CITS4404 Team Project - Building AI Trading Bots

Group 6 - Part II: Report

Chen, Zijian
22998691@student.uwa.edu.au

Dai, Ethan
23625929@student.uwa.edu.au

Ida Bagus Prawira Janottama Kn
23894575@student.uwa.edu.au

Nadeesha Adikari
24041382@student.uwa.edu.au

Su, Daniel
22965999@student.uwa.edu.au

Townshend, Nathan
22970882@student.uwa.edu.au

Abstract

This literature review provides an overview for six nature-inspired population-based algorithms. A synopsis is provided for each algorithm with reference to their original papers and a discussion on their merits in relation to obtaining an optimal solution is provided in the conclusion.

1 Introduction

The increasing volatility and complexity of cryptocurrency markets, particularly Bitcoin, present a compelling challenge for algorithmic trading systems [1]. This project aims to address this challenge by developing AI-powered trading bots capable of making intelligent buy, sell, and hold decisions based on technical indicators and historical price patterns. The focus of this phase is to implement and evaluate trading strategies optimized by nature-inspired algorithms, using historical Bitcoin OHLCV data spanning several years.

For Deliverable 2, mainly two algorithms were selected from our prior research: Modified Particle Swarm Optimisation with Simulated Annealing (PSO-SA) and the Artificial Bee Colony (ABC) algorithm. All two methods are rooted in swarm intelligence and have demonstrated robust search capabilities across diverse optimization problems. PSO was implemented as a useful baseline for comparison.

The PSO-SA algorithm combines Particle Swarm Optimization's global search with Simulated Annealing's local refinement, making it well-suited for tuning technical indicators in noisy, dynamic financial markets [2]. The Artificial Bee Colony (ABC) algorithm, inspired by honeybee foraging, balances exploration and exploitation through a decentralized process involving employed, onlooker, and scout bees, offering a simple yet effective approach to optimizing multi-parameter trading strategies [3].

This report outlines the implementation of optimization algorithms to optimize a rule-based

trading strategy using technical indicators such as Simple Moving Average (SMA) , Exponential Moving Average (EMA) and Linear-weighted moving average (LMA). The trading bots are trained on historical data and evaluated on unseen test data, with performance measured based on the final cash value held at the end of the simulation, reflecting the profitability of the strategy.

2 Overview of Investigated Algorithms

In the first part of the project, six algorithms were explored and analysed, focusing on their inspirations, operational mechanics, and effectiveness in tackling complex optimization problems. These algorithms are categorized based on their sources of inspiration—Swarm Intelligence, Physical Phenomena and Laws of Science, and Organism-Based strategies. Each algorithm brings unique features to the field of optimization, as well as various levels of adaptability and convergence efficiency.

Particle Swarm Optimization with Simulated Annealing (PSO-SA) combines Particle Swarm Optimization (PSO) and Simulated Annealing (SA). PSO draws inspiration from the social behavior of birds and fish, where individuals update their positions based on both personal and group knowledge. SA introduces randomness via the Metropolis criterion, enabling convergence toward the global optimum. The hybrid PSO-SA leverages PSO's global search ability with SA's local search strength. To avoid early convergence, PSO-SA uses SA's temperature parameter, initially high to increase randomness and explore diverse areas. As temperature decreases, the search becomes more exploitative, helping the algorithm escape local optima and refine solutions. [2]

The Gravitational Search Algorithm (GSA) is inspired by physical phenomena. It models optimization as agents interacting through Newtonian gravity, where heavier masses attract others more strongly. This encourages diverse movement early, aiding in convergence. However, as gravitational forces weaken, exploration diminishes, and the algorithm can stagnate near local optima.[5]

Grey Wolf Optimizer (GWO) is a swarm intelligence algorithm inspired by grey wolf hunting and leadership. It updates solutions using a weighted influence from the top four wolves—alpha, beta, delta, and omega. This balances exploration and exploitation by considering multiple leaders, reducing premature convergence. [6]

Harris Hawks Optimization (HHO) is based on Harris hawks' cooperative hunting. It dynamically switches between exploration/exploitation based on an escape energy variable tied to solution quality. This helps HHO adapt and escape local optima by increasing randomness when needed.[7]

Artificial Bee Colony (ABC) is a swarm intelligence algorithm based on bee foraging. It divides bees into employed (exploit known solutions), onlooker (choose based on quality), and scout (randomly explore) roles. This division prevents early convergence by continuously exploring while refining solutions. [3]

The Firefly Algorithm (FA) uses brightness-based attraction inspired by fireflies. Solutions move toward brighter ones, with influence decreasing with distance. This limits the global

best's dominance and promotes distributed searching. However, performance relies on tuning attractiveness and can slow in high-dimensional spaces.[8]

These algorithms offer a spectrum of nature-inspired approaches. PSO-SA and ABC were chosen for implementation due to their complementary strengths—PSO-SA for balanced global/local search via hybridization, and ABC for robust diversity handling. These traits suit them for complex financial optimization tasks.

3 Bot Design and Parameterisation

Bitcoin as an asset is fundamentally unlike other classes of assets. It is a currency rather than a stock and is consequently closer to foreign exchange trading rather than stock trading. There are few, if any, fundamental indicators for Bitcoin making it disproportionately impacted by naked supply and demand. This was especially true in its early days as it hadn't yet gained a reputation amongst younger investors as the trendy new replacement for gold as a store of value. Thus, common technical analysis indicators were chosen as inputs for the trading signal.

Three moving averages common in technical analysis are included in the trading signal informing the bot:[9]

- Simple moving average (SMA)
- Exponential moving average (EMA)
- Linear-weighted moving average (LMA)

Different moving averages have sensitivities to different behaviours in the market. The SMA weighs all historical data points equally resulting in an indicator which is more resistant to market volatility. Both EMA and LMA place a heavier weight on more recent data points with the former having an exponential weighting to more recent data points compared to the linear weighting of the latter. EMA is more prone to following the market while LMA provides a middle ground between SMA and EMA.

These three moving averages form two trading signal components:

1. SMA − EMA
2. EMA − LMA

Both differences between averages seek to measure the deviation of the market from the mean, where a crossover is indicative of a potential change in market direction. The first is more sensitive to market fluctuations as SMA carries far more momentum due to its equal weighting of its data points where the second is a more detailed indicator of market direction as their crossovers would provide information into shorter term variations in price direction.

During the training loop, the number of days for each of the above moving averages is optimised. The momentum of each of the averages, calculated as daily deltas, are also factored into the trading signal, allowing it to capture some information about their movement and volatility:

3. SMA momentum
4. EMA momentum

5. LMA momentum

The information provided to the trading bot may not be equally as important to the final trading signal. In recognition of this, five weight parameters were added which control the strength of each of the above five indicators.

6. SMA – EMA weight
7. EMA – LMA weight
8. SMA momentum weight
9. EMA momentum weight
10. LMA momentum weight

Parameter	**Domain**	**Description**
SMA window	$\in \mathbb{Z}^+$, (0, 100]	Window for the calculation of SMA
EMA window	$\in \mathbb{Z}^+$, (0, 100]	Window for the calculation of EMA
LMA window	$\in \mathbb{Z}^+$, (0, 100]	Window for the calculation of LMA
EMA α	$\in \mathbb{R}$, (0, 1)	Used as weight decay rate term for calculating EMA
w_1	$\in \mathbb{R}$, [0, 100]	Weighting for SMA-EMA term
w_2	$\in \mathbb{R}$, [0, 100]	Weighting for EMA-LMA term
w_3	$\in \mathbb{R}$, [0, 100]	Weighting for SMA momentum term
w_4	$\in \mathbb{R}$, [0, 100]	Weighting for EMA momentum term
w_5	$\in \mathbb{R}$, [0, 100]	Weighting for LMA momentum term
θ	$\in \mathbb{R}$, [0, 500]	Threshold for clamping signal to -1, 1

Table 1: Parameters that are optimised for the trading bot.

The final component of the trading signal is the threshold parameter θ. When the signal value exceeds the positive threshold, a buy signal is triggered and vice versa. This simplifies the trading signal to a value between 1 and -1 for buy and sell respectively. The trading signal is as follows where the function s_{raw} produces the unclipped trading signal at time interval t and s is the final clipped trading signal.

$$\begin{aligned} s_{raw}(t) = w_1(\mathrm{SMA}_t - \mathrm{EMA}_t) + w_2(\mathrm{EMA}_t - \mathrm{LMA}_t) + \\ w_3(\mathrm{SMA}_t - \mathrm{SMA}_{t-1}) + w_4(\mathrm{EMA}_t - \mathrm{EMA}_{t-1}) + w_5(\mathrm{LMA}_t - \mathrm{LMA}_{t-1}) \end{aligned} \quad (1)$$

$$s = \begin{cases} 1, & s_{raw} > \theta, \\ -1, & s_{raw} < -\theta. \end{cases} \tag{2}$$

There are ten total parameters which are optimised for use by the bot (Table **??**). These vary the time windows for the calculation of the moving averages and impact the moving average values within function (1).

4 Algorithm Selection and Implementation

Both ABC and PSO-SA can trace their lineages to the original PSO algorithm. PSO was the initial introduction of a nature-inspired optimisation algorithm based on the flocking or swarming behaviour of animals such as birds or fish. PSO is suited to domains with high dimensionality due to its computational efficiency compared to methods based on genetic algorithms. The time available to investigate and tune our models was limited which was a factor in the decision to select PSO-based algorithms.

Early convergence due to a bias towards exploitation rather than exploration is a common issue with PSO. Although the magnitude of velocity updates of the individuals were random, the direction was always towards a combination of global best and personal best depending on the hyperparameters chosen. ABC and PSO-SA both address this by increasing the amount of exploration performed in the search space.

The papers introducing ABC and PSO-SA have benchmarked their performance on a variety of search spaces, including high dimensional and multi-modal functions. In this report, the efficacy of the different exploration methods will be tested in comparison to basic PSO.

4.1 ABC

ABC splits the swarm population into three distinct roles which focus on either exploration or exploitation:

1. Employed bees exploit around known best positions and employs a neighbourhood search to improve their positions.
2. Onlooker bees assess the quality of solutions based on the information gathered by employed bees and select positions based on that information to explore further in the most promising known areas.
3. Scout bees explore the search space by randomly replacing discarded positions with new ones randomly selected from the entire search space.

This role-based approach allows the simultaneous exploitation of the known best positions within the search space and exploration of the entire search space through random sampling.

4.2 PSO-SA

PSO-SA utilises a hybrid algorithm approach by incorporating Simulated Annealing (SA) into the new position acceptance process. When the PSO particle position update process occurs,

particles have their fitness evaluated. The acceptance process uses the temperature during a particular iteration. Higher temperatures allow sub-optimal updates with a higher probability which allows the initial exploration of the search space even if new positions aren't optimal, while lower temperatures encourage exploitation with lower probabilities of accepting worse new positions.

The metropolis process is time consuming, especially at lower temperatures as velocities and positions are repeatedly recalculated for particles which fail the acceptance criteria. Due to time constraints, our implementation contains a hard limit on how many times a particle can be recalculated during the SA loop.

5 Experiments and Evaluation Methodology

This project used daily Bitcoin OHLCV (Open, High, Low, Close, Volume) data from the publicly available Bitcoin Historical Dataset on Kaggle [4]. The dataset spans from 2014 to 2022, offering clean and consistent daily price and volume records. We selected the full available date range to ensure sufficient variation in market conditions for both training and evaluation. Although only the daily data was used in this project, the original dataset [4] also includes hourly and minute-level data.

The daily data was split into a training set (covering the years upto 2019) consisting of 1,861 data points, and a testing set (covering from 2020) with 792 previously unseen data points. This split reflects a realistic workflow in which the trading bot is optimized using historical data and then evaluated on future data to assess its generalizability and robustness.

We employed a grid search approach to tune the strategy's hyperparameters for each optimizer (PSO, PSO-SA, ABC). Over 100 iterations, we explored combinations of values for parameters such as population size, acceleration coefficients, and others. Each candidate configuration was evaluated using a custom scoring function, which was designed to reflect desirable characteristics in a trading bot's behavior. The final selected hyperparameters for each optimizer are listed in the table below. The number 42 was used as the seed in the grid search, as well as for initial training and testing.

Optimizer	**Hyper-parameter**	**Value**
PSO	Number of agents in the swarm (`population_size`)	10
	Personal (cognitive) acceleration coefficient (`p_increment`)	0.2
	Social acceleration coefficient (`g_increment`)	0.1
PSO-SA	Number of agents in the swarm (`population_size`)	10
	Personal acceleration coefficient (`p_increment`)	0.1
	Global acceleration coefficient (`g_increment`)	0.1
	Initial temperature for simulated annealing (`init_temp`)	100
	Cooling rate (`cool_rate`)	0.9
ABC	Colony Size	10
	Position Age Limit	3

Table 2: Optimizers and discovered optimal hyperparameters

Due to time and resource constraints, the grid search was performed only for a selected set of parameter values, and additional parameters would have been explored with more time and resources. The evaluation cost in optimization algorithms like PSO, PSO-SA, and ABC is a key practical challenge. Each candidate solution requires running a trading simulation over historical data, which can be computationally expensive. The number of evaluations grows with the parameter space size and optimization iterations. The evaluation process involves training the strategy on historical data, calculating custom performance metrics, and assessing results, leading to substantial resource usage in terms of time and memory. A major trade-off exists between exploring a broad parameter space, which can yield better results but increases evaluation time, and narrowing the search space, which reduces time but may miss optimal solutions. Given limited resources, efficient allocation is crucial. Techniques like parallel computing can help mitigate evaluation costs, but careful consideration of available resources is still needed to balance performance with practicality.

6 Results and Analysis

6.1 Summary of Training Statistics

We compared three population-based optimisers across training and testing phases, using daily Bitcoin OHLCV data spanning 2014–2019 for training and 2020–2022 for testing. Our primary evaluation metric was final portfolio value (cash after liquidating positions), averaged over five random seeds, with standard deviation to capture robustness.

Algorithm	**Training Score**	**Training Std. Dev.**	**Test Score**
PSO	31353.24	6537.15	2374.42
PSO-SA	28283.67	3934.15	2211.53
ABC	25316.08	4262.97	2070.11

Table 3: Training vs. test final cash across optimisers

Table **??** clearly shows that PSO achieves the highest training returns but suffers the largest drop to test, indicative of overfitting. PSO-SA strikes a middle ground—slightly lower training performance than PSO but more consistent test results. ABC is the most conservative: lowest training and test values, but a modest train-test gap.

6.2 Algorithm Behaviour

6.2.1 Convergence Behaviour

In Figure 1, PSO shows the steepest initial ascent but quickly plateaus at a suboptimal fitness on the test set. ABC climbs more gradually with low inter-seed variance. PSO-SA follows PSO's rapid rise in early iterations, then leverages a Metropolis acceptance (simulated annealing) to escape local traps and achieve the highest final fitness with the tightest clustering.

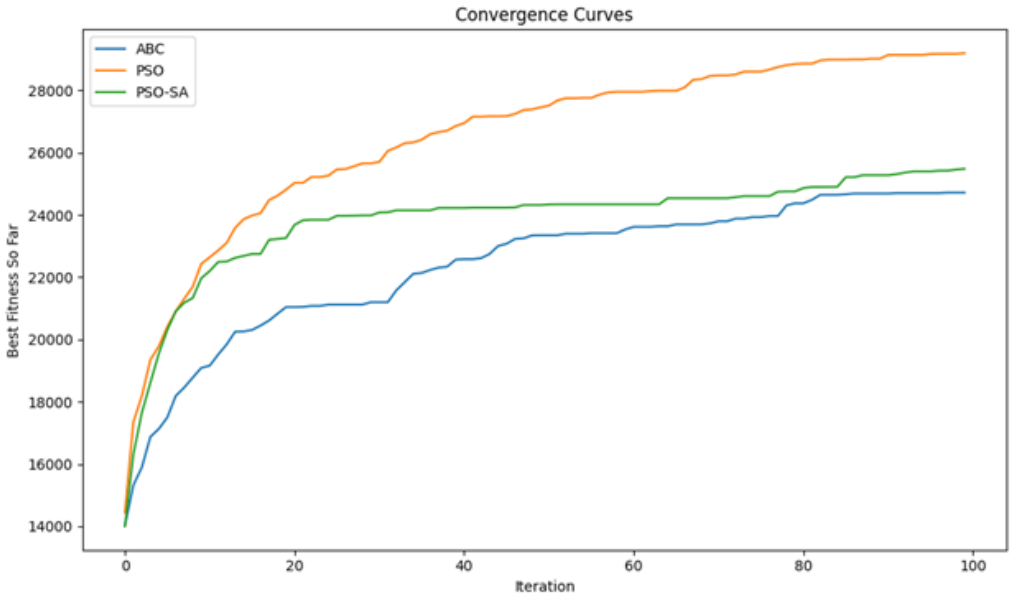

Figure 1: Average Convergence Curves

6.2.2 Robustness

Measured by training-set standard deviation across seeds (PSO $\approx$ 6537, ABC $\approx$ 4263, PSO-SA $\approx$ 3934), PSO-SA consistently exhibits the lowest variability in both training and test returns, making it the most repeatable. ABC sits in the middle—its scout phase injects diversity but also yields moderate scatter. Pure PSO, lacking any "escape" mechanism, is prone to getting stuck; one unlucky seed can drastically underperform.

6.2.3 Exploration vs. Exploitation

1. **PSO**: Inertia-cognitive-social updates provide broad early exploration, but once particles converge on the global best they rarely deviate, leading to premature stagnation.
2. **ABC**: Alternates exploitation (employed/onlooker bees refining known sources) with exploration (scout bees randomly seeding new positions), maintaining diversity at the cost of slower convergence.
3. **PSO-SA**: Adds a Metropolis acceptance rule to PSO's velocity update. High "temperature" early on allows uphill moves (exploration); as temperature cools, it reverts to pure PSO (exploitation). This dynamic schedule balances global search and local refinement.

6.2.4 Equity Curves

Figure 2 shows the mean portfolio value over the unseen test period. PSO-SA delivers the smoothest equity growth—smaller drawdowns and steady gains. PSO's curve is far more jagged, echoing its over-sensitivity to initial seeds. ABC yields the flattest, most conservative growth.

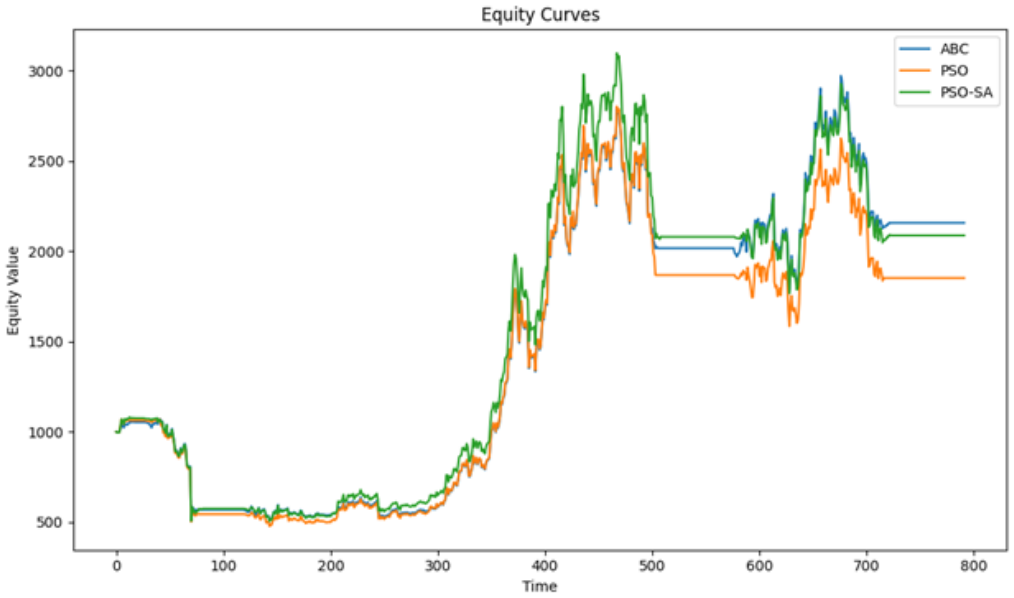

Figure 2: Average Equity Curves

6.2.5 Indicator Evolution

We visualise how the learned thresholds and buy/sell signals evolve on the same test series for each algorithm, show how the buy/sell indicator toggles between +1, –1, and 0 over time. PSO-SA's signal exhibits balanced responsiveness: it captures major up-trends and down-trends without excessive whipsaws. PSO generates more frequent, erratic flips, explaining its higher transaction count and associated slippage. ABC produces sparser signals, often missing shorter trend reversals.

The trading signal figures for this section are provided in the Appendix.

All three algorithms capture the major Bitcoin uptrends, but PSO often chases minor fluctuations (many small trades), consistent with its overfitting. ABC trades less frequently, while PSO-SA provides a middle ground.

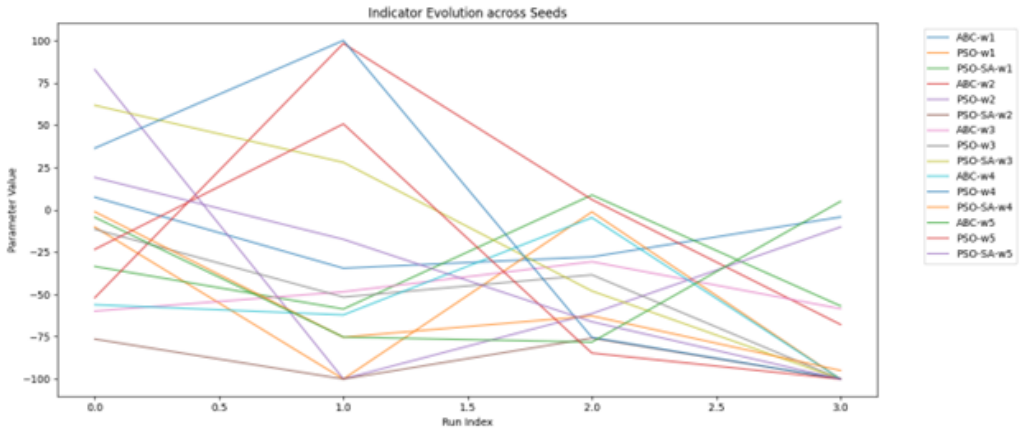

Figure 3: Parameter evolution

Figure 3 tracks five key momentum weights (w1–w5) across seeds for each algorithm's best solution per run. ABC's weights fluctuate most dramatically (due to its scout resets). PSO-SA's curves are the smoothest, signalling highly consistent parameter choices. PSO shows sporadic jumps, reflecting its greedy no-escape policy.

6.2.6 Parameter Sensitivity

We inspected correlations among the ten learned hyperparameters (averaged over seeds). Figure

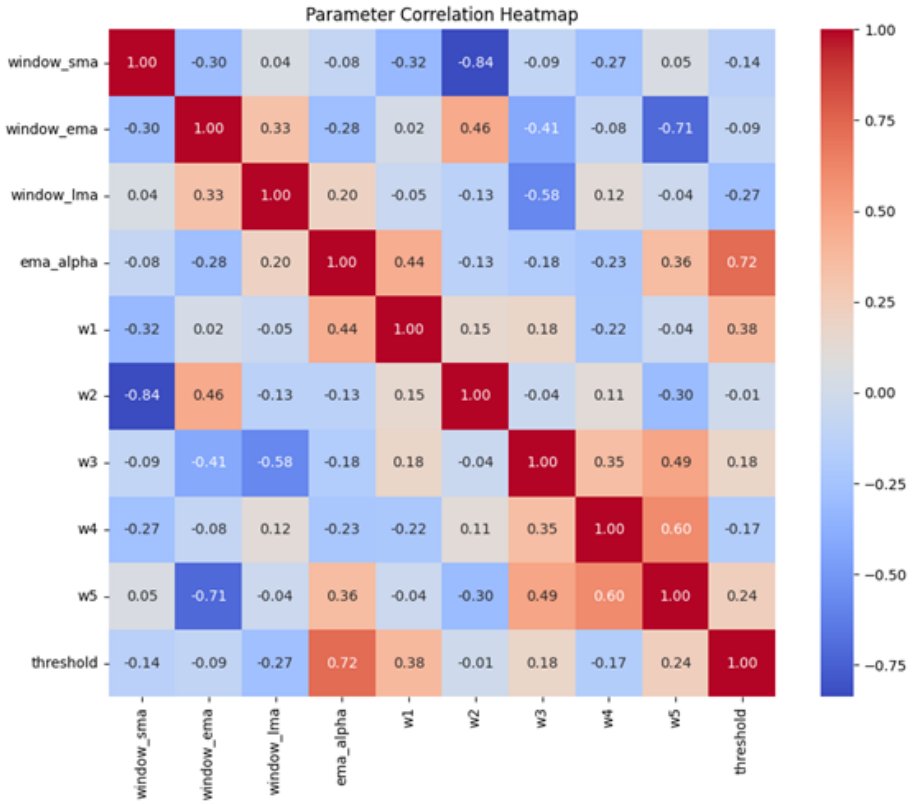

Figure 4: Parameter correlation heatmap

4 reveals pairwise correlations among the ten optimised parameters. A striking -0.84 correlation between `window_sma` and w_2 (EMA−LMA difference weight) suggests that longer SMA windows are systematically offset by more negative momentum weights. Such insights can drive more efficient future grid searches by pruning redundant dimensions.

7 Conclusion

In this project, we successfully implemented and evaluated three nature-inspired algorithms: Particle Swarm Optimisation (PSO), PSO with Simulated Annealing (PSO-SA), and Artificial Bee Colony (ABC) for optimising a rule-based Bitcoin trading bot. The core objective was to leverage these algorithms to find effective parameters for a strategy based on Simple, Exponential, and Linear-Weighted Moving Averages (SMA, EMA, LMA). Key outcomes include the functional implementation of the optimisation pipeline and a preliminary comparison of the

algorithms based on back-tested performance on historical data (2017-2019 training, 2020-2021 testing). Quantitatively, while PSO achieved the highest average training score (31353.24), its test score (2374.42) was comparable to PSO-SA (2211.53) and ABC (2070.11), suggesting challenges in generalisation, especially to new market conditions.

Several lessons were learned throughout this process. Regarding algorithm behaviour, the project confirmed the known tendency of standard PSO towards premature convergence. The selection of PSO-SA and ABC was motivated by their mechanisms to enhance exploration, PSO-SA through simulated annealing's probabilistic acceptance and ABC via its role differentiation, aiming to better navigate the complex, potentially multimodal search space of trading parameters. This practical application highlighted the balance between exploration and exploitation inherent in optimisation.

We found that, while building the bot, preparing trading strategies for optimisation is a quite challenging. Defining the search space by parameterising indicator window sizes and weights resulted in a high-dimensional problem. The performance was sensitive not only to the algorithm's effectiveness but also to the choice of indicators and the structure of the trading rules. Furthermore, the extreme volatility and signal noise characteristic of the cryptocurrency market made it difficult for the optimisation algorithms to find consistently profitable parameters. Financial data often changes its patterns over time, making it difficult for strategies optimised on past data to be always effective, illustrating the limitations of back-testing.

For future work, there are a few paths to explore. We could test the optimised bots using data from recent times or in a live trading situation to better understand their reliability. We could get better results by using more advanced indicators or features, or by exploring other algorithms from Part 1 (for example, GWO or HHO). Furthermore, refining the evaluation to include consistency metrics, such as the standard deviation of scores, in addition to the final profit, may lead to more robust and reliable trading strategies.

A Trading Signals for PSO

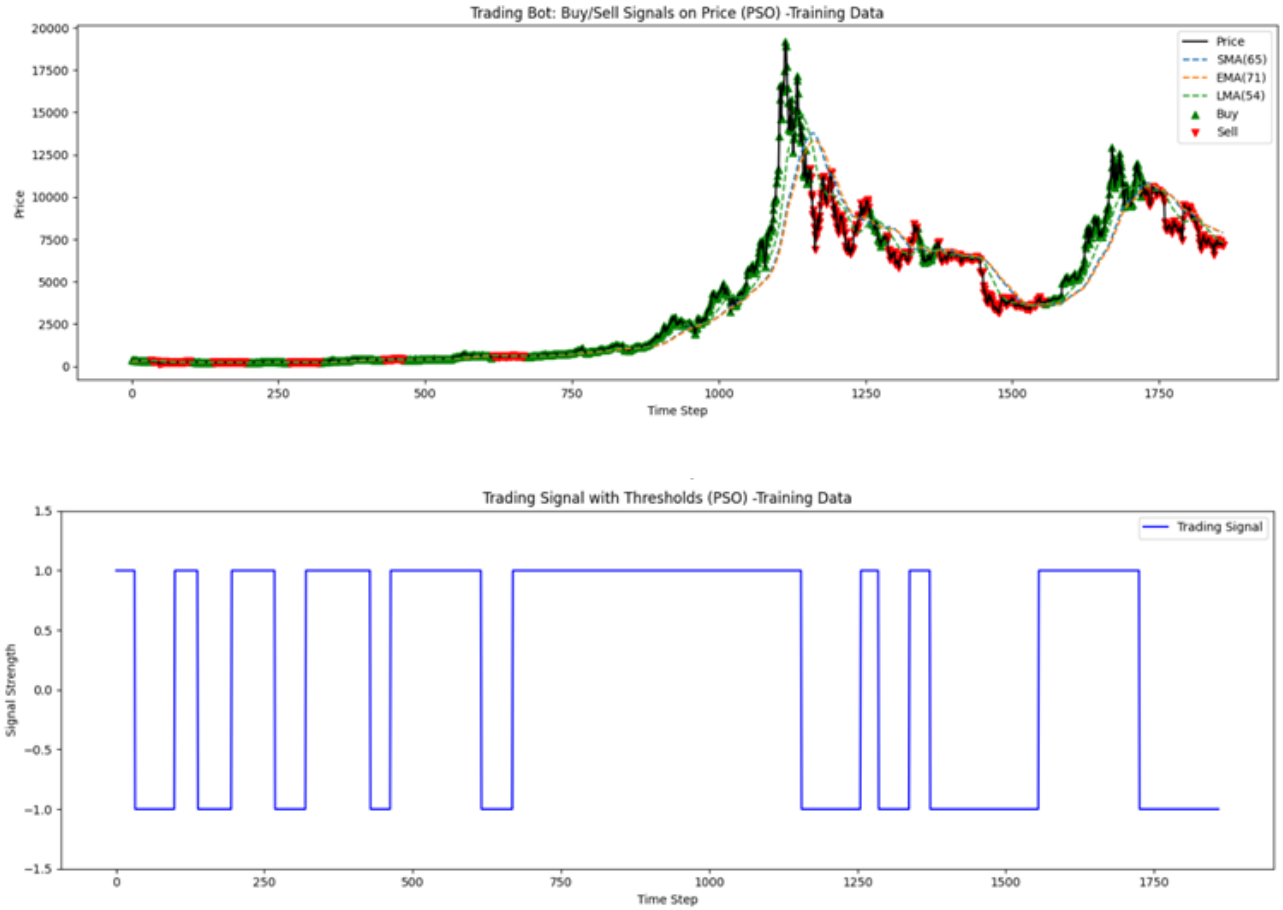

Figure 5: The trading signal and matching points on the price graph for the **training** set.

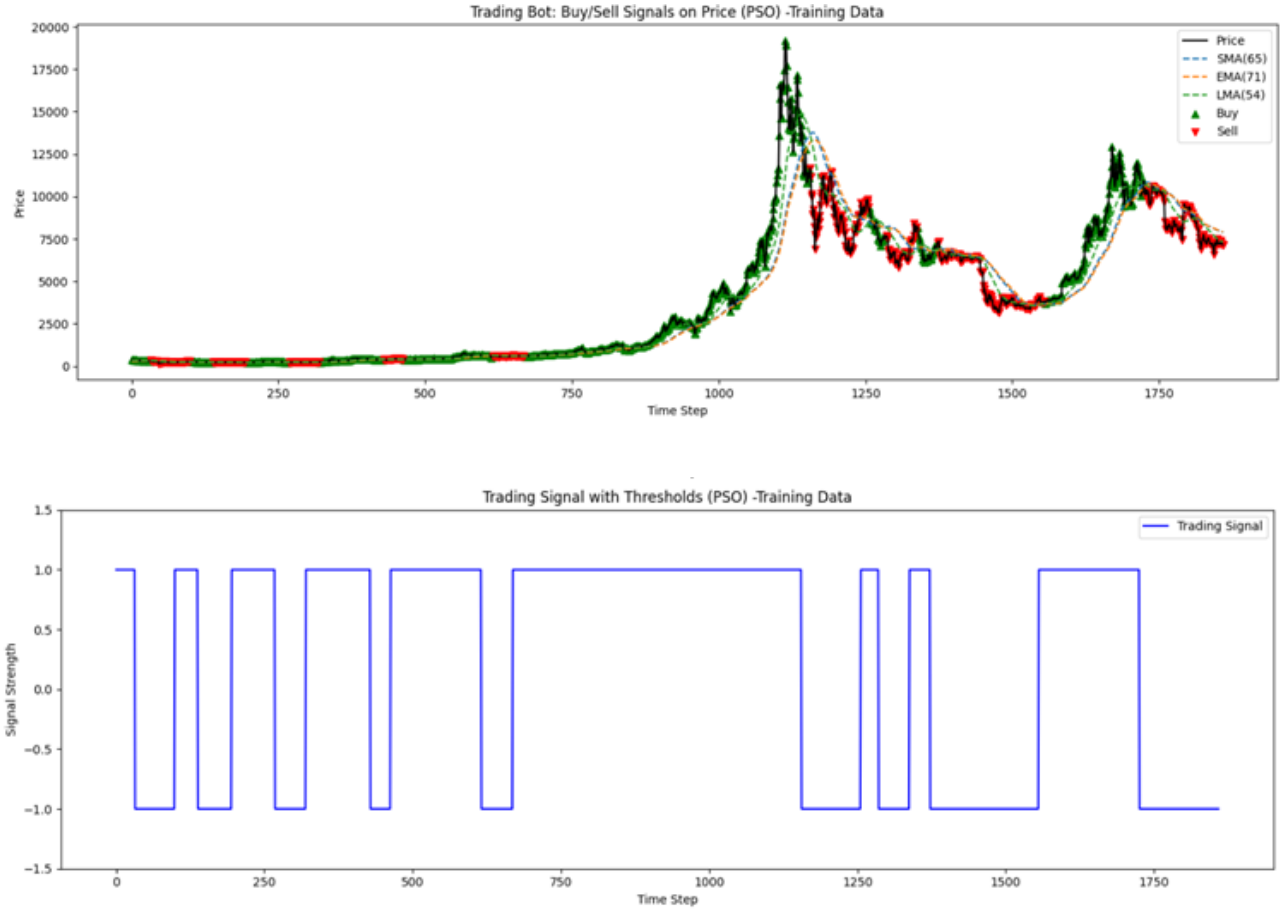

Figure 6: The trading signal and matching points on the price graph for the **test** set.

B Trading Signals for PSO-SA

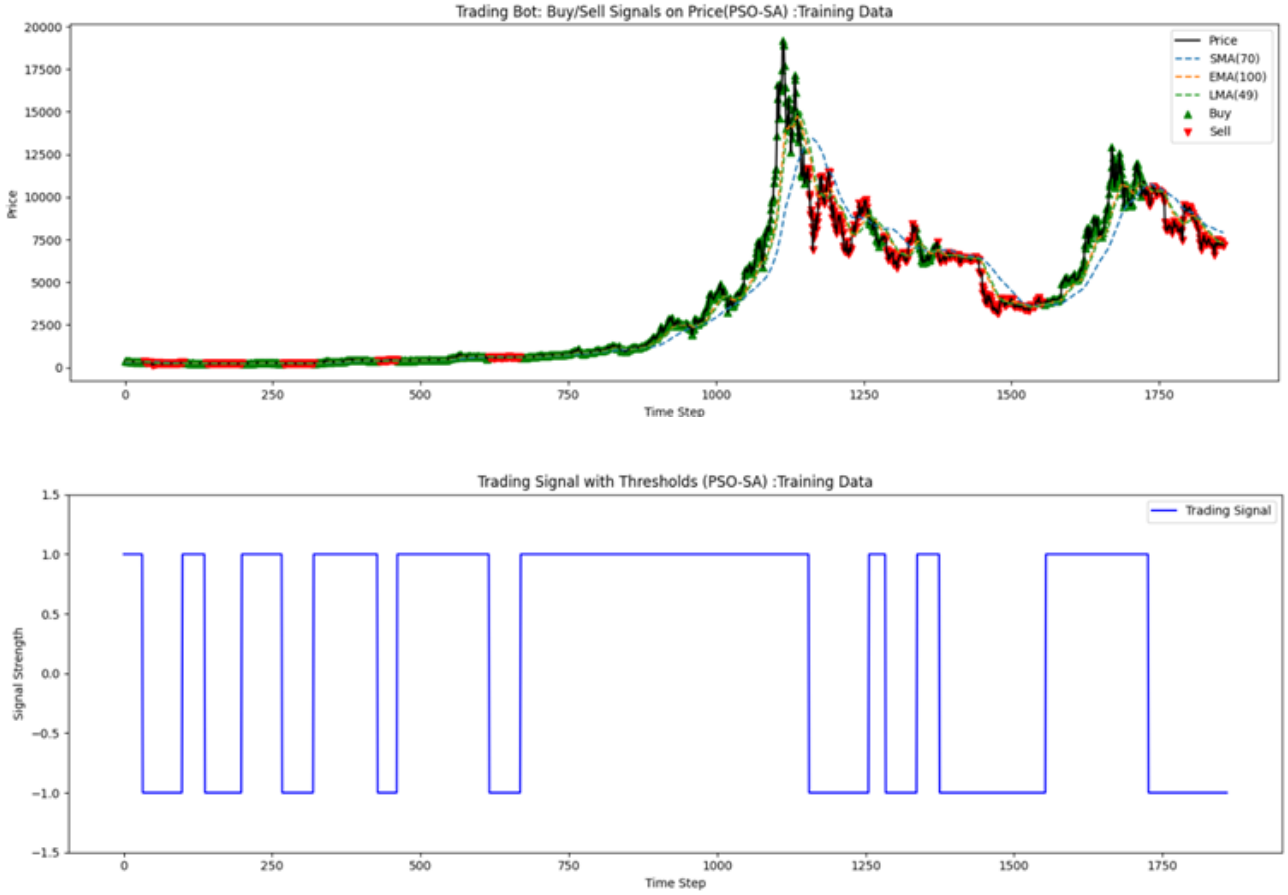

Figure 7: The trading signal and matching points on the price graph for the **training** set.

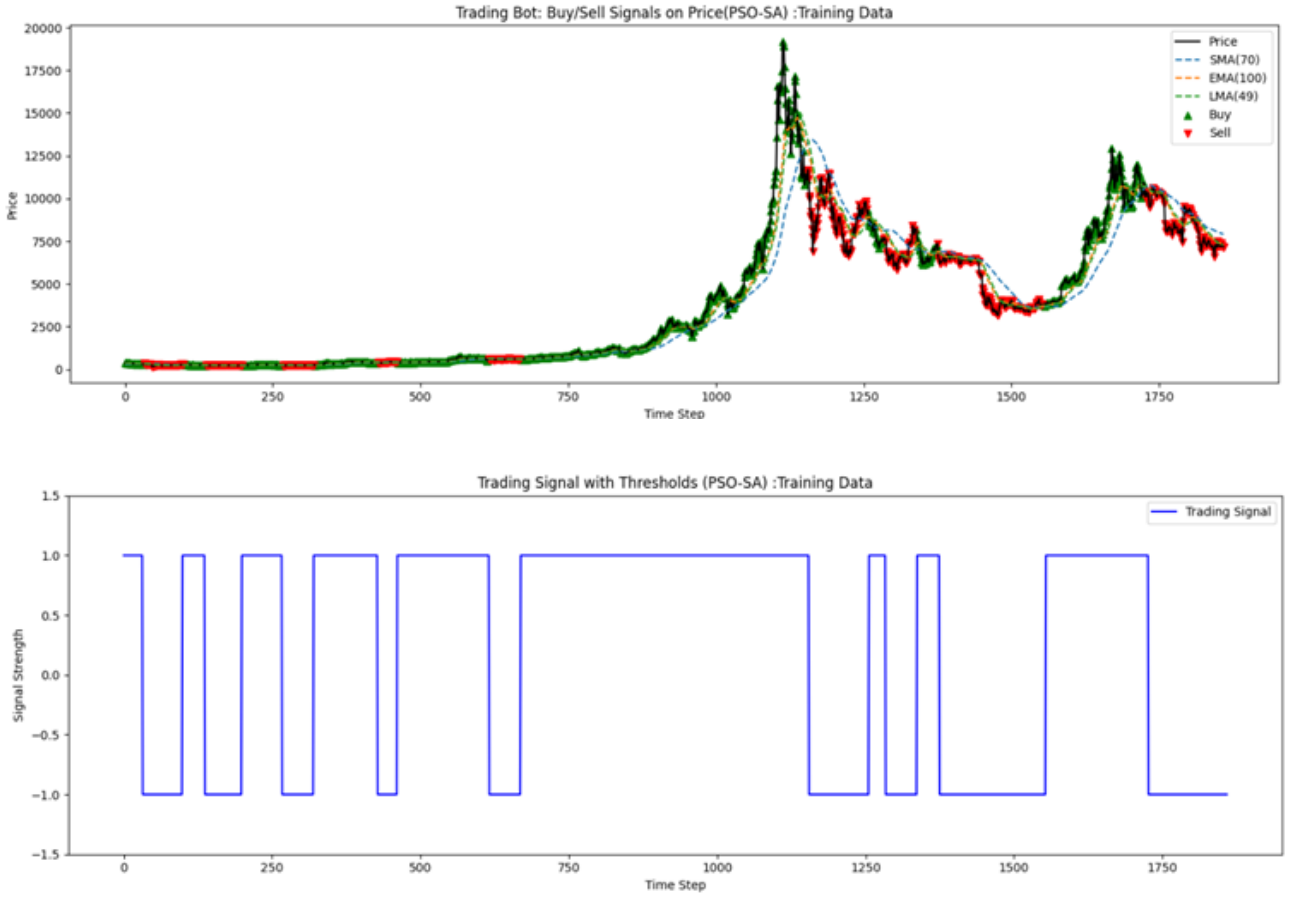

Figure 8: The trading signal and matching points on the price graph for the **test** set.

C Trading Signals for ABC

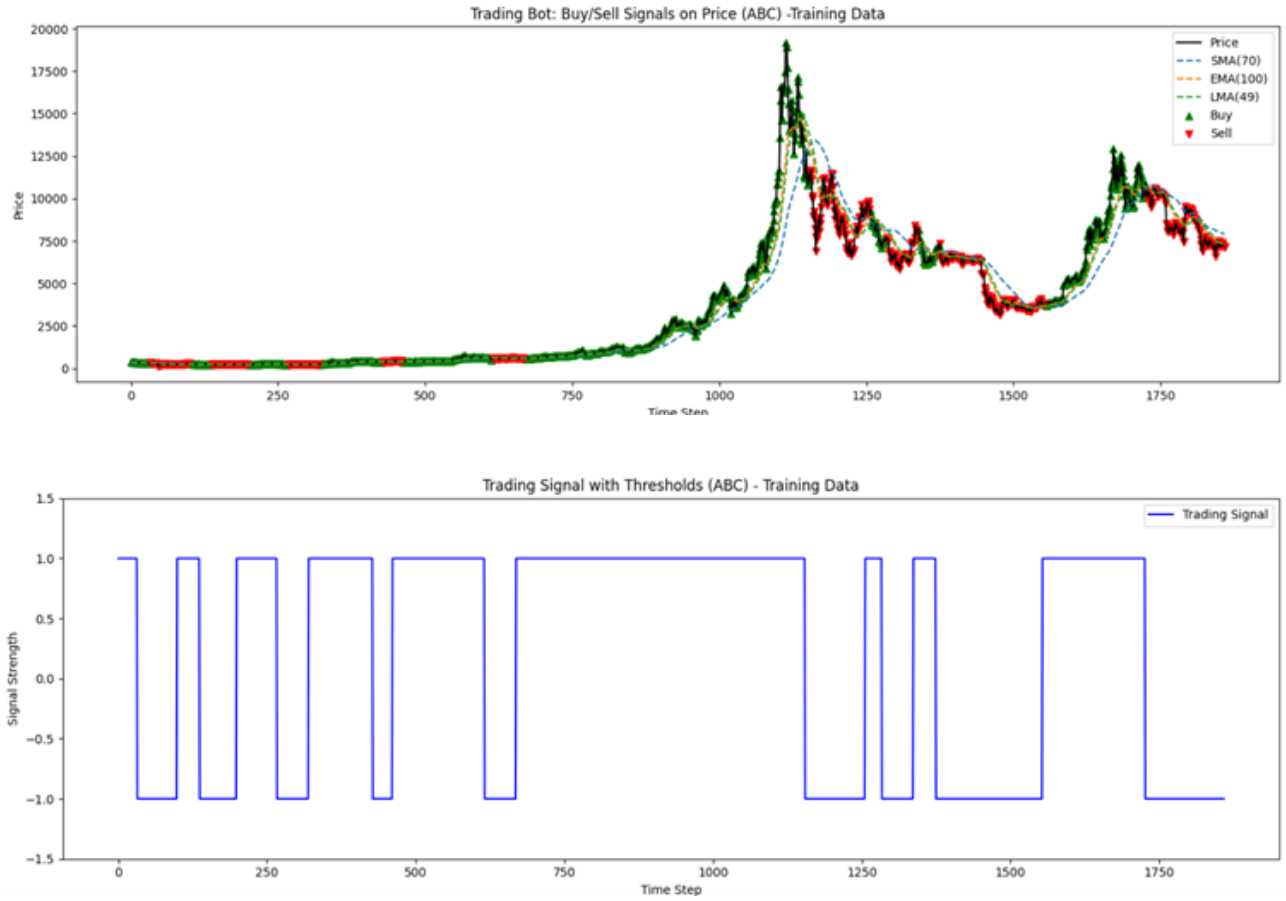

Figure 9: The trading signal and matching points on the price graph for the **training** set.

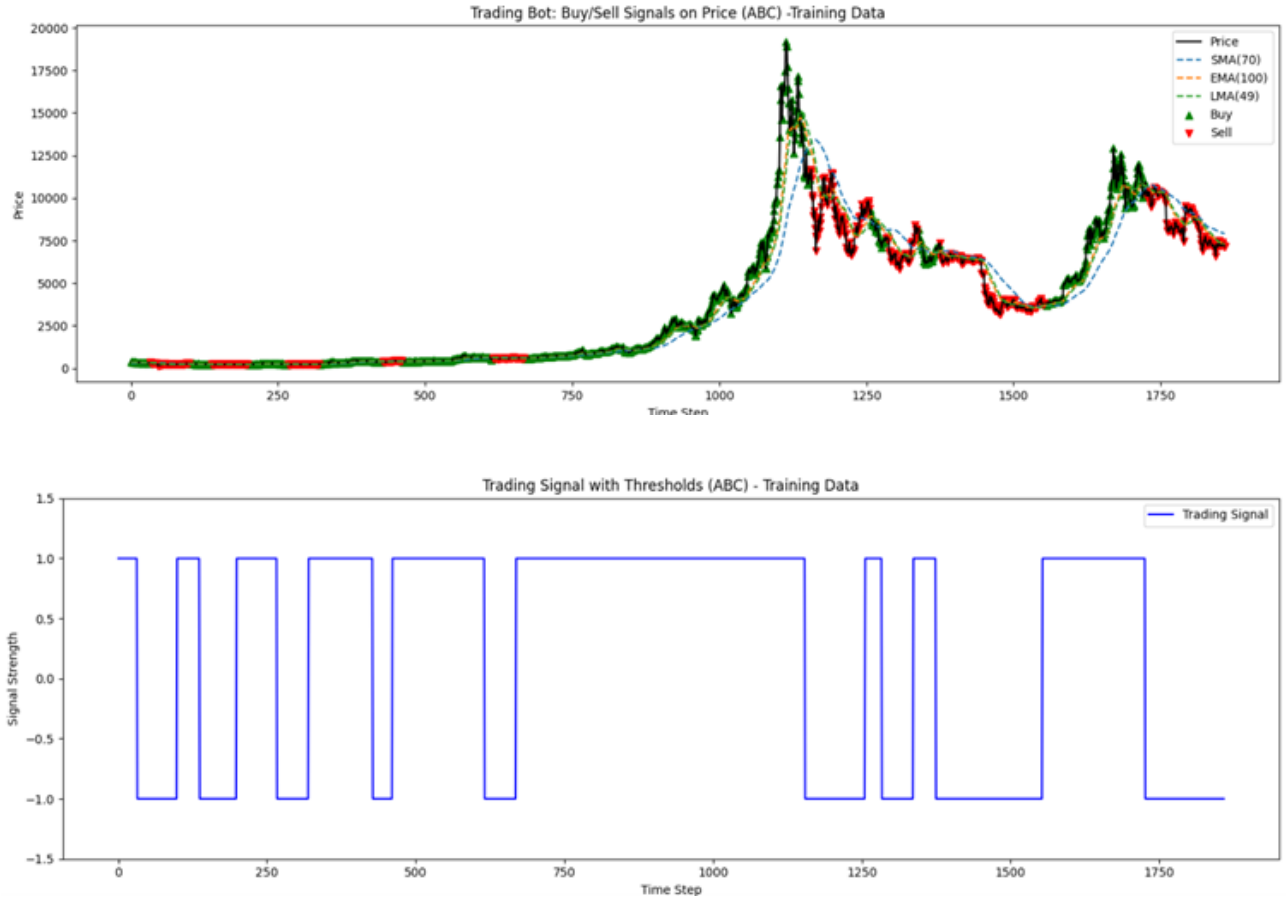

Figure 10: The trading signal and matching points on the price graph for the **test** set.