Neural Computing and Applications (2023) 35:21445–21462
https://doi.org/10.1007/s00521-023-08516-x

S.I. : TECHNOLOGIES OF THE 4TH INDUSTRIAL REVOLUTION WITH APPLICATIONS

Combining deep reinforcement learning with technical analysis and trend monitoring on cryptocurrency markets

Vasileios Kochliaridis[1] · Eleftherios Kouloumpris[1] · Ioannis Vlahavas[1]

Received: 30 December 2022 / Accepted: 21 March 2023 / Published online: 20 April 2023

Abstract
Cryptocurrency markets experienced a significant increase in the popularity, which motivated many financial traders to seek high profits in cryptocurrency trading. The predominant tool that traders use to identify profitable opportunities is technical analysis. Some investors and researchers also combined technical analysis with machine learning, in order to forecast upcoming trends in the market. However, even with the use of these methods, developing successful trading strategies is still regarded as an extremely challenging task. Recently, deep reinforcement learning (DRL) algorithms demonstrated satisfying performance in solving complicated problems, including the formulation of profitable trading strategies. While some DRL techniques have been successful in increasing profit and loss (PNL) measures, these techniques are not much risk-aware and present difficulty in maximizing PNL and lowering trading risks simultaneously. This research proposes the combination of DRL approaches with rule-based safety mechanisms to both maximize PNL returns and minimize trading risk. First, a DRL agent is trained to maximize PNL returns, using a novel reward function. Then, during the exploitation phase, a rule-based mechanism is deployed to prevent uncertain actions from being executed. Finally, another novel safety mechanism is proposed, which considers the actions of a more conservatively trained agent, in order to identify high-risk trading periods and avoid trading. Our experiments on 5 popular cryptocurrencies show that the integration of these three methods achieves very promising results.

Keywords Deep reinforcement learning · Machine learning · Proximal policy optimization · Trading · Technical analysis · Risk optimization

1 Introduction

Cryptocurrencies are digital currencies that circulate through a computer network, which is not reliant on any central authority [1]. Over the last few years, both popularity and value of this type of technology has risen, so many traders and investors have shifted their attention to trading cryptocurrency assets, such as Bitcoin.

Cryptocurrency assets are traded in a similar manner to how stocks are traded, but are fundamentally different [2]. First of all, cryptocurrency markets are accessible every hour. Secondly, there are no intermediaries involved in cryptocurrency transactions, so the transaction costs could be lower. Finally, cryptocurrency markets are characterized by their tremendous volatility and rapid fluctuations. For all these reasons, cryptocurrency markets provide traders with great money-earning opportunities, but also involve higher risk [2].

Nowadays, financial markets information spreads easier and more quickly than ever before. As a result, numerous professional investors and traders use technical analysis, which is a tool that is applied on past market data and allows traders to forecast market trends. Technical analysis provides technical indicators, which are pattern-based indications of an asset's momentum, volatility and trend [3]. However, technical indicators are prone to producing

✉ Vasileios Kochliaridis
vkochlia@csd.auth.gr

[1] School of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

false trend signals, so investors and financial analysts usually combine a set of technical indicators [3].

Some researchers and financial investors combined technical analysis with machine learning approaches to forecast upcoming trends [4] in both stock and cryptocurrency markets as further described in Sect. 2. The same works prove that machine learning approaches, especially DRL, has the potential to outperform traditional trading methodologies. Our research, however, highlights a number of crucial aspects, specifically for cryptocurrency trading, that were overlooked by previous works. First, several popular technical indicators were missing from the training data. Second, the majority of these works lack a social indicator, which could possibly confirm several trend signals [5] alongside with the market data. Lastly, previous DRL research relies heavily on black-box agents that attempt to maximize PNL returns, but without safety mechanisms to prevent losses, caused by the agent's uncertain actions [6, 7]. This issue is well addressed on this work, by integrating rule-based safety mechanisms during the trading procedure, and thus reducing trading uncertainty.

In this paper, we extend the methodology of TraderNet-CR [8], which is a trading system composed of three modules. The first module involves the training of a DRL agent, named *TraderNet*, using a novel reward function, named *Round-Trip Strategy* [8]. Furthermore, in this work, we modified the Round-Trip Strategy, in order to combine both market and limit orders, which resulted in significantly higher PNL returns. The second module deploys a rule-based safety mechanism, named *"N-Consecutive,"* which inspects a window of TraderNet's previous actions in order to examine whether a suggested action is uncertain and prevent it from being executed. Finally, in the third module, another safety mechanism is used, named *Smurfing*, which trains a DRL agent, named *Smurf*, more conservatively than TraderNet. Smurf agent is deployed during the exploitation phase in order to detect high-risk trading periods and avoid high-risk trading activity. This approach successfully maximizes PNLs, reduces the trading risk and also improves other portfolio performance measures that are also included in this work.

The remaining of the paper is structured as follows: Section 2 presents the literature review of this research; Section 3 presents the fundamentals of technical analysis, the reinforcement learning (RL) background and TraderNet-CR architecture, which is our previous approach; Section 4 further describes the improvements and additions of TraderNet-CR methodology; Section 5 presents the experiment design, results and discussion; and Sect. 6 includes the conclusion and future work.

2 Related work

Huang et al. [9] investigated cryptocurrency return predictability using technical analysis. More specifically, they constructed a tree-based model for return prediction, which was trained on 124 technical indicators. Their study provided evidence that it is possible to increase the predictive power of a machine learning model by applying technical analysis on market data.

Guarino et al. [4] compared the performance of algorithmic trading agents, which employ technical analysis to build trading strategies, with adaptive and autonomous agents, such as DRL agents, on cryptocurrency markets and other financial assets. The trading agents have been evaluated on well-studied portfolio performance measures in the trading test period. In their work, they discovered that DRL agents used technical indicators more efficiently than automated trading agents. However, in their work, all agents operated in commission free markets. Also, as the authors point out, DRL agents lack explainability, which makes algorithmic trading agents more preferable to investors.

Satarov et al. [6] applied DQN algorithm in order to identify profitable trading points. In this work, a DRL agent was rewarded only during sell actions, with the reward being a subtraction between the current selling price and the most recent buying price. Moreover, penalties were given to the same sequential actions, in order to avoid holding or selling multiple times in a row. The work demonstrated that the reinforcement learning (RL) approach performed better than three traditional technical strategies, considering trading fees of 0.15 percent, which is considerably low.

Mahayana et al. [10] applied PPO to automate BTC trading, using 1-minute candlesticks data and a set of 18 technical indicators. The authors designed a reward function based on the agent's position, including also a penalty which increased for every time step that the agent held a losing position. Five agent variations based on PP0 were evaluated in simulated cryptocurrency trading scenarios. Yet, the work did not consider any transaction costs, and the final evaluation revealed that none of the agents were able to outperform the buy and hold strategy.

In Schnaubelt research [7], PPO was applied on Bitcoin limit orders data, in order to learn optimal limit order placement strategies. Compared to aggressive market order executions, PPO agent reduced total transaction costs by up to 36.93% by using limit orders. PPO agent's strategy was evaluated in comparison with other DRL algorithms, such as DQN and double DQN, as well as a number of other thoroughly researched execution techniques and was found to have superior performance.

Li et al. [11] proposed another deep reinforcement learning architecture for high-frequency trading (HFT) of cryptocurrencies. The key component of their system was the long sequence representation extractor (LSRE), which is a transformer-based architecture designed to extract long-sequence representations. Because HFT requires fast-performing systems, the authors modified the transformer–encoder to include an attention bottleneck based on latent units, which was able to decrease the model's time complexity from quadratic to linear. The learned representations are fed into the cross-asset attention network (CAAN) in order to produce asset scores for portfolio management, and the entire architecture is trained using PPO. Experimenting on four crypto datasets, the LSRE-CAAN trading strategy had better profitability and risk measures compared to previous works, but with some limitations. First, the system's complexity makes it challenging to implement for high-frequency trading. Second, the work only considers historical candlestick data for market order placement and disregards other external factors such as social indicators. Lastly, the system is tested under considerably low fees.

While the previous work considers a single DRL agent to manage the entire portfolio, Lucarelli and Borrotti [12] employed a multi-agent framework by training local DRL agents for each cryptocurrency asset (Bitcoin, Etherium, LiteCoin, Riple). The performance of each local agent produced a local reward signal, which is combined with the rest signals to formulate a global reward signal. The goal of this multi-agent framework was the maximization of the global reward signal, in order to achieve optimal portfolio management. The state space consisted of closing prices across all assets. Even though they achieved very promising results, they completely disregarded the commissions fees.

In another work by Cui et al. [13], the authors trained proximal policy optimization (PPO) agent on market data. In order to construct a low-risk cryptocurrency trading system, they made use of a conditional value at risk (CVaR) reward function, which could effectively capture the compounding effect of tail risk in financial markets. Even though their trading system was able to eliminate devastating market estimation errors during fluctuating periods, their work completely disregarded transaction costs.

To finish with this short related work review, a major problem of the existing literature is that the current state of the art DRL methodologies train agents that are intended mainly for unsupervised use and on low commission fees. Additionally, the previous works prioritized in finding optimal strategies to maximize profits, with very little work done in minimizing trading risk. In our work, we aim to improve upon existing literature by: (a) combining market data with technical indicators and a social indicator, (b) experimenting with a high-performance deep RL algorithm and a novel risk-adjusted reward function and (c) adding layers of safety mechanisms as an extension of our main methodology, which customizes the agent's trading behavior, optimizes trading risks and improves other portfolio performance measures as well.

3 Background

This section focuses on providing important literature regarding the fundamentals of technical analysis as well as the reinforcement learning approaches. Also, this section presents PPO algorithm, which has been used to train the agents, as well as its benefits. Finally, the architecture of TraderNet-CR and the main concepts of our previous work are presented.

3.1 Technical analysis

Technical indicators are used by investors in order to simplify market information and help them formulate trading strategies [3]. In this work, we analyze and provide the definition only of a small subset of technical indicators that are presented on Huang et al. [9] work, which, however, have gained high popularity by traders.[1]

Definition 1 (Exponential moving average) or EMA is a popular type of moving average, which is used to smooth [14] and lessen the amount of noise on a signal. The mathematical formula of EMA is described by Eq. (1).

$$\mathrm{EMA}(n) = \mathrm{Price}(n) * k + \mathrm{EMA}(n-1) * (1-k) \tag{1}$$

where

$$k = \frac{2}{N+1}$$

is the smoothing factor and N is the rolling window size of the indicator.

Definition 2 (Double exponential moving average) or DEMA is a trend indicator, which is used to reduce the lag produced by the exponential moving average (EMA) indicator [14]. The mathematical formula of DEMA is described by Eq. (2).

$$\mathrm{DEMA_N} = 2\mathrm{EMA_N} - \mathrm{EMA}\ \mathit{of}\ \mathrm{EMA_N} \tag{2}$$

[1] The most popular technical indicators used by traders and financial experts have been selected by *AlphaVantage* platform, which can be found on this URL: https://www.alphavantage.co/documentation/.

Definition 3 (Moving average convergence/divergence) or MACD is also a trend-following indicator, which shows the relationship between two EMAs of different periods [14]. The MACD formula is described by Eq. (3).

$$\mathrm{MACD} = \mathrm{EMA}_{12} - \mathrm{EMA}_{26} \tag{3}$$

Definition 4 (Aroon) indicator is used to identify trend changes and estimate their strength [14] . The mathematical formula for Aroon up and Aroon down are described by Eqs. (4) and (5), respectively.

$$\frac{25 - \text{Period since new High}}{25} * 100 \tag{4}$$

$$\frac{25 - \text{Period since new Low}}{25} * 100 \tag{5}$$

Definition 5 (Commodity Channel Index) or CCI is a trend indicator, which is used to calculate price trend, direction and strength [14]. The mathematical formula of CCI is described by Eq. (6).

$$\mathrm{CCI} = \frac{\mathrm{TP}(n) - 20 - \text{Period}\,\mathrm{EMA}_{\mathrm{TP}}}{\text{Mean Deviation}} * 0.015 \tag{6}$$

where

$$\mathrm{TP}(n) = \frac{\mathrm{High}(n) + \mathrm{Low}(n) + \mathrm{Close}(n)}{3}$$

Definition 6 (Average Directional Index) or ADX is a very popular another trend indicator, which is used to determine whether the strength of a trend is strong [14]. The exact mathematical formula of ADX is described by Eq. (7).

$$\mathrm{ADX} = \mathrm{MA} * \frac{\mathrm{PDI} - \mathrm{NDI}}{\mathrm{PDI} + \mathrm{NDI}} * 100 \tag{7}$$

where

$\mathrm{MA} \rightarrow$ Moving Average

$\mathrm{PDI} \rightarrow$ Positive Directional Indicator

$\mathrm{NDI} \rightarrow$ Negative Directional Indicator

Definition 7 (Stochastic oscillator) or STOCH is a momentum indicator, which is most effective in large trading ranges or slow-moving trends [14]. The mathematical formula of STOCH is described by Eq. (8).

$$\frac{\mathrm{Close}(n) - L_{14}}{H_{14} - L_{14}} \tag{8}$$

where

$$L_{14} = \min\{\mathrm{Low}(n), \mathrm{Low}(n-1), \mathrm{Low}(n-2), \ldots, \mathrm{Low}(n-13)\}$$
$$H_{14} = \max\{\mathrm{High}(n), \mathrm{High}(n-1), \mathrm{High}(n-2), \ldots, \mathrm{High}(n-13)\}$$

Definition 8 (Relative Strength Index) or RSI is another very popular momentum indicator, which is used to identify securities that may be primed for a trend reversal or corrective pullback [14]. The mathematical formula of RSI is described by Eq. (9).

$$\mathrm{RSI} = 100 - \frac{100}{1 + \mathrm{RS}} \tag{9}$$

where

$$\mathrm{RS} \rightarrow \frac{\text{Average Gain}}{\text{Average Loss}}$$

is the ratio of average gains (increases) to average losses (drops) of the close price.

Definition 9 (On-Balance Volume) or OBV is a popular technical analysis tool that uses volume flow to predict changes in the prices of an asset [14]. The mathematical formula of OBV line is described by Eq. (10).

$$\mathrm{OBV}(n) = \mathrm{OBV}(n-1) + \left\{ \begin{array}{cc} \mathrm{Volume}(n) & \mathrm{Close}(n) < \mathrm{Close}(n-1) \\ 0 & \mathrm{Close}(n) = \mathrm{Close}(n-1) \\ -\mathrm{Volume}(n) & \mathrm{Close}(n) > \mathrm{Close}(n-1) \end{array} \right\} \tag{10}$$

Definition 10 (Bolliger Bands) or BBANDS is a widely used volatility indicator, which is plotted with 2 lines of the standard deviation of the simple moving average [14]. The mathematical formula of BBANDS for the upper line and the bottom line are described by Eqs. (11) and (12), respectively.

$$\mathrm{BBAND}_{\mathrm{UP}} = \mathrm{Mean}(\mathrm{TP}) + 2 * \mathrm{Std}(\mathrm{TP}) \tag{11}$$

$$\mathrm{bband}_{\mathrm{DOWN}} = \mathrm{Mean}(\mathrm{TP}) - 2 * \mathrm{Std}(\mathrm{TP}) \tag{12}$$

where Mean(TP) is the average typical price and Std(TP) is the standard deviation of typical price.

Definition 11 (Volume-Weighted Average price) or VWAP is a volume technical analysis tool, which as the name suggests, is the average price of an asset weighted by the total trading volume, over a period of time [14]. The mathematical formula of VWAP is described by Eq. (13).

$$\mathrm{VWAP}(n) = \frac{\sum_{n=0}^{N-1} TP(N-i) * \mathrm{Volume}(N-i)}{\sum_{n=0}^{N-1} \mathrm{Volume}(N-i)} \tag{13}$$

Definition 12 (Accumulation/Distribution Line) or ADL is a volume-based indicator, which was designed to measure underlying supply and demand [14]. It accomplishes this by determining whether traders are actually accumulating (buying) or distributing (selling). The mathematical formula of ADL indicator is described by Eq. (14).

$$\mathrm{ADL}(n) = \mathrm{ADL}(n-1) + \frac{(\mathrm{Close} - \mathrm{Low}) - (\mathrm{High} - \mathrm{Close})}{\mathrm{High} - \mathrm{Low}} \tag{14}$$

3.2 Reinforcement learning (RL)

A typical RL problem is formulated as a Markov decision process (MDP), which involves an environment and an agent, who has the role of traversing the environment's states by executing actions and receiving rewards for each action [15]. At each discrete time t, the agent observes the state S_t of the environment, selects an action A_t and then receives a reward or a penalty r_{t+1} from the environment. The goal of the agent is to maximize the discounted cumulative reward, as described by Eq. (15).

$$\mathrm{G}_t = r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \cdots = \sum_{t=0}^{T} \gamma^t * r_{t+1} \tag{15}$$

The discount factor $\gamma \in (0.0, 1.0)$ is used, in order to ensure the convergence of G_t.

An alternative way to formulate the MDP problem is to find a policy π that maximizes the expected discounted cumulative return, as described by Eq. (16).

$$\max_{\pi} E_{\pi}[\mathrm{G}_t] \tag{16}$$

To estimate the expected return of a particular state s, the policy can use a state value function, which is defined as in Eq. (17).

$$V_{\pi}(s) = E[\mathrm{G}_t \mid s_0 = s] = E\left[\sum_{t=0}^{T} \gamma^t r_{t+1} \mid s_0 = s\right] \tag{17}$$

State values are sufficient to define an optimal policy. However, an alternative way is to estimate an action value function, which is used by value-based algorithms and is defined as in Eq. (18).

$$Q^{\pi}(s,a) = E[R \mid s, a, \pi] \tag{18}$$

Finally, some algorithms use the advantage function to optimize a policy, which combines both $V_{\pi}(s)$ and $Q^{\pi}(s,a)$ and is defined as in Eq. (19).

$$A_{\pi}(s,a) = Q(s,a) - V(s) \tag{19}$$

3.2.1 Deep reinforcement learning

In many real-world decision-making problems, the states of an MDP environment are high-dimensional and cannot be easily solved by using traditional RL approaches. DRL is a subfield of machine learning, which incorporates deep learning approaches allowing agents to make both discrete and continuous actions from continuous state spaces as well [15]. This is done by approximating the policy $\pi(a \mid s)$ using a neural network.

There are many DRL techniques to train policies, each having their own benefits [16]. These techniques can be separated between value-based and policy gradient algorithms [15]. Value-based algorithms, such as double DQN [15], try to approximate only a value function, which is then used to find a corresponding policy. On the other hand, policy gradient approaches explicitly build a representation of a policy, which they directly try to optimize [15]. There are also some approaches, such as PPO [17], which attempt to combine the best of both methodologies. policy gradient algorithms are usually less prone to failure and can be also adjusted to continuous action spaces as well.

3.2.2 Policy gradient methods

Policy gradient methods attempt to parameterize the policy function defined as $\pi_{\theta}(s,a) = P(a \mid s, \theta)$, with a parameter θ [15]. The simplest algorithm of policy gradient method is *REINFORCE* [15], which tries to maximize the expected return $J(\theta)$ with respect to policy π, usually with the help of *Gradient Ascent* methods. The parameter θ is updated along the direction of $J(\pi_{\theta})$, through Monte Carlo updates, which is defined by Eq. (20).

$$\nabla J(\pi_{\theta}) = E\left[\sum_{t=0}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) Q_{\pi\theta}(s_t, a_t)\right] \tag{20}$$

This algorithm, however, introduces high variance in log probabilities and cumulative reward values, because during training each trajectory can deviate from each other at great degrees. One way to reduce variance and increase training stability is to subtract the cumulative reward by a baseline function $B(s_t)$ as described in Eq. (21).

$$\nabla J(\pi_\theta) = E\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(Q_{\pi\theta}(s_t, a_t) - B(s_t)\right] \tag{21}$$

Usually, the value function is used as baseline function [16]. By combining Eqs. (21) and (19), with $V(s_t) = B(s_t)$, it is possible to achieve a more stable training formula, which is described by Eq. (22).

$$\nabla J(\pi_\theta) = E\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t)\right] \tag{22}$$

3.2.3 Proximal policy optimization (PPO)

PPO belongs to policy gradient family of algorithms as well and is an improvement of *trusted-region policy optimization (TRPO)* algorithm [17]. However, PPO usually achieves higher performance and is less computationally expensive due to the use of first-order methods, while managing to maintain satisfying policy update sizes [17].

The objective function that PPO tries to maximize is defined by Eq. (23).

$$L(\theta) = \hat{E}_t\left[r_t \hat{A}_t\right] \tag{23}$$

where

$$r_t = \frac{\pi\theta(a_{ta} \mid s_t)}{\pi\theta_{\text{old}}(a_{ta} \mid s_t)}$$

PPO suggests using an importance sampling factor (or clipping factor) ϵ to clip the policy updates and constraint the new policy from going far off from the old policy, as described by Eq. (24).

$$L^{\text{CLIP}}(\theta) = \hat{E}_t\left[\min(r_t\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{24}$$

In other words, the objective function of PPO is upper-bounded, subsequently maximizing gradient step size without letting it become regrettably large. PPO-Clip is usually performed in multiple steps of taking minibatch Stochastic Gradient Descent [17].

To make training even more stable, PPO incorporates the "actor-critical" architecture [17]. This architecture involves two networks: actor and critic. The actor network decides which action should be taken using its policy and the critic network informs the actor about the quality of the taken action and how to update its policy. The learning of the actor is based on the policy gradient approach, while the critic network tries to approximate the value function.

PPO has been proved to be a high-performance algorithm that usually outperforms other policy gradient algorithms, as well as value-based algorithms, in many DRL tasks [17]. PPO training procedure is shown in Fig. 1.

3.3 TraderNet-CR

TraderNet-CR is a risk-aware cryptocurrency trading system composed of two modules: the DRL module and the risk management module. DRL module involves the use of a simulated trading environment to train TraderNet, in order to make appropriate actions and maximize its returns. PPO was selected as the learning algorithm of the agent, because it is stable, robust, fast and easy to implement. The risk management module deploys a safety mechanism, named N-Consecutive, which prevents uncertain actions from being executed.

The goal of the agent is to maximize PNL returns by spotting and exploiting profitable round-trips. A round-trip is a pair of two opposite orders placed one after the other (BUY-SELL or SELL-BUY) that aims to take advantage of price differences and produce profit.

3.3.1 Problem formulation

The trading environment can be formulated as an MDP problem, defined as a tuple (S, A, P_a, R_a), where S, A, R_a define the state space, action space and reward function of the problem, respectively, while P_a denotes the probability transition function. The goal of the agent is to find a good policy π_θ, which chooses the best action a_t in a state s_t. Therefore, the optimization objective of the agent is to find the parameters of θ in order to maximize the PNL returns given by the reward function.

The action space A is defined by 3 actions: {"BUY", "SELL", "HOLD"}. First, the agent selects an action a_t at a state s_t, then receives a reward r_{t+1} by the reward function and finally transits into the next state s_{t+1}.

The state space of the environment is defined as $S \in R^{N\times 21}$. A state s_t is a sequence of ordered vectors $\lambda_t = \{v_t, v(t-1), \ldots, v(t-N)\}$, with N being a user-defined sequence length. In essence, each state includes information regarding the market state at N previous 1-hour time intervals. Each vector has 21 features, including market features and technical indicators, as well as a Google trends score, which is a social indicator that has been shown to be able to increase the forecasting ability of machine learning models [5].

3.3.2 Round-Trip Strategy

After opening a round-trip, TraderNet expects an extreme price fluctuation to occur, in order to close a round-trip and receive a reward, based on the returned PNL, so it is forced

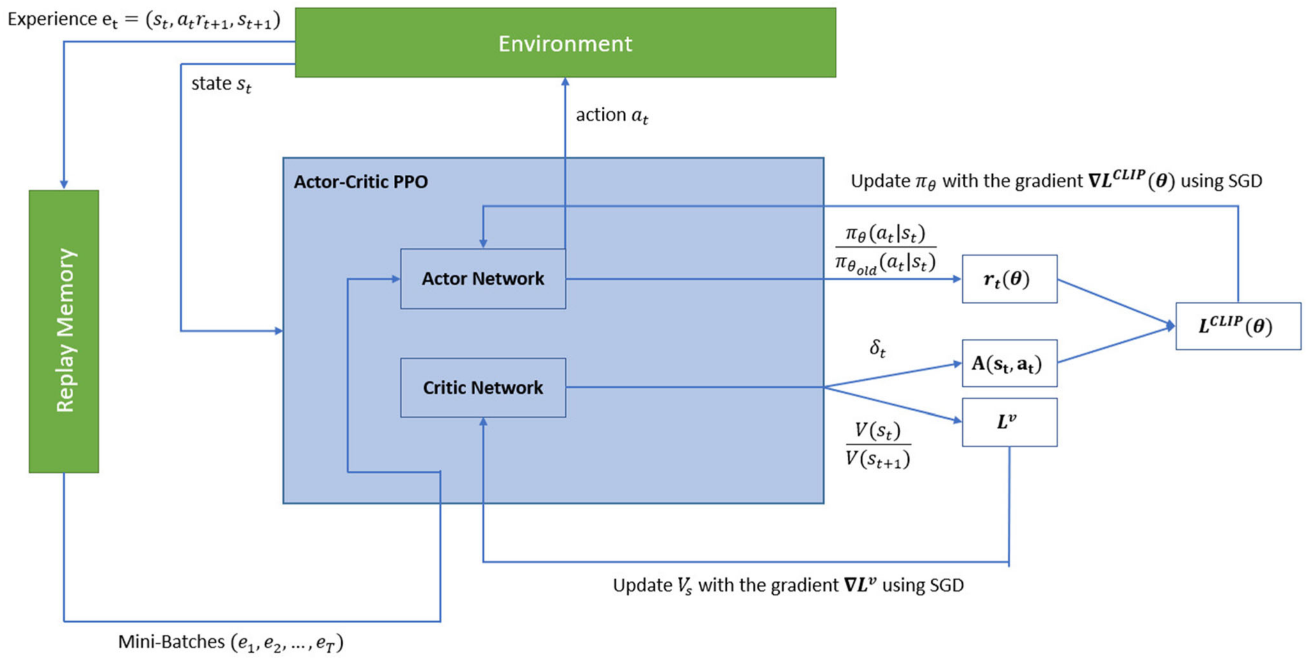

Experience $e_t = (s_t, a_t r_{t+1}, s_{t+1})$
Environment
state s_t
action a_t
Actor-Critic PPO
Update π_θ with the gradient $\nabla L^{CLIP}(\theta)$ using SGD
Replay Memory
Actor Network
$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
$r_t(\theta)$
$L^{CLIP}(\theta)$
Critic Network
δ_t
$A(s_t, a_t)$
L^v
$\frac{V(s_t)}{V(s_{t+1})}$
Update V_s with the gradient ∇L^v using SGD
Mini-Batches $(e_1, e_2, \ldots, e_T)$

Fig. 1 Diagram of proximal policy optimization algorithm using the actor–critic method. The actor uses its policy to select an action a_t, while the critic evaluates that action and informs actor how to update its policy

to wait a few time steps. Because this method could potentially slow down the training process, the agent is immediately rewarded after opening a round-trip based on the maximum possible return within the next K time steps, which correspond to the next K hours. The K parameter is a user-defined horizon value. This eventually trains the internal layers of the agent's networks to estimate future price fluctuations within the near future and use the Round-Trip Strategy to make profit.

Given that f is the fee percentage and C_t the close price of an asset at time step t, the reward function can be mathematically modeled by Eq. (25).

$$r_{t+1} = \left\{ \begin{array}{ll} C_{t_{\max}} - C_t - f(C_{t_{\max}} + C_t) & a_t = \text{BUY} \\ C_t - C_{t_{\min}} - f(C_{t_{\min}} + C_t) & a_t = \text{SELL} \\ -\max(r_{t(a_i)}) & a_t = \text{HOLD} \end{array} \right\} \tag{25}$$

where

$$C_{t_{\max}} = \max\{C_{t+1}, C_{t+2}, \ldots, C_{t+k}\} \tag{26a}$$

$$C_{t_{\min}} = \min\{C_{t+1}, C_{t+2}, \ldots, C_{t+k}\} \tag{26b}$$

The presented reward function ensures that if the agent anticipates a spectacular increase in the price when buying or a huge drop in the price when selling, then it receives a high reward. Additionally, the agent is rewarded more in states where the price volatility is high and less in states that involve low profit potential due to low price volatility, which helps the agent adjust trading risk.

In some previous works, no reward was used ($r_{t+1} = 0$) when choosing HOLD. However, during our investigation, it was found that in some states the agent would prefer to hold its position and avoid trading, due to early losses resulted by exploration. By adopting Round-Trip Strategy as the reward function, the agent is discouraged from frequently holding, because it receives penalties when making unnecessary holds.

3.3.3 N-Consecutive rule

Small price fluctuations in the market, such as noise, could possibly distort the overall market trend. Even with the use of many technical indicators, the agent could be tricked by the market noise and execute unprofitable actions. Such cases can be identified, for example, when the agent switches between BUY and SELL actions in consecutive time steps. To avoid such cases, a rule-based mechanism is deployed, which constraints an action a_t from being executed, unless the N previously suggested actions by the agent are the same ($a_t = a_{t-1} = a_{t-2} = \ldots = a_{t-N+1}$). Because the price of an asset will not change much after the first few time steps, this constraint allows the agent to also examine the suggested action on the next consecutive states and thus reduce the uncertainty.

This mechanism does not interfere with the agent's training and is deployed during the exploitation phase only. Another benefit of N-Consecutive is that it can customize the trading behavior of TraderNet according to the user's preference. Investors that prefer more safe investments may choose large window sizes, while more aggressive investors may choose smaller window sizes.

4 Methodology

In the following section, we present two major improvements of TraderNet-CR, which resulted in higher PNL returns and a more satisfying portfolio optimization performance.

4.1 Modified Round-Trip Strategy

Because TraderNet operates on discrete time steps t, it might miss out small price fluctuations, which have not been considered by the previous Round-Trip Strategy, due to the fact that TraderNet can make market orders only. In this work, we further modify Round-Trip Strategy to allow the agent to use market orders to open round-trips and then fill limit orders to close them, within a fixed number of time steps (referred to as a horizon of K time steps in Sect. 3). Using limit orders allows the broker service to execute the order when the desired price is reached, rather than requiring the agent to make a market order at a specific time step.

Another issue with TraderNet's reward function is that it is heavily dependent by the close price. An alternative approach to modeling TraderNet's reward function could be to use percentage PNL returns, which, however, cannot be summed up, as the base value for each percentage would be different at each time step. So, the objective of the agent, which is to maximize the cumulative discounted reward, as described in Eq. (15), would be inappropriate.

For that reason, we further improve the reward function by transforming PNL percentages to natural logarithmic returns. We can then calculate the total PNL return by summing up these logarithmic returns. If H_t and L_t denote the high price and low price, respectively, between the time $(t-1, t)$, the reward function can now be described by Eq. (27).

$$r_{t+1} = \left\{ \begin{array}{ll} \ln\left(\frac{H_{t_{\max}} - f * H_{t_{\max}}}{C_t + f * C_t}\right) & a_t = \text{BUY} \\ \ln\left(\frac{C_t + f * C_t}{L_{t_{\min}} + f * L_{t_{\min}}}\right) & a_t = \text{SELL} \\ -\max(r_{t(a_i)}) & a_t = \text{HOLD} \end{array} \right\} \quad (27)$$

where

$$H_{t_{\max}} = \max\{H_{t+1}, H_{t+2}, \ldots, H_{t+k}\} \quad (28a)$$

$$L_{t_{\min}} = \min\{L_{t+1}, L_{t+2}, \ldots, L_{t+k}\} \quad (28b)$$

We can also use the well-known logarithmic property described by Eq. (29) to further simplify the reward function, as described by Eq. (30)

$$\ln(A * B) = \ln A + \ln B \quad (29)$$

$$r_{t+1} = \left\{ \begin{array}{ll} \ln\left(\frac{H_{t_{\max}}}{C_t}\right) + l & a_t = \text{BUY} \\ \ln\left(\frac{C_t}{L_{t_{\min}}}\right) + l & a_t = \text{SELL} \\ -\max(r_{t(a_i)}) & a_t = \text{HOLD} \end{array} \right\} \quad (30)$$

where

$$l = \ln\left(\frac{1-f}{1+f}\right)$$

is a constant value. The calculation of modified Round-Trip Strategy is also shown in Fig. 2.

Another benefit of using logarithmic PNL returns instead of actual PNL returns is *"raw-log equality"*: When returns are very small, which is a common thing on round-trips due to short holding duration and high commission fees, the approximation described by Eq. (31) ensures they are close in value to raw returns.

$$\log(1+r) \approx r, r \ll 1 \quad (31)$$

The above mathematical property motivates the agent not to trade on uncertain trading periods, where the potential profit is very small. On the other hand, the agent will prefer trading on periods where there is higher potential of large profits.

4.2 Smurfing

Even though modified Round-Trip Strategy discourages TraderNet to avoid trading on risky states with small potential of profits, we noticed that in some states TraderNet could overestimate price volatility and engage unprofitable round-trips. A potential solution that works well in this problem is to train another agent, named *"Smurf,"* with the goal of detecting high-risk states that it is best to be avoided.

To achieve that, Smurf is trained similarly to TraderNet, but with some modifications. First, Smurf is trained with higher commission fees f', with $f' > f$, leading to less high PNL returns, which makes Smurf more conservative trader. Furthermore, because Smurf is deployed on TraderNet's environment, which includes lower commission fees, it reduces the probability of overestimating PNL returns. Second, we set the reward for the holding action to be a positive small constant $w \ll 1$. The w constant can be adjusted by users, so that higher values of w will result in Smurf tending to avoid trades more frequently, while a

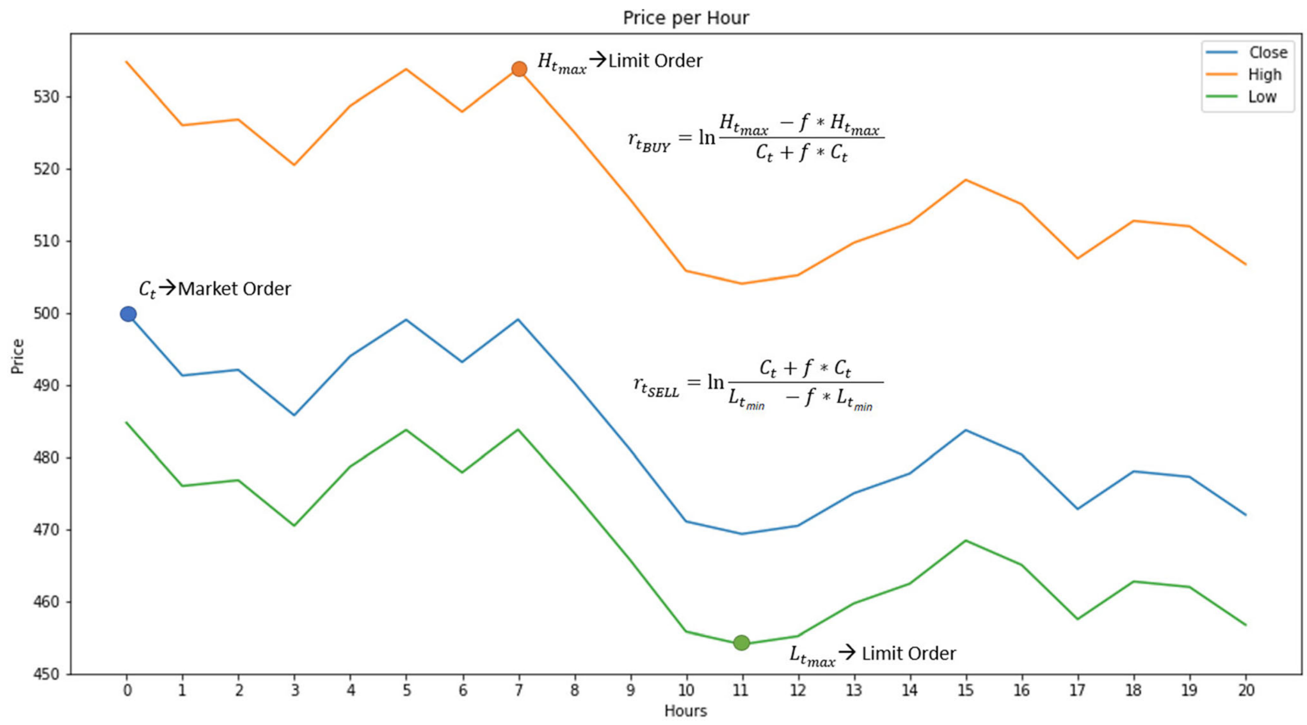


Fig. 2 Visualization of Modified Round-Trip Strategy. The top line represents the high prices, while the middle and the bottom lines represent the close and low prices respectively. The agent opens a round-trip when the price equals C_t, but uses limit orders to automatically close the round-trip

smaller values will ensure that Smurf engages profitable round-trips more frequently. Smurf's reward function can be described by Eq. (32).

$$r_{t+1} = \begin{Bmatrix} \ln\left(\frac{H_{t_{\max}}}{C_t}\right) + l' & a_t = \text{BUY} \\ \ln\left(\frac{C_t}{L_{t_{\min}}}\right) + l' & a_t = \text{SELL} \\ w & a_t = \text{HOLD} \end{Bmatrix} \tag{32}$$

$$l' = \ln\left(\frac{1-f'}{1+f'}\right)$$

4.3 The integrated TraderNet-CR

In this paper, we integrate all the aforementioned modules into a single integrated agent. For each market, we first train TraderNet using PPO algorithm and the modified Round-Trip Strategy. Then, we train a Smurf agent on same markets, but with slightly higher commission fees.

During the exploitation phase, we use Smurf's policy to determine whether opening a round-trip at a state s_t has high PNL potential. Then, we use TraderNet's policy to select an action a_t, which is a BUY or SELL. Finally, we use N-Consecutive mechanism to further reduce the uncertainty of action a_t. If action a_t passes from N-Consecutive mechanism, then we open a round-trip by executing the selected action. The exploitation phase is also shown in Fig. 3.

5 Experiments and results

In this section, we first analyze the datasets and the preprocessing part. Next, we evaluate the performance of each module, as well as the Integrated TraderNet-CR architecture. The evaluation is performed on five cryptocurrency markets. To assess the effectiveness of each module, we use a range of portfolio performance metrics, which are described in the following section. Finally, we examine the performance of the Integrated TraderNet-CR when using a simpler, but widely used DRL algorithm, which is double DQN. The implementation of the extended TraderNet-CR is also available on *Github*[2].

5.1 Datasets

For our experiments, we used historical hourly *Open-High-Low-Close-Volume (OHLCV)* market data of five highly traded cryptocurrency tokens, which were downloaded

[2] https://github.com/kochlisGit/TraderNet-CRv2.

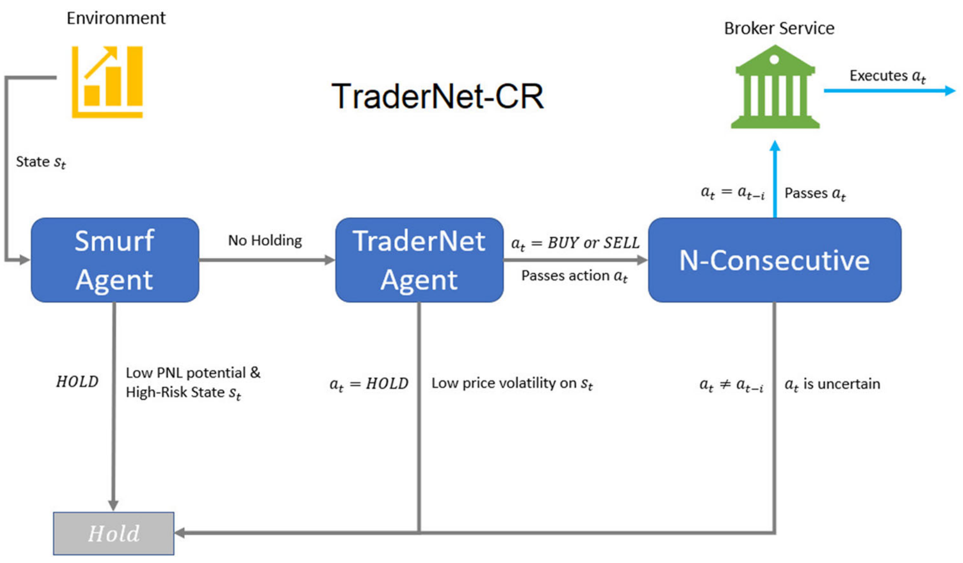


Fig. 3 Integrated TraderNet-CR architecture. First, Smurf receives a state and selects an action. If the action is not HOLD, then the state is passed on TraderNet, which selects an action to open a round-trip. Finally, the N-Consecutive mechanism examines TraderNet's certainty and allows the action to be executed by the broker service

from *CoinAPI* platform. The datasets include past OHLCV data of *Bitcoin (BTC)*, *Ethereum (ETH)*, *Cardano (ADA)*, *Litecoin (LTC)* and *XRP*[3], from 2016, up until November 2022. After the data collection, we applied technical analysis on each dataset, by using widely used technical indicators. These technical indicators, which are also described in Sect. 2, are provided by *AlphaVantage* platform. Additionally, we added hourly Google Trends scores for each cryptocurrency asset, as a social indicator, which can be retrieved by Google Trend's platform[4]. To retrieve Google Trends scores, we constructed a request for each asset separately by using its name.

During the preprocessing phase, we observed significant variances in the ranges of values for various features. To avoid biases in the training process and to speed up the process, we scaled all features to a range between 0 and 1.0, using Min–Max scaling method, which is described by Eq. (34). To ensure that our model generalizes well, we first scaled the training data by computing $\max(x)$ and $\min(x)$ from the training samples, and then, we used the same transformation min and max parameters to scale the test data as well.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{34}$$

Finally, we built the environment states using sequences of size $N = 12$. To calculate the reward function, we set round-trip horizon to $K = 20$. These parameter values have also been used in the original TraderNet-CR paper and were highly effective in this approach as well.

5.2 Experiment setup

To train the agents, a training and an evaluation environment was built for each market. The agents were trained on the training environments, but were evaluated on the evaluation environments every 10,000 training steps. The evaluation environments include the final 2250 hours of each market dataset, which is approximately equivalent to a trading period of three months.

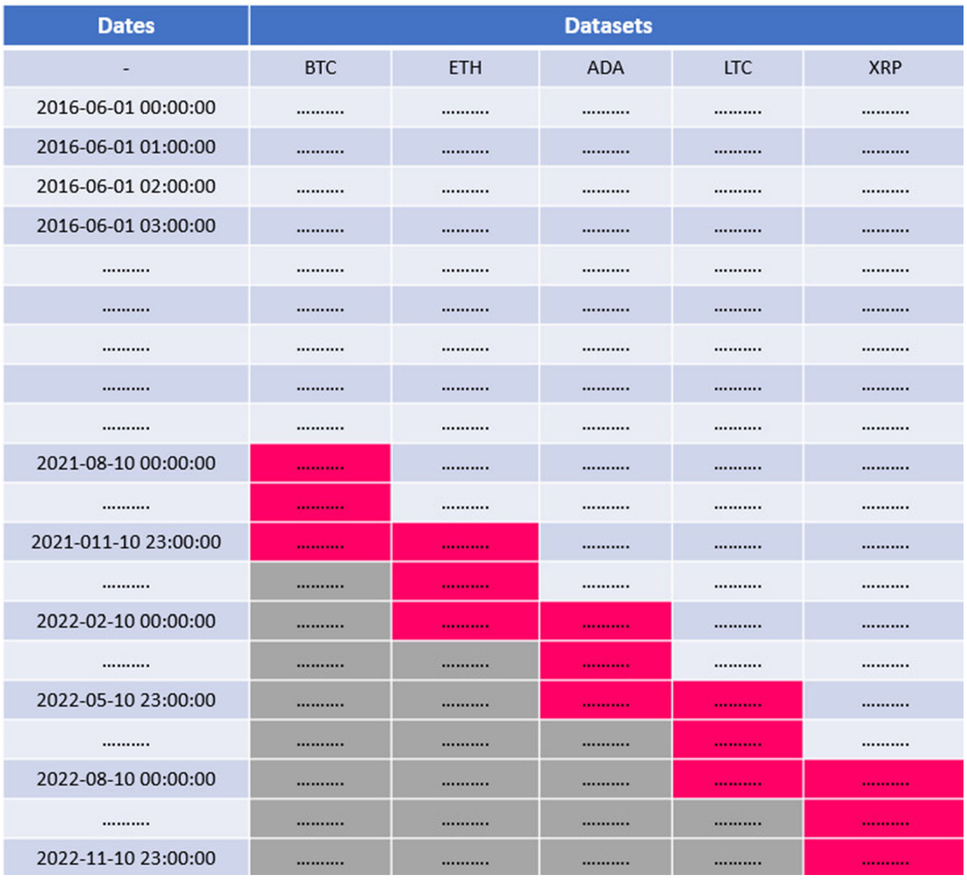
To ensure that TraderNet does not overfit on data, the evaluation samples were only available during the evaluation and were not used on training. Moreover, during our investigation, we noticed that different datasets of the same time period were similar (e.g., BTC price volatility was same as ETH price volatility). For that reason, we used different evaluation timelines in each evaluation environment so that no timeline overlapped with another, as shown in Fig. 4.

We added 1.0% commission fees to each market environment to simulate the effects of trading in a real cryptocurrency market. These fees are representative of what is typically charged by exchanges and brokers in the cryptocurrency market.

To evaluate the performance of TraderNet, we used portfolio performance metrics as in [4] work, which include: *(i) cumulative returns (CR), (ii) cumulative PNL (CP), (iii) investment risk (IR), (iv) Sharpe ratio (SHR), (v) Sortino ratio (SOR) and (vi) maximum drawdown (MDD).* The cumulative returns is defined as the sum of all returns. Because PNL returns are expressed as logarithmic values, we can use the logarithmic property described by Eq. (35) and the exponential constant e to calculate the actual cumulative returns as described by Eq. (36).

[3] CoinAPI URL: https://www.coinapi.io/.

[4] Google Trends URL: https://trends.google.com/trends/?geo=GR.



Dates	Datasets				
-	BTC	ETH	ADA	LTC	XRP
2016-06-01 00:00:00	....	....	....	....	....
2016-06-01 01:00:00	....	....	....	....	....
2016-06-01 02:00:00	....	....	....	....	....
2016-06-01 03:00:00	....	....	....	....	....
....	....	....	....	....	....
....	....	....	....	....	....
....	....	....	....	....	....
....	....	....	....	....	....
....	....	....	....	....	....
2021-08-10 00:00:00	🟥	....	....	....	....
....	🟥	....	....	....	....
2021-011-10 23:00:00	🟥	🟥	....	....	....
....	▨	🟥	....	....	....
2022-02-10 00:00:00	▨	🟥	🟥	....	....
....	▨	▨	🟥	....	....
2022-05-10 23:00:00	▨	▨	🟥	🟥	....
....	▨	▨	▨	🟥	....
2022-08-10 00:00:00	▨	▨	▨	🟥	🟥 ....
....	▨	▨	▨	▨	🟥 ....
2022-11-10 23:00:00	▨	▨	▨	▨	🟥 ....

Fig. 4 Each agent is evaluated on a different time period, which is shown as red color. The blue color indicates the training time period of each agent. Finally, gray color shows samples that are excluded from the training environment. This is because it is unfair to train an agent on future time periods and evaluate it on the past

$$\ln\frac{y_1}{y_0}+\ln\frac{y_2}{y_1}+\cdots+\ln\frac{y_T}{y_{T-1}}=\ln\frac{y_1 * y_2 * \cdots * y_T}{y_0 * y_1 * \cdots * y_{T-1}}=\ln\frac{y_T}{y_1} \tag{35}$$

where y_t expresses the log PNL return at time step t and the ratio $\frac{y_T}{y_1}$ can be interpreted as the percentage change from the initial value to the final value. We can use this property to compute CR as Eq. (36).

$$\mathrm{CR}=e^{\sum_{t=0}^{T} r_{t+1}} \tag{36}$$

where r_{t+1} is the log return given by reward function at time step t.

To compute the rest of the metrics, we define the step Log PNL return on each time step

$$R_t=\left\{\begin{matrix} r_{t+1} & a_t=\text{BUY } or \text{ SELL} \\ 0 & a_t=\text{HOLD} \end{matrix}\right\}$$

The cumulative PNL is similar to CR, but without the HOLD rewards. The definition of CP can be described by Eq. (37).

$$\mathrm{CP}=e^{\sum_{t=0}^{T} R_t} \tag{37}$$

Both CR and CP metrics measure the overall performance of TraderNet and both should be maximized. However, these metrics are not risk-adjusted and do not provide any information about trading risk. Traders usually consider other more risk-adjusted metrics, such as investment risk, SHR, SOR and MDD.

IR is defined as the probability of a round-trip returning negative PNL divided by all returned PNLs. A well-trained agent should have low investment risk and high returns. The mathematical formula of IR metric is described by Eq. (38):.

$$\mathrm{IR}=\frac{\mathrm{BT}}{\mathrm{BT}+\mathrm{GT}} \tag{38}$$

where

$$\begin{gathered} \mathrm{BT}=\sum_{t=0}^{T}\left\{\begin{matrix} 1 & R_t<0 \\ 0 & \mathrm{R}_t>0 \end{matrix}\right\} \\ \mathrm{GT}=\sum_{t=0}^{T}\left\{\begin{matrix} 1 & \mathrm{R}_t>0 \\ 0 & \mathrm{R}_t<0 \end{matrix}\right\} \end{gathered} \tag{39}$$

SHR is defined by Eq. (40).

$$\mathrm{SHR} = \frac{E[\mathrm{R}_t] - \mathrm{r}_f}{\sigma[\mathrm{R}_t]} \tag{40}$$

where r_f indicates the risk-free interest rate. For this experiment setup, we used $\mathrm{r}_f = 0$. Even though SHR is a widely used metric, a better risk-adjusted metric is SOR, which is similar to SHR, but it uses the Standard Deviation only of negative PNL returns instead. The mathematical formula of SOR can be described by Eq. (41).

$$\mathrm{SOR} = \frac{E[R(t)] - r_f}{\sigma\left[\sum_{t=0}^{T}\begin{Bmatrix} R(t) & R(t) < 0A \\ 0 & R(t) \geq 0 \end{Bmatrix}\right]} \tag{41}$$

MDD measures the maximum loss percentage of a portfolio wealth and can be mathematically described as Eq. (42).

$$\mathrm{MDD} = \max \frac{C_t - \min\{C_{t+1}, C_{t+2}, \ldots, C_N\}}{C_t} \tag{42}$$

Finally, to ensure the viability of our approach, we assume that the agent's actions do not affect the market state and that a broker service is available to execute market orders and fill limit orders whenever it is requested.

5.3 Hyper-parameter tuning

PPO components have a number of hyper-parameters. The combinatorial space of those hyper-parameters for both algorithms is too large for an exhaustive search; therefore, we performed limited tuning. We started with the values used in the original paper of PPO that introduced each component, and tuned the most sensitive among hyper-parameters using the grid search method. To determine the best values for each hyper-parameter, we trained our TraderNet model using the modified Round-Trip strategy on Bitcoin environment and selected the values that achieved the highest CR. Due to limited computational resources available to tune our system, the Smurf agent used the same hyper-parameters as TraderNet.

PPO uses a clipping factor $\epsilon = 0.2$, which was selected among $\{0.1, 0.2, 0.3\}$ and discount factor $\gamma = 0.99$, which is a typical discount factor value in RL problems. It trains on mini-batches of 128 samples, and each mini-batch is trained on 40 epochs. For the actor and critic networks, we tried various architectures, including both convolutions and hidden dense layers. For both networks, we used a convolutional layer of 32 filters and $kernelsize = 3$ and 2 hidden layers of 256 units each. Both networks used *Adam* optimizer with learning rate $lr = 0.001$ to update their weights. We selected this configuration, because the agent was trained fast and achieved satisfying cumulative returns among other configurations. The selected hyper-parameters of PPO are also given in Table 1.

Table 1 PPO hyper-parameters

Parameter	Value
Epsilon clipping ϵ	0.2
Mini-batch size	128
Discount factor γ	0.99
Optimizer	*Adam*
Learning rate	0.001
Actor network convolutional layers	$[(32, 3)]$
Actor network hidden dense layers	$[256, 256]$
Critic network convolutional layers	$[(32, 3)]$
Critic network hidden dense layers	$[256, 256]$

In the final evaluation of Integrated-TraderNet-CR, we also used double DQN (DDQN) algorithm. Double DQN uses exploration of $\epsilon = 0.1$, which is a typical exploration value in most DRL environments. We also tested a variation of DQN, on which ϵ starts with $\epsilon = 1.0$ and decays over the time until it drops to zero, but we achieved similar results with the agent training slower. The target network is updated every 3000 steps, which was selected among $\{1000, 2000, 3000, 5000\}$ steps. We also used the *Adam* optimizer to train the Q-network with initial learning of $lr = 0.0005$ and batch size equals 64 as this achieved the best results. The discount factor γ was also set to 0.99. Finally, the architecture of both Q and Target network is the same as actor network of PPO, because it was also fast to train and achieved the most satisfying performance. Table 2 also shows the selected DDQN hyper-parameters.

5.4 Modified Round-Trip Strategy evaluation

In this experiment, we examined how the modified Round-Trip Strategy improves the overall performance of the agent, when compared to the previous Round-Trip Strategy. The experiment results for the modified reward

Table 2 Double DQN hyper-parameters

Parameter	Value
Exploration ϵ	0.1
Batch size	64
Target network update steps γ	3000
Discount factor γ	0.99
Optimizer γ	*Adam*
Learning rate	0.001
Q-network convolutional layers	$[(32, 3)]$
Q-network hidden dense layers	$[256, 256]$

Table 3 Modified Round-Trip Strategy performance

Market	Reward function	CR	CP	IR	SHR	SOR	MDD
BTC	Round-Trip Strategy	1.186	1.186	0.617	1.030	1.097	0.978
	Modified Round-Trip Strategy	**13.173**	**13.173**	**0.477**	**1.357**	**4.475**	**0.357**
ETH	Round-Trip Strategy	17.237	17.249	0.472	1.365	3.035	0.581
	Modified Round-Trip Strategy	**35.808**	**35.808**	**0.331**	**1.776**	**54.248**	**0.112**
ADA	Round-Trip Strategy	14.934	14.934	0.495	1.339	2.609	0.901
	Modified Round-Trip Strategy	**29.304**	**29.933**	**0.338**	**1.736**	**26.970**	**0.791**
LTC	Round-Trip Strategy	16.973	17.000	0.466	1.380	3.329	0.896
	Modified Round-Trip Strategy	**27.939**	**27.939**	**0.353**	**1.706**	**24.905**	**0.156**
XRP	Round-Trip Strategy	21.790	21.790	0.465	1.379	3.991	0.972
	Modified Round-Trip Strategy	**37.607**	**37.967**	**0.341**	**1.668**	**73.033**	**0.328**

The bold entries indicate that Modified Round-Trip Strategy achieved higher performance in all metrics than previous Round-Trip Strategy

Modified Round-Trip Strategy outperformed the previously implemented Round-Trip Strategy in all market environments

function performance are shown in Table 3 and Fig. 5. It can be observed that the modified Round-Trip Strategy outperforms the old reward function in all five markets. More specifically, TraderNet achieves ten times more PNL returns in Bitcoin market and two times more in the other four markets (Ethereum, Cardano, Litecoin, XRP). Overall, the modified Round-Trip Strategy is a more effective trading strategy, as evidenced by its ability to significantly increase PNL returns compared to the previous reward function, which can be attributed to several factors.

One key factor is the use of Log PNL returns, which are calculated based on the natural logarithm of the net profit or loss from a trade. This metric is advantageous because it is not influenced by the close price of a token, allowing the agent to make more informed decisions based on the overall trend of the market rather than being swayed by short-term price fluctuations. Additionally, the modified Round-Trip Strategy utilizes both market orders and limit orders, allowing the agent to take advantage of small price movements that occur between time steps. This allows the agent to more effectively execute trades and optimize its portfolio by maximizing profits while also minimizing risk.

5.5 N-Consecutive evaluation

In this experiment, we tested the performance of TraderNet when combined with N-Consecutive mechanism, for $N = \{2, 3, 4, 5\}$. The experiment results for the N-Consecutive mechanism are shown in Table 4. It can be observed that, usually, larger window sizes result in lower investment risk and maximum drawdown. Additionally, it can be seen that a window size between 3 and 4 performs well in reducing risk metrics while still achieving a satisfactory profit return. This is an important consideration for investors, as a high-risk trading can lead to significant financial losses.

However, larger window sizes also result in decreased PNL returns, due to TraderNet engaging in fewer trades. This is likely because N-Consecutive mechanism forces TraderNet to take a more conservative approach by waiting for more certain market signals before trading. This can be a good strategy in some cases, but it may also result in missed opportunities for profit. For example, using a window size of 5 actions might lead to a significant less profits, such as in XRP and LTC markets environment, where the total PNL dropped almost by 60%.

Overall, it is important to carefully consider the trade-off between risk and return when choosing the window size for the N-Consecutive mechanism. For window size equals to 2 or 3, it usually achieves satisfying balance between investment risk and profit return.

Although the N-Consecutive mechanism can lead to positive cumulative profits for the agent with less trading uncertainty, there are cases where the agent may receive a negative cumulative return by the reward function, as seen in the XRP market example given in Table 4. This happens because agent is penalized for avoiding trades in situations where there is a potential for profit, even during high-risk trading periods. However, since the N-Consecutive rule is only applied during the exploitation period and not during

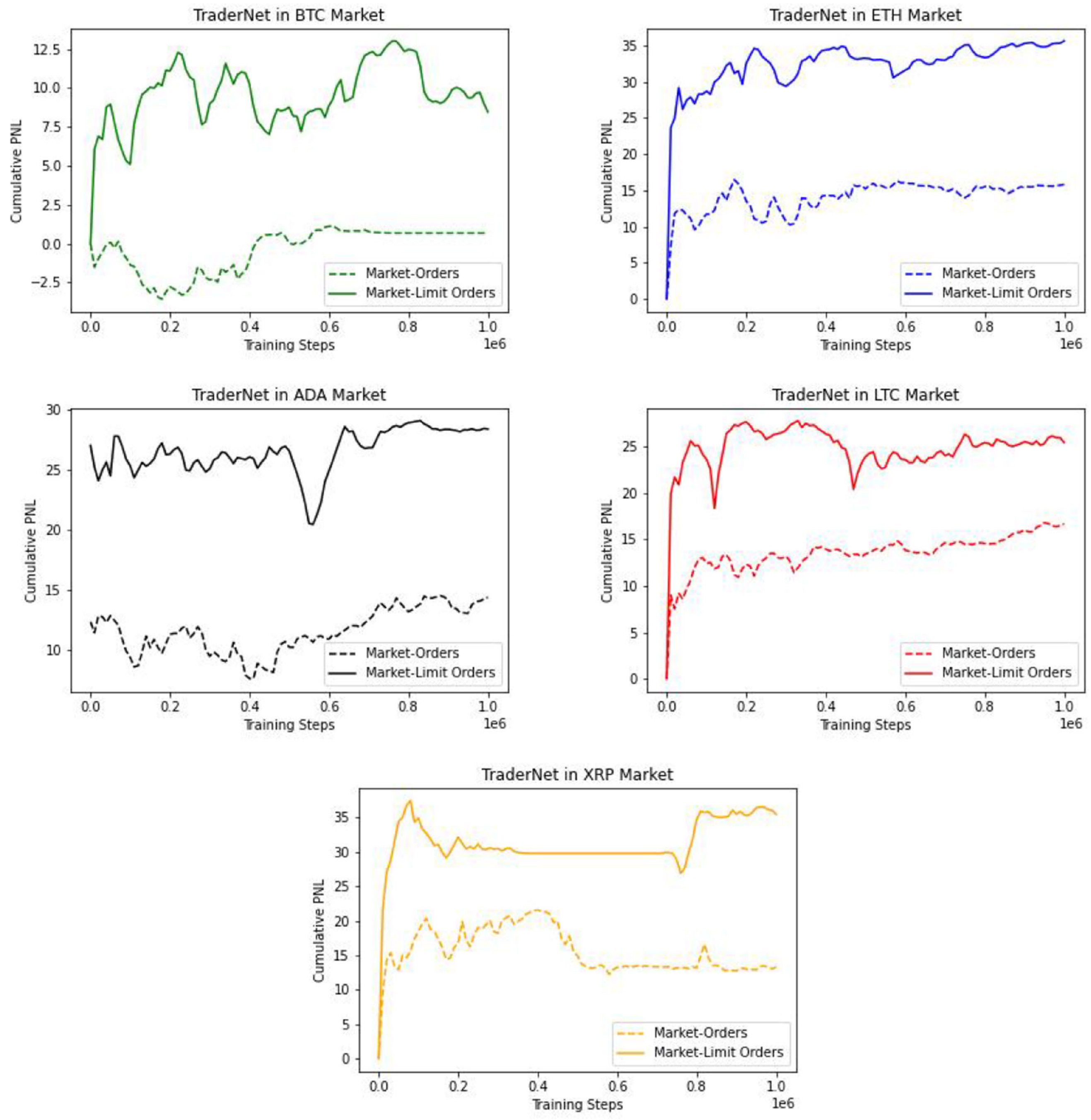


Fig. 5 Performance of TraderNet using modified Round-Trip Strategy vs previous Round-Trip Strategy. In both cases, TraderNet was trained for 1 million steps and was evaluated every 10,000 steps

learning period, it does not affect the trained policy of the agent.

5.6 Smurfing evaluation

In this experiment, we used *Smurfing* method as a safety mechanism for TraderNet. Smurf agent was trained using modified Round-Trip Strategy with *HOLD* reward $r_{t_{HOLD}}$=0.0055 for all markets, commission fees $f' = 1.3\%$ for ETH, ADA, LTC, XRP markets and $f' = 1.2\%$ for BTC market. These values were selected after several experiment trials. The results of this experiment are given in Table 5. While TraderNet alone achieves higher profit percentage, it can be seen that by combining TraderNet

Table 4 N-Consecutive performance

Market	N-Consecutive	CR	CP	IR	SHR	SOR	MDD
BTC	1	**13.173**	**13.173**	0.477	**1.357**	**4.475**	0.357
	2	11.096	12.754	0.475	1.350	4.256	0.817
	3	9.373	12.356	0.475	1.343	2.958	0.531
	4	7.669	11.984	0.474	1.336	2.780	0.572
	5	5.978	11.578	**0.473**	1.329	2.647	**0.242**
ETH	1	**35.808**	**35.808**	0.331	**1.776**	**54.248**	0.112
	2	31.467	34.562	0.331	1.750	46.450	0.126
	3	27.016	33.190	0.326	1.723	39.871	0.124
	4	22.532	31.698	0.327	1.693	33.118	0.110
	5	18.269	30.324	**0.325**	1.666	28.661	**0.102**
ADA	1	**29.304**	**29.934**	0.338	**1.736**	**26.970**	0.791
	2	10.552	24.101	0.331	1.60	13.433	0.853
	3	3.395	21.958	0.329	1.572	10.766	0.350
	4	− 3.219	19.670	0.328	1.528	8.148	0.331
	5	− 8.399	18.014	**0.321**	1.501	6.585	**0.314**
LTC	1	**27.939**	**27.939**	0.353	**1.706**	**24.905**	0.156
	2	15.494	23.926	0.353	1.624	15.518	0.165
	3	8.756	21.073	0.341	1.585	13.630	0.151
	4	2.561	18.767	0.339	1.548	11.178	0.143
	5	− 2.885	16.541	**0.335**	1.517	9.215	**0.140**
XRP	1	**37.607**	**37.967**	0.341	**1.668**	**73.033**	0.328
	2	7.098	28.076	0.321	1.515	24.391	0.470
	3	− 9.646	22.575	0.311	1.439	12.714	0.324
	4	− 21.461	18.467	0.306	1.387	7.803	0.320
	5	− 29.829	15.443	**0.297**	1.351	5.649	**0.301**

Performance of the N-Consecutive mechanism for different window sizes. Window size $N = 1$ means that the rule is ignored. The N-Consecutive mechanism was deployed after TraderNet's training, using the modified Round-Trip Strategy. Based on the bold entries, it can be observed that when using smaller window sizes, the agent achieves higher CR and CP, but will higher transaction risks

Table 5 Smurfing performance

Market	Modules	CR	CP	IR	SHR	SOR	MDD
	TraderNet	**13.173**	**13.173**	0.477	1.357	**4.475**	0.357
BTC	TraderNet + smurfing	− 22.734	4.126	**0.394**	1.197	1.549	**0.289**
	TraderNet	**35.808**	**35.808**	0.331	1.776	**54.248**	0.112
ETH	TraderNet + smurfing	28.370	33.601	**0.294**	**1.783**	45.097	**0.035**
	TraderNet	**29.304**	**29.933**	0.338	**1.736**	**26.970**	0.791
ADA	TraderNet + smurfing	17.658	26.034	**0.319**	1.609	17.915	**0.230**
	TraderNet	**27.939**	**27.939**	0.353	**1.706**	**24.905**	0.156
LTC	TraderNet + smurfing	12.706	24.789	**0.344**	1.588	17.031	**0.148**
	TraderNet	**37.607**	37.967	0.341	1.668	73.033	0.328
XRP	TraderNet + smurfing	37.158	**38.576**	**0.330**	**1.702**	**80.101**	**0.308**

Smurfing approach effectively reduces IR and MDD metrics on all markets. In XRP market, it outperformed TraderNet alone. Also, in ETH market, it also achieved higher Sharpe ratio. Finally, the Smurfing mechanism causes the Bitcoin agent to receive negative cumulative returns. However, the cumulative profit is still reasonable and both IR and MDD have been improved, as it is displayed by bold entries

with Smurf, we are able to achieve a satisfying profit percentage return and at the same time reduce investment risk, as well as maximum drawdown in every market. Additionally, in XRP market, it even outperformed TraderNet in all portfolio performance measures.

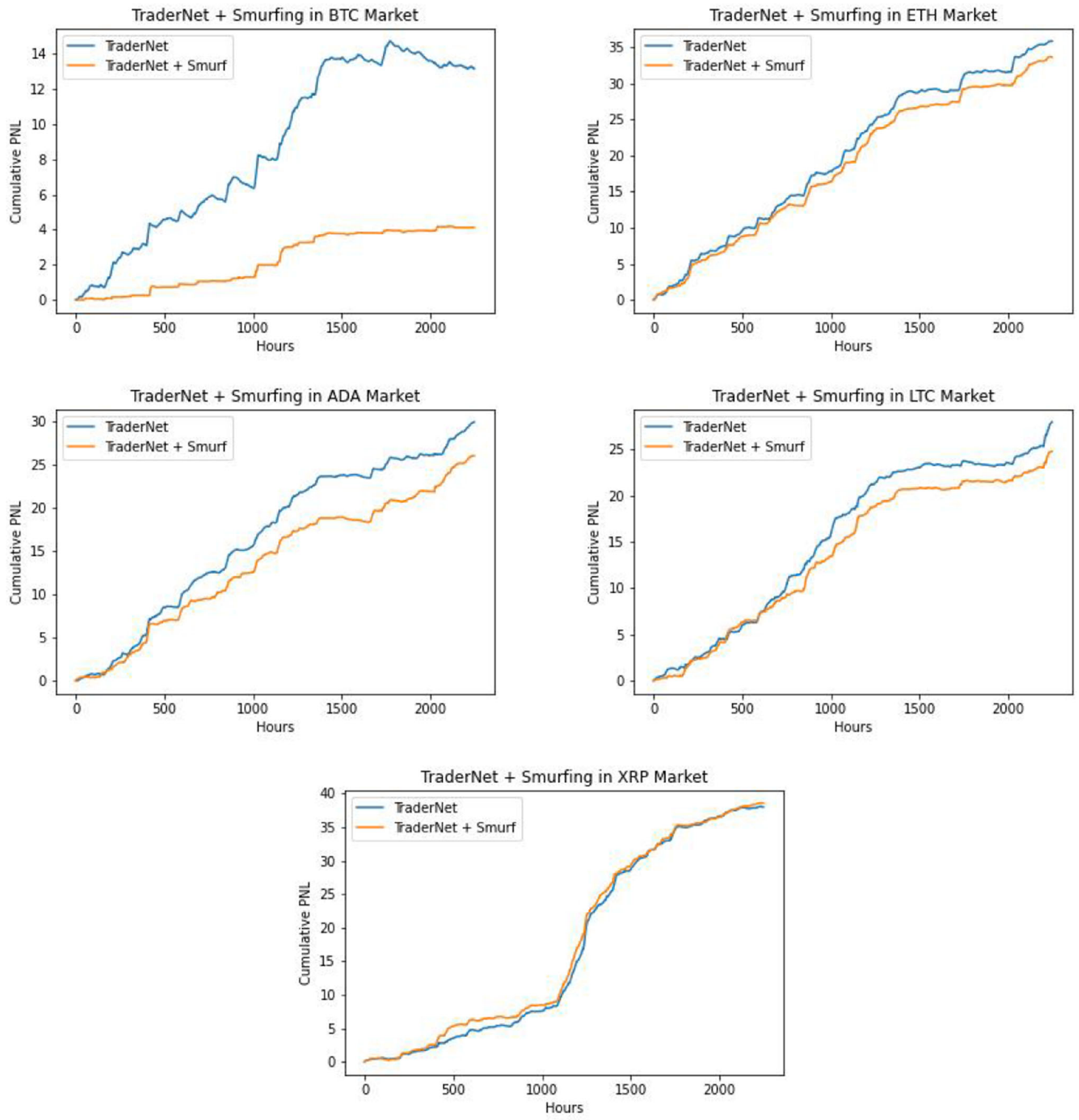


Fig. 6 Performance of standalone TraderNet, compared to the Smurfing approach. The Smurfing approach results in lower PNL returns, but less risky trading

Similarly to using N-Consecutive mechanism, the agent receives lots of penalties for holding, which explains the lower cumulative returns in every market; however, cumulative PNLs are similar in all markets, except BTC. The comparison between TraderNet and TraderNet combined with the Smurfing approach is also shown in Fig. 6. From the figures, it can be observed that when TraderNet is combined with Smurfing, cumulative PNL increases more steadily and has lower drawdowns compared to TraderNet alone, in all experiments.

5.7 Integrated TraderNet-CR evaluation

In Fig. 7, we compare the performance of the Integrated TraderNet-CR approach, as measured by its cumulative PNL, with the corresponding curves for the TraderNet,

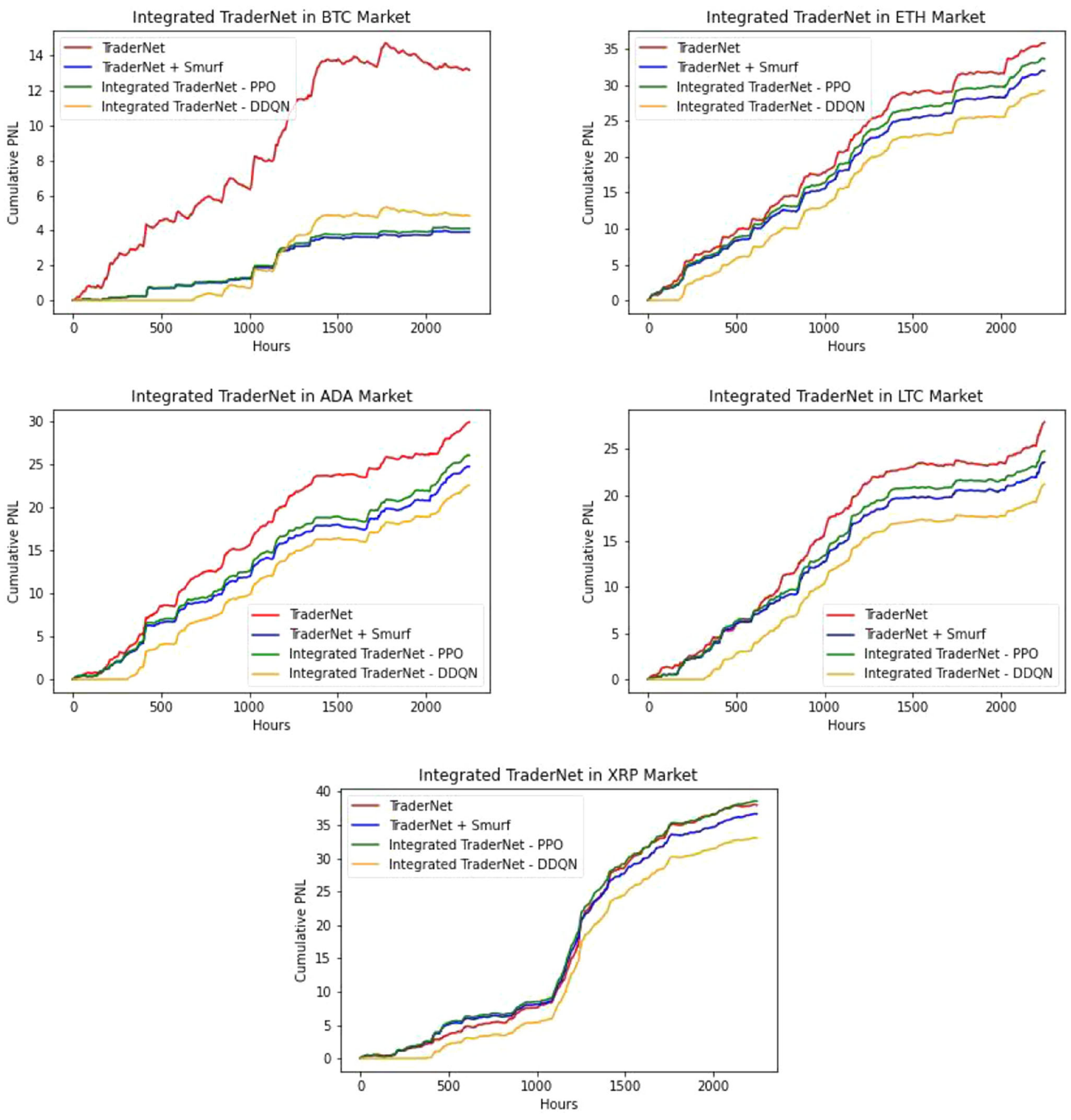


Fig. 7 Performance of Integrated-TraderNet-CR. Integrated TraderNet-CR avoids high-risk trading just like Smurfing approach, but also makes more certain actions, which results in fewer losses

TraderNet-Smurf and lighter version of Integrated TraderNet-CR (trained with DDQN). The figure shows that the performance of the integrated approach is slightly better than its lighter version in four markets (BTC, ADA, LTC and XRP), as well as the TraderNet-Smurf approach, without the use of the N-Consecutive mechanism. Finally, it can be seen that when TraderNet is trained using DDQN, it has the worst performance most of the times when combined with the rest of modules, which makes it an unreliable learning algorithm for this specific problem.

Even though TraderNet alone achieves the highest PNL measures, it is still consider an unreliable trading algorithm, as it can be seen by its PNL curve that it has frequent and bigger drawdowns. On the other hand, the integrated approach results in lower but steadily increasing profits over the trading period.

6 Conclusion and future work

In this paper, we have demonstrated that several modules can be integrated into TraderNet-CR architecture, in order to achieve high PNL performance and reduce trading risk at the same time. More specifically, we have shown that the integrated algorithm is a better consideration for investors, who prefer lower but steadily increasing profits.

Despite the effectiveness of the N-Consecutive mechanism in preventing uncertain actions from being executed and reducing losses, the TraderNet-CR approach can still be impacted by overestimations of price volatility. To address this issue, we have incorporated the Smurfing method into the TraderNet-CR architecture. The Smurfing method helps to identify high-risk trading periods and seeks high-profit opportunities. As a result, the Integrated TraderNet-CR approach, which combines the N-Consecutive mechanism and Smurfing method, has shown improved performance compared to the standalone TraderNet-CR and TraderNet-Smurf approaches.

Finally, in this work, we have focused on training TraderNet using PPO, which has also been used in the previous TraderNet-CR architecture. We have also compared the performance of integrated algorithm when using a well-known value-based method instead, which is DDQN. DDQN has shown sub-optimal performance in all experiments, except in Ethereum market, which has shown a slightly better performance.

Future work regarding the Integrated TraderNet-CR methodology can try to add more technical indicators, as well as try more complicated DRL algorithms, such as Apex-DQN and soft actor–critic (SAC). Additionally, a feature importance on technical indicators might also improve the TraderNet's performance, because state space would become simpler.

Funding Open access funding provided by HEAL-Link Greece. The authors did not receive support from any organization for the submitted work.

Data availability This research uses only publicly available cryptocurrency data from CoinAPI platform, as well as Trends data from Google Trends platform.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. Decent Bus Rev 21260
2. Fang F et al (2022) Cryptocurrency trading: a comprehensive survey. Financ Innov 8(1):1–59. https://doi.org/10.1186/s40854-021-00321-6
3. Lin TC (2012) The new investor. UCLA L Rev 60:678
4. Guarino A, Grilli L, Santoro D, Messina F, Zaccagnino R (2022) To learn or not to learn? Evaluating autonomous, adaptive, automated traders in cryptocurrencies financial bubbles. Neural Comput Appl 34(23):20715–20756. https://doi.org/10.3390/app10041506
5. Arratia A, López-Barrantes AX (2021) Do google trends forecast bitcoins? Stylized facts and statistical evidence. J Bank Financ Technol 5(1):45–57. https://doi.org/10.1007/s42786-021-00027-4
6. Sattarov O et al (2020) Recommending cryptocurrency trading points with deep reinforcement learning approach. Appl Sci 10(4):1506. https://doi.org/10.3390/app10041506
7. Schnaubelt M (2022) Deep reinforcement learning for the optimal placement of cryptocurrency limit orders. Eur J Oper Res 296(3):993–1006. https://doi.org/10.1016/j.ejor.2021.04.050
8. Kochliaridis V, Kouloumpris E, Vlahavas I (2022) Tradernet-cr: cryptocurrency trading with deep reinforcement learning. Springer, Berlin, pp 304–315
9. Huang J-Z, Huang W, Ni J (2019) Predicting bitcoin returns using high-dimensional technical indicators. J Finance Data Sci 5(3):140–155
10. Mahayana D, Shan E, Fadhl'Abbas M (2022) Deep reinforcement learning to automate cryptocurrency trading, pp 36–41. IEEE
11. Li J, Zhang Y, Yang X, Chen L (2023) Online portfolio management via deep reinforcement learning with high-frequency data. Inf Process Manag 60(3):103247
12. Lucarelli G, Borrotti M (2020) A deep Q-learning portfolio management framework for the cryptocurrency market. Neural Comput Appl 32(23):17229–17244. https://doi.org/10.1007/s00521-020-05359-8
13. Cui T, Ding S, Jin H, Zhang Y (2023) Portfolio constructions in cryptocurrency market: a CVaR-based deep reinforcement learning approach. Econ Model 119:106078
14. Pring MJ (1991) Technical analysis explained. McGraw-Hill, New York
15. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT Press, Cambridge
16. Lazaridis A, Fachantidis A, Vlahavas I (2020) Deep reinforcement learning: a state-of-the-art walkthrough. J Artif Intell Res 69:1421–1471. https://doi.org/10.1613/jair.1.12412
17. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347. https://doi.org/10.48550/ARXIV.1707.06347