trade). For each coin, we hope to see it only buy once and sell once, as we have determined during live trading that multiple trades is almost never profitable due to the fees associated with each transaction. We wanted to encourage model convergence by trimming complex behaviors, leaving the model to only learn when to buy and sell once. We implemented this soft "one trade" restraint by enforcing a illegal action fee when it tries to buy again or sell again. //

When we first started training the agent, we found that it would refuse to buy; this is because it will receive a large initial negative reward for buying a coin at random and failing to sell at the right point, given that most of them end up going to zero. To encourage it to explore, we manually implemented a dynamic reward function, whose values changed over the course of training. Figure 2 displays how this changes over time.

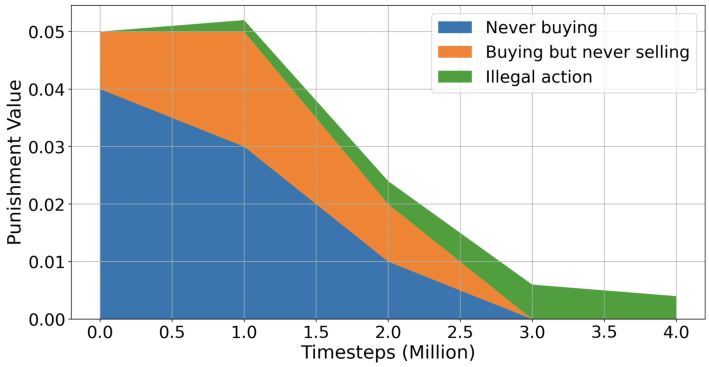

Figure 2: Punishment Over Time For Different Scenarios

We constantly have the fixed reward of profit and loss over trading each coin. However, we initially have a large punishment for the agent never buying, to encourage it to explore more. The punishment for buying but not selling starts out relatively small compared to this (we found that if it became too large, the bot would learn to simply buy than instantly sell). If the agent never sells, we automatically sell it at the last price seen in the transaction data, which is usually very low. We slowly raised this punishment over time, with it peaking at $1,000,000$ time steps at which point we expected the agent to consistently. After it had learned this, we didn't want it to be forced to trade every coin, so both of these punishments eventually dropped away later in our trading. Our final punishment is for an illegal action, such as trying to sell when it didn't have any of the coin, or trying to buy a second time.

7 Results

7.1 LSTM Forecasting Performance

We found that the LSTM models began to converge after relatively few epochs of gradient descent. After this time period, they began to overfit, and validation accuracy would decrease. To combat this, we kept a hidden size of only 30. The results are displayed below.

	1 Minute	5 Minutes	10 Minutes
Only Movement Data	0.062	0.110	0.278
Including Extra Features	0.067	0.098	0.227

Table 2: LSTM Performance (MSE)

It achieved relatively strong performance on each of the tasks. While this appears to be a short period of time it's predicting over, often coins only last 15 minutes and have massive amounts of volatility during this time, so making any predictions is hard.

As the table demonstrates, the LSTM was able to predict accurate returns one minute out with only the movement data from the past 100 transactions. We initially got weaker returns over the longer horizon predictions, so we included the extra features derived from the time series data (a similar set to Table 1.). This helped for the predictions 5 and 10 minutes out, lowering MSE by 11% and 18% respectively.

7.2 Reinforcement Learning Algorithm Trading Performance

The graph below displays our reward by number of time steps. For scale, a reward of 0.10 is equivalent to a 10% return, on average, per coin.

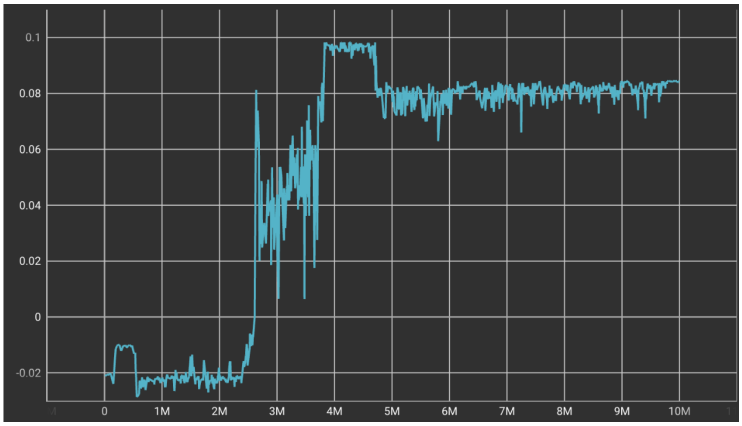

Figure 3: Reward Vs Number of Time Steps

At the end of training (10M times steps), the agent's average per-coin return on the training set was 0.08 (8%). We then evaluated on the hold-out coins, achieving an average return of 0.07 (7%). Notably, the agent trained to a slightly higher average return of nearly 0.1 for a million time steps, before returning to 0.08. Despite our attempts to steer the training by dynamically updating the reward, training was still very unstable, and it likely converged to a local loss minimum, rather than an absolute minimum. In the future, we hope to encourage more stable training by using and ensemble of PPO models.

7.3 Qualitative Analysis

After training the model, we took a look at the coins the agent would actually trade. We found that it would buy very few coins, and the coins that it would buy are nearly all coins with high volume. This both poses a problem as well as reveals a hidden benefit.

Problem: We increase variance by trading on few coins

The agent appears to never buy these smaller coins. While there are a few large coins in our data (which our model does trade), the majority of our nearly 9000 coins do not trade with high transaction volume. Because of this, our model is making very few investments. In trading, the goal is to maximize expected return while minimize variance. Having a guaranteed net 0 return on the majority of coins with large expected value on a few of them exposes you to risk. If none of the few coins we trade are successful, we run at a lost. Instead, if we invested in thousands of coins with small edges in each investment, we would have a high likelihood of profiting. This is why many modern quant firms prioritize ensuring high trade counts, since it reduces risk due to variance.

Upside: We reduce our exposure to scams.

Some of the coins in our dataset appear to go straight up then suddenly fall. However, if you had bought at the moment it started going up (well before the historical data shows it falling) you would

have lost all your money. These coins, known as "honeypots", have their prices artificially inflated by bot traders, which proceed to synchronously sell the coin after someone buys it. However, this is only true for smaller coins. There's a tiny fee for each purchase, and faking trading in a coin of the volume we're trading would be prohibitively expensive. Before seeing our results, we were anticipating having to update the model live to avoid trading into these traps. By only trading high volume coins, we can be confident our historical data would match the real price movement of a coin had we put our money into it.

8 Discussion

Our experiments confirm that decoupling short-term price prediction from policy learning greatly stabilizes training in high-volatility, short-lifespan markets. Training an end-to-end RL agent on raw transaction sequences resulted in persistent instability: the agent could not distinguish noise from genuine price trends while simultaneously learning appropriate trading behaviors. By offloading this prediction task to an LSTM, we provided the RL agent with a more informative and less noisy observation space.

Fee structure. We currently assume a fixed 0.1% fee per trade based on pump.fun's advertised rate. In live trading, however, users must also specify slippage tolerance, transaction speed, and a tip amount (see Figure 4). Slippage tolerance defines the maximum acceptable price deviation between order placement and execution—critical in volatile markets. The speed setting purportedly increases execution priority at an unknown additional cost, and the tip (minimum 0.003 SOL, which is roughly equivalent to 0.44 USD) further influences execution speed. These hidden "options" menu fees render our simple fee model inaccurate and reflect a predatory service design.

Action constraints. Based on limited live experience, profitable strategies in this environment tend to consist of a single buy–sell cycle per coin. Accordingly, we shaped our reward to favor one purchase and one sale, and restricted the agent to a fixed buy quantity. While these constraints simplify training, they should be relaxed in future work to allow multiple trade cycles and dynamic position sizing.

Adversarial behaviors. Our simulator replays historical transactions and does not capture how other market participants react to the agent's actions. This omission precludes scenarios such as "honeypot" scams, where bots inflate a coin's price and then collectively dump it, causing severe losses for real traders. Since past data cannot reproduce these dynamics, live deployment will be necessary to expose the agent to adversarial trading behaviors.

Future work will address these limitations by modeling realistic, variable transaction costs; removing artificial trading restrictions; and integrating the agent into a live environment to learn from genuine market interactions.

9 Conclusion

We have introduced a modular RL framework tailored to the rapid, noisy dynamics of small-cap cryptocurrency markets. By abstracting away short-term price prediction into dedicated LSTM models and feeding their outputs into a PPO agent augmented with handcrafted market features, we achieve stable training and consistent profitability in simulation—8 % average return during training and 7 % on unseen coins. Crucially, decoupling the forecasting task reduces observation noise and accelerates policy convergence in environments where raw transaction data alone proves too erratic for end-to-end learning.

Looking ahead, we plan to enrich our simulator with realistic, variable transaction costs such as slippage, priority fees, and tipping mechanisms. We also hope to relax current trading constraints by allowing multiple buy–sell cycles and dynamic volume selection. Finally, integrating the agent into a live or paper-trading platform will expose it to genuine market feedback and adversarial behaviors, such as honeypot scams, enabling further refinement and validation of its real-world performance.

10 Team Contributions

- **Alex Bloom:** Designed and implemented the data collection and preprocessing pipeline using the Solana Tracker API, gathering and cleaning all coins and transactions. Created

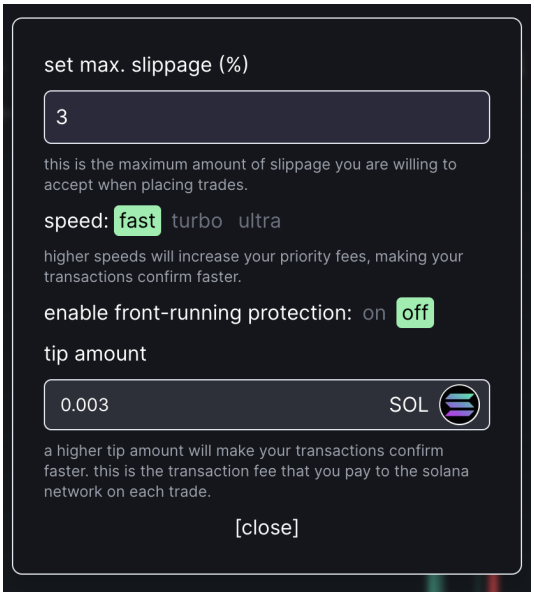

Figure 4: Pump.fun Additional Fee Structure

simulated trading environment for testing MLP-LSTM model. Created LSTM models for 1-, 5-, and 10-minute return predictions. Designed dynamic reward functions. Wrote and reviewed final report.

- **Michael Liu:** Created simple environments to test preliminary MLP-LSTM. Implemented the trading environment in Gymnasium and Stable Baselines wrappers. Integrated LSTM forecasts and handcrafted market features into a PPO-based RL policy, and managed end-to-end training and evaluation for both in-sample and held-out test sets. Wrote and reviewed final report.

Changes from Proposal No changes from proposal.

References

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3), 653–664. https://doi.org/10.1109/TNNLS.2016.2522401

Liu, F., Li, Y., Li, B., Li, J., & Xie, H. (2021). Bitcoin transaction strategy construction based on deep reinforcement learning. *Applied Soft Computing*, 113, 107952. https://doi.org/10.1016/j.asoc.2021.107952

Liu, Y., & Tsyvinski, A. (2021). Risks and Returns of Cryptocurrency. *The Review of Financial Studies*, 34(6), 2689–2727. Retrieved from https://econpapers.repec.org/article/ouprfinst/v$_3$a34$_3$ay$_3$a2021$_3$ai$_3$a6$_3$ap$_3$a2689 − 2727..htm

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4), 875–889. https://doi.org/10.1109/72.935097

Sarlakifar, F., Mohammadzadeh Asl, M. R., Rezvani, K. S., & Salimi-Badr, A. (2025). A Deep Reinforcement Learning Approach to Automated Stock Trading, using xLSTM Networks. *arXiv preprint arXiv:2503.09655*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*