National University of Singapore

Final Report for CP3106

A Study of Cryptocurrency Investment with Dollar Cost Averaging

Shanmu Wang

Department of Computer Science

Supervised by

Prof. Bingsheng He

1 December 2022

Contents

Abstract

Cryptocurrencies are rising in the digital financial market and have played an important role in some portfolios and trading strategies. However, due to the volatility of cryptocurrency market, investors who do not want to spend too much effort on this urgently need an investment strategy that can mitigate the time risk of cryptocurrencies. In this report, we do the back-testing experiments on cryptocurrencies with Dollar Cost Averaging (DCA) method, demonstrating its potential of eliminating the time risk of investment in cryptocurrencies. To further improve the performance of DCA, we formalize the problem of finding a day with lower price to invest as an Optimal Stopping problem. And we implement Rainbow DQN, a well-performed reinforcement learning algorithm for solving this problem. Empirical results show that our agent is able to improve the profits by 2%-4% in the testing price data.

Keywords: Dollar Cost Averaging, Optimal Stopping, Deep Q-learning

1 Introduction

A **cryptocurrency** is a digital or virtual currency that is secured by cryptography, which makes it nearly impossible to counterfeit or double-spend [1]. Because crtptocurrencies can be traded 24 h a day, 7 days a week and transactions can take place between individuals, in different venues across the world, cryptocurrencies are rising in the digital financial market. Figure 1 shows the total cryptocurrency market cap from 2017.01.01 to 2022.11.01. The rapid growth of the cryptocurrency market cap has attracted a lot of investors and more and more financial institutions have included cryptocurrencies in their portfolios in recent years. On the other hand, researchers have also conducted a lot of research in cryptocurrency, including portfolio research [2], [3], and cryptocurrency trading [4], [5], etc.

Figure 1: The total cryptocurrency market cap [6]

However, as is shown in the Figure 2, cryptocurrencies also have drastic fluctuations and are considered more volatile than stocks [7]. Although the rapid fluctuations of prices can provide traders with great money-earning opportunities, but it also includes more risk [8]. For a lot of investors who do not want to spend a plethora of time focusing the change in the cryptocurrency markets, a strategy that can eliminate the risk of crytocurrencies is on demand.

Among a lot of different investment strategies, **Dollar Cost Averaging** (DCA) is widely recommended by professional investment advisors and commentators [9]. DCA requires an investor to invest the same amount of money at regular intervals, typically weekly,

Figure 2: The price change of BTC during Oct. 2022 [6]

monthly, or quarterly [10]. It allows investors to hedge against regret that results from investing a lot of money during a market high [10]. As a result, this strategy has the potential to mitigate timing risk, making it most often employed for riskier investments and suitable for cryptocurrency investment.

By following DCA, an investor ends up purchasing more shares when prices fall and fewer shares when prices rise. If the investor can invest on the day with the lowest price during each investment interval, they can end up with more shares at the end and are likely to make more profits. In practice, an investor can see the current price and previous prices and decide whether to invest today. This sequential decision-making can be formulated as an **Optimal Stopping question**, which is found in areas of statistics, economics, and financial mathematics [11]. However, this still requires investors to keep a close eye on the cryptocurrency market, which is contrary to our original intention of introducing DCA.

Since the **Reinforcement Learning** (RL) methods are widely used in dynamic decision-making processes, it is natural to use reinforcement learning algorithms to solve the optimal stopping problems. Reinforcement learning is a branch of Machine Learning. The core idea of RL algorithms is an agent learns to take action in the environment to maximize a reward signal based on the feedback from the environment. While RL algorithms are used extensively in missions of playing games, they are also widely applied in solving financial problems [7].

The remainder of this paper is organized as follows. Section 2 shows the back-testing results on cryptocurrency with DCA, demonstrating the benefits of DCA. Section 3 first formalizes the optimal stopping problem and then gives details of the Deep Q-learning RL algorithm and its variants. In Section 4, we detail how we build up the RL environment for experiments and in Section 5 we elaborate on the experiment setup. Section 6 gives the empirical results. We conclude this report in Section 7.

2 Back-testing for Dollar Cost Averaging

In order to have a clear demonstration of DCA returns, we back-tested DCA methods on different cryptocurrencies and investment portfolios, using the price data provided by CoinMarkerCap [6]. We show our simulation workflow and experiment results in the following sections.

2.1 Simulation Methodology

In our experiments, we first back-test the return of buying a single cryptocurrency, such as Bitcoin or Ethereum. Then we try an investment portfolio as detailed in [12], named Bitwise 10 Crypto Index Fund, which tracks an index of the 10 largest crypto assets and weights the assets by market capitalization. This portfolio includes established giants like Bitcoin and Ethereum, as well as up-and-coming assets. In this report, we call Bitwise 10 Crypto Index Fund Index-10 for simplicity. There is also a portfolio named based on Index-10, which sets a limitation of 20% to the weight of each cryptocurrency so that the strategy can pay more emphasis to those up-and-coming assets. We name this strategy as Max-Ratio for simplicity.

The simulation is implemented by three main functions, as is shown in Table 1. Function *ShowEffectiveness* takes an input of (BeginDate, Year, Interval), and will first generate the starting date list from *BeginDate* to the last day that can support investing for that many years. The function then passes each date in the starting date list, as well as the year and interval, to the strategy function, i.e, *AtomStrategy* and *IndexStrategy*. The function *AtomStrategy* takes an input of (StartDate, Year, Interval, Name), where *Name* specifies which cryptocurrency to invest. Then the function will simulate investing in an *Interval* from the *StartDate* for *Year*; on each investment cycle, the function will simulate investing 100 USD and get the equivalent crypto asset. At the end, the function calculate the final amount and change it to the equivalent US dollars at an average price of the last cycle, and then calculate the Return / Investment Ratio. The workflow for

function *IndexStategy* is similar, except that at every cycle it needs to read the price and market capitalization of the top 10 cryptocurrencies and will need to store the amount of each coin with a dictionary. We also add an argument of *Ratio* for this function, so that *Ratio*=1 represents the Index-10 strategy while *Ratio*=0.2 represents the Max-Ratio strategy.

Function	**Desicrition**
ShowEffectiveness	Generate the starting date list and call the strategy function
AtomStrategy	The strategy of buying a single cryptocurrency
IndexStrategy	The strategy of buying multiple cryptocurrencies based on market capitalization

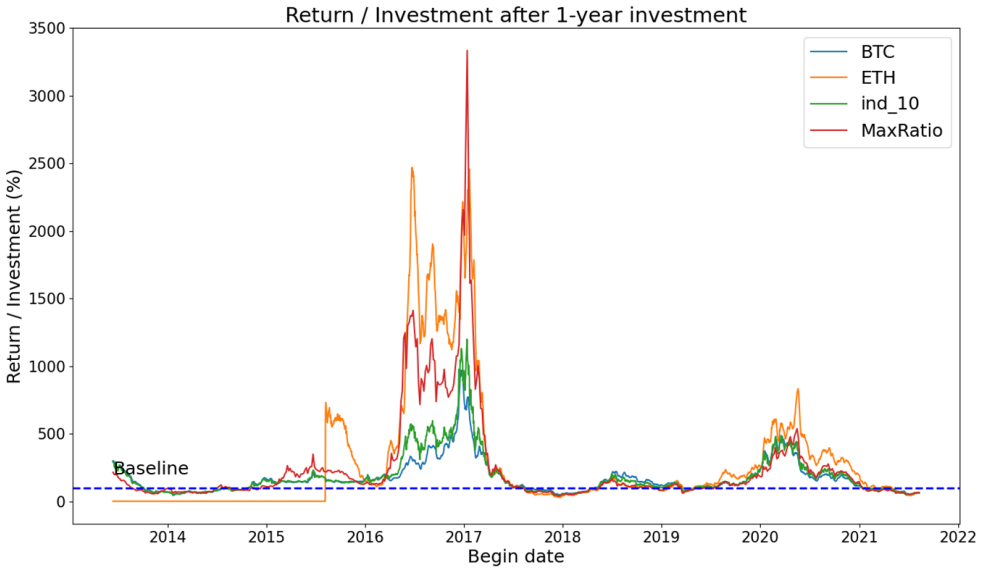
Table 1: Main functions for simulation

2.2 Simulation Setup

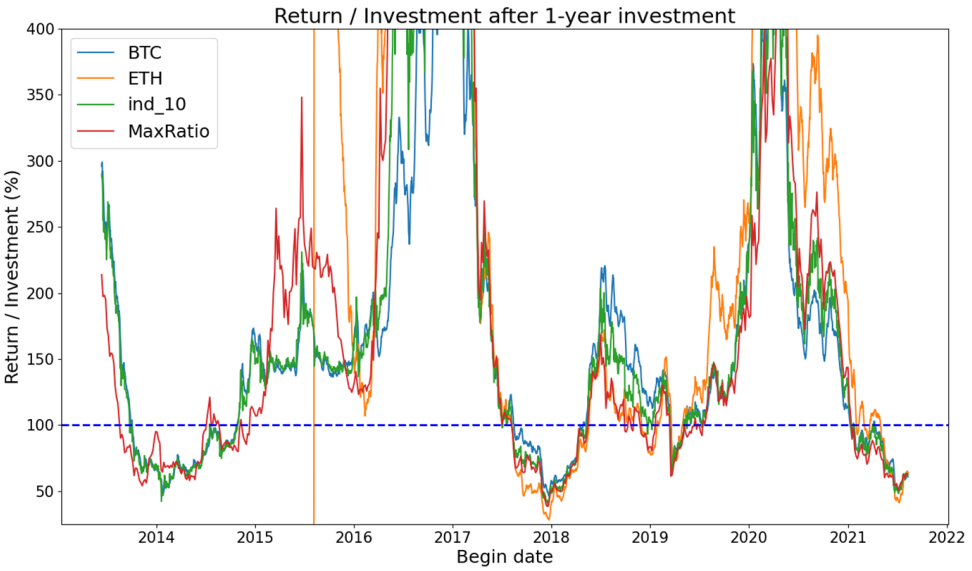
Based on the previous price and market capitalization data, we conduct back-testing experiments from 1 year to 5 years. In detail, we first choose a starting day and then simulate investing 100 USD every 10 days from that date onwards (Suppose our investment will not affect the price and market capitalization). After one or several years, we sum up our total crypto assets and exchange them for the equivalent in US dollars, and then calculate the Return / Investment ratio.

2.3 Back-testing Results

As described before, we simulate the investment strategies of buying Bitcoin, buying Ethereum, buying Index-10, and buying Max-Ratio. The result of the 1-year investment is shown in Figure 3. Each data point *(Begin date, Return / Investment)* indicates that if we start to invest on the beginning date, with an investment cycle of 10 days, and invest for 1 year, we get such a Return / Investment ratio at the end. The blue dotted line is the baseline, which is set to 100%. It is worth noting that the earliest price data available for ETH was in August 2015, so ETH curves in figures start from that time. Due to space limitations, we have included more results in Appendix A.



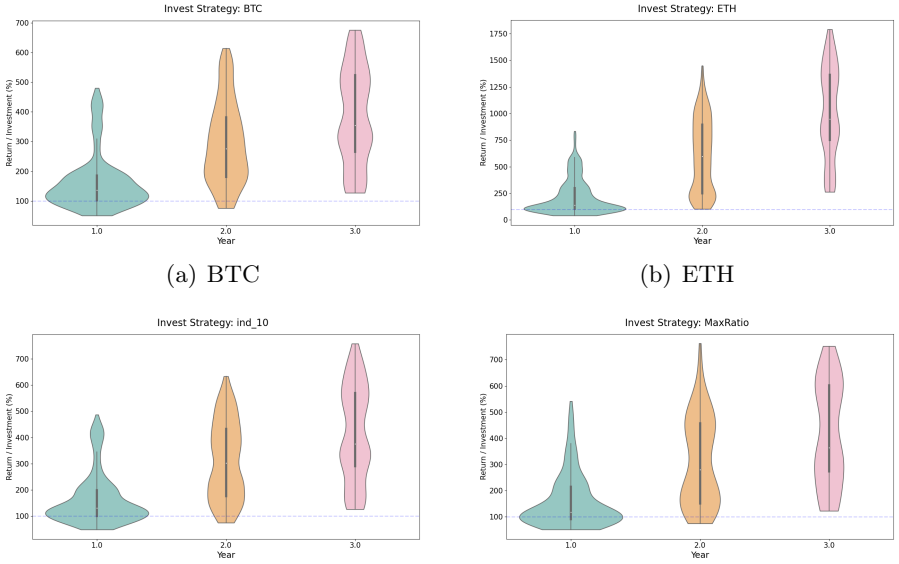
(a) Overview



(b) Details around the baseline

Figure 3: Back-testing for 1-year investment on different portfolios

Since it is obvious that investments made in the early days of cryptocurrencies can reap huge benefits, and to better demonstrate the returns, we have selected investments with a starting date after August 2018 for further analysis. The distribution of the Return / Investment for different strategies and years is shown in Figure 4. And some statistics can be found in Table 2.



(a) BTC (b) ETH

(c) Index-10 (d) Max-Ratio

Figure 4: Distribution of the Return / Investment for different strategies and years

From the distribution and statistics, we can obtain some conclusions: whether investing in a single cryptocurrency or multiple cryptocurrencies with different portfolios, DCA is effective in mitigating risk and volatility and earning benefits for investors. In detail, there is a close to 40% probability (even close to 50% if investing in ETH) of gaining 150% in just one year of fixed investment. This probability even increases to nearly 80% when the period is extended to two years. And when investing for three years, investors can even be guaranteed a solid return. To further eliminate the risk, one can also choose to start the round at different times of the year to average risk or simply extend the investment.

Overall, our back-testing results have shown that the dollar cost averaging strategy has the ability to mitigate timing risk, making it a suitable strategy for cryptocurrency investment.

Larger than		100%	150%	200%	250%	300%
Year 1	BTC	76.4%	41.3%	18.2%	13.1%	11.1%
	ETH	78.5%	48.3%	39.8%	32.2%	24.4%
	Index-10	75.2%	38.4%	24.5%	14.3%	11.1%
	Max-Ratio	63.5%	35.2%	26.9%	16.0%	11.4%
Year 2	BTC	93.4%	88.6%	71.5%	56.8%	44.7%
	ETH	100%	93.4%	88.5%	74.9%	70.8%
	Index-10	93.0%	84.0%	71.9%	56.6%	50.1%
	Max-Ratio	93.2%	74.7%	67.1%	52.7%	47.9%
Year 3	BTC	100%	88.5%	81.4%	77.3%	69.9%
	ETH	100%	100%	100%	100%	91.8%
	Index-10	100%	90.1%	85.2%	77.5%	73.2%
	Max-Ratio	100%	87.7%	78.1%	78.1%	68.5%

Table 2: Statistics of Return / Investment Ratio

2.4 Further Improvement

To further improve the performance of DCA, an intuitionally thought is to develop an agent that is able to choose a day with a lower price during each investment cycle and purchase the crypto asset on that day. In other words, every time when a new investment cycle begins, the agent will hold the money until it thinks the day is suitable to invest.

This can be formulated as an **Optimal Stopping** question, which is concerned with the problem of choosing a time to buy based on sequentially observed prices in order to maximize the amount of chosen cryptocurrency. We will detail this part in the following chapter.

3 Optimal Stopping Problem and Reinforcement Learning Methods

As mentioned before, we want to automatically choose a better day during each investment cycle to purchase the crypto asset(s), which should be able to improve the final profits. This can be formulated as an Optimal Stopping problem. In this section, we first give a formal definition of the optimal stopping problem in our scenario. Then we lay a foundation for Reinforcement Learning and detail the algorithms we used.

3.1 Problem Definition

The optimal stopping problem consists in finding the optimal time to stop in order to maximize an expected reward. This problem is found in areas of statistics, economics, and financial mathematics [11]. Some traditional way to solve the optimal stopping problem is by using a regression-based method to approximate the optimal continuation value at each state of the system. But such numerical methods do not easily scale to high dimensional problems and assume that the underlying stochastic model is known. And in most cases, it will require the fine-tuning of basis functions [13]. Recently, some researchers are using Reinforcement Learning based methods to solve optimal stopping problems.

In this work, we adopt the following notations:

- $\mathcal{A}_t$: the set of possible actions at time t
- S : the set of all possible states
- T : the horizon of the problem, i.e., the investment cycle in our scenario
- Π : the set of all possible policy $\pi : S \to \mathcal{A}$
- s_t : the state at time t
- $s_{0:T}$: a trajectory $[s_0, \ldots, s_T]$
- U_t : the payout received when stopping at time t after observing $s_{0:t}$

Specifically, $\mathcal{A}_t := \{hold, buy\}$ for $t < T$, and $\mathcal{A}_T := \{buy\}$. The stopping time with policy π is defined as: $\tau_\pi = min\{t \in [0, \ldots, T] \; s.t. \; \pi(s_t) = stop\}$. In our scenario, at each time prior to the end day of the investment cycle, the investor decides whether to invest or not, depending on whether the current payoff is greater than the continuation

value, which refers to the max payoff in the remaining days. So an optimal policy π^* should be able to do the following decisions:

$$\pi^*(s_t) = \begin{cases} \text{buy} & \text{if } \mathbb{E}[U_t \mid s_{0:t}] \geq \mathbb{E}\left[U_{\tau_\pi^{t+1}} \mid s_{0:t}\right] \\ \text{hold} & \text{otherwise} \end{cases}$$

, where $\tau_\pi^t = min\{t' \in [t, \ldots, T,\ s.t,\ \pi(s_{t'}) = buy\}$. In other words, if the payoff of stopping at the current time is greater than the maximum payoff after the current day, $\pi^*(s_t) = buy$, otherwise $\pi^*(s_t) = hold$.

The sequential decision-making problem described above can be seen as an analogy of a standard reinforcement learning notation:

$$Q(s,a) = \begin{cases} r(s,\ \text{buy}\ ), & \text{if } a = \text{ buy} \\ r(s,\ \text{hold}\ ) + \gamma \mathbb{E}\left[\max\left(Q\left(s',\ \text{buy}\ \right), Q\left(s',\ \text{hold}\ \right)\right)\right], & \text{otherwise} \end{cases}$$

In this notation, $Q(s,a)$ refers to the quality of action a under state s, $r(s,a)$ refers to the reward by taking action a under state s, s' is the next state, and γ is the discounted factor. A policy for such an optimal stopping problem is to choose the action depending on which action has a higher Q-value.

3.2 Reinforcement Learning: DQN and its variants

3.2.1 Basic Idea

Reinforcement Learning is a branch of machine learning. The basic idea of reinforcement learning is an agent learning to take action in the environment to maximize a reward signal by the feedback from the environment. In detail, at each discrete time step t, the agent observes the state s_t provided by the environment and does the action a_t, and then the environment provides the reward r_t for this action and next state s_{t+1}. Specifically, in our scenario, an agent decides whether to buy or to hold after observing the price movement and then gets the reward from this action. During the training process, the agent continuously upgrades its strategy based on feedback from the environment.

3.2.2 Deep Q-learning

The DQN (Deep Q-Network) algorithm was developed by DeepMind in 2015 [14]. Among various RL algorithms, Deep Q-leaning Network (DQN) and its variants have been proven to be useful in many missions with discrete action space. Compared to tra-

ditional Q-learning, Deep Q-learning uses a neural network to substitute the Q-table, which makes it possible for missions with large state and/or action spaces to learn Q value estimates for each state and action pair independently. Specifically, DQN uses a neural network to approximate the action values for a given state s. At each step, the agent chooses an action ϵ-greedily with respect to the action values based on the current state and adds a transition $(s_t, a_t, r_t, s_{t+1}, done_t)$ to a replay buffer, where $done_t$ indicates whether current episode ends. The parameters θ of the neural network are optimized by using gradient descent to minimize the loss $\delta_t(\theta) = \left[(y_t - Q(s_t, a_t; \theta))^2\right]$, where $y_t = r + \gamma \max_{a'} Q\left(s_{t+1}, a'; \bar{\theta}\right)$, and t is a time step randomly picked from the replay memory. $y_i - Q$ represents TD (temporal difference) error. The gradient of the loss is then back-propagated into the parameters θ of the action network. The term $\bar{\theta}$ represents the parameters of a target network, which is a periodic copy of the action network. At each step, gradient descent is performed on a mini-batch sampled uniformly from the replay buffer. The pseudo-code is shown as Algorithm 1.

Algorithm 1: Deep Q-learning with experience replay

input : A set of M episodes $\{s^i_{0:T}\}_{i=1}^{M}$
initialize: replay buffer D to capacity C of episodes
initialize: action-value network Q with random weights θ
initialize: target-value network $\bar{Q}$ with random weights $\bar{\theta}$

for *episode* i **to** M **do**
  for $t = 0$ **to** T **do**
    With probability ϵ choose a random action a_t
    Otherwise select $a_t = \arg\max_a Q\left(s^i_t, a; \theta\right)$
    Execute action a_t and observe reward r_t , store transition $\left(s^i_t, a_t, r_t, s^i_{t+1}, done_t\right)$ in D. If D is full, drop the oldest episode.
  end
  if *enough experiences in* D **then**
    Sample a random mini-batch of N transitions from D
    for *every transition* $(s_j, a_j, r_j, s_{j+1}, done_j)$ *in mini-batch* **do**
      $y_j = \begin{cases} r_j & \text{if } done_j \\ r_j + \gamma \max_{a'} \bar{Q}\left(s_{j+1}, a'; \bar{\theta}\right) & \text{otherwise.} \end{cases}$
    end
    Calculate the loos $\mathcal{L} = 1/\mathrm{N} \sum_{j=0}^{N-1} \left(y_j - Q\left(s_j, a_j\right)\right)^2$
    Perform gradient descent on $\mathcal{L}$ with respect to network parameters θ
    Every U episodes reset target network $\bar{Q} = Q$
  end
end

3.2.3 Workflow of the Basic DQN Algorithm

To make the process as clear as possible, we illustrate the workflow of a DQN agent in our scenario, as is shown in Figure 5. In our scenario, a trajectory consists of T states, where T is the investment cycle. Each time when the environment resets or the last episode ends, the training process will move to a new trajectory. And for i in 1 to T, the DQN takes as input of state s_i, and either choose an action randomly with probability ϵ or selects the action with max output Q value with probability $1 - \epsilon$. After that, the agent does the selected action and then reaches the next state s_{i+1} and gets the reward r_i. Then the transition $(s_i, a_i, s_{i+1}, r_i, done_i)$ will be stored in replay buffer, where $done_i$ indicates whether current episode ends. If $a_i = Hold$, we step to the next state and repeat the process. If we reach a day when $a_t = Buy$ or $t = T$, this episode ends and we run into a new episode to continue the training process. If there are enough experiences in the replay buffer, we sample a minibatch of transitions, compute the TD loss, and use gradient descent to update the weights of the neural networks. The rewards from the episode are then summed up to get a score for this episode, which can be used as an important indicator of the agent's training process.

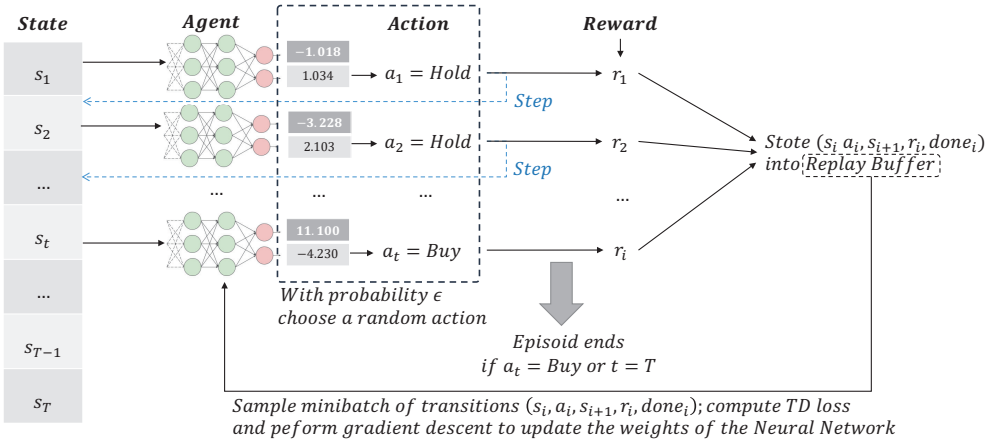


Figure 5: The workflow of a DQN agent

3.2.4 Extensions to DQN

Deep Q-learning has attracted a lot of interest since it was introduced. However, there are some problems with the basic DQN algorithm, including overestimation, long training time, and so on. Our previous experiments with basic DQN also show that the performance is not satisfactory. Actually, many researchers have proposed a lot of improvements

to the DQN algorithm. In the following paragraphs, we will give an introduction to six useful extensions to DQN, which can be fruitfully combined as the Rainbow DQN [15].

Double Q-learning: Conventional Q-learning is affected by an overestimation bias, which harms learning [15]. Double DQN [16] was introduced later that proposed the use of two different sets of parameters, one for choosing the best action and another for evaluating such actions. As in DQN, Double DQN still estimates the value ϵ-greedily according to the current values, using a network with parameters θ_t. However, Double DQN uses the second set of weights θ'_t to fairly evaluate the value of this policy. This second set of weights can be updated symmetrically by switching the roles of θ and θ' [16]. In short, it is like 2 function approximators aggregating on each other's choice of the best action, which solves the problem of overestimation and helps to learn.

Prioritized replay: As is shown in Algorithm 1, basic DQN samples mini-batch uniformly from the replay buffer. In [17], researchers proposed to use a Prioritized Experience Replay Buffer instead of a normal replay buffer. Prioritized replay gives more priority to important experiences based on TD error so that the algorithm can sample more frequently those transitions that are much to learn. To be more specific, the prioritized replay buffer samples transitions with probability p_i relative to the last encountered absolute TD error $|\delta_i|$:

$$p_i \propto \left| r_{i+1} + \gamma \max_{a'} Q(s_{i+1}, a'; \bar{\theta}) - Q\left(s_i, a_i; \theta\right) \right|^{\omega},$$

where ω is a hyper-parameter that determines the shape of the distribution [17]. Since TD error indicates how far the value is from its next-step bootstrap estimate, prioritized replay helps the agent to learn faster.

In practice, we set the priority of transition i as $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant. We add this term in order to guarantee all transactions can be possibly sampled. Then we define the probability of sampling transition i as

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}},$$

The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

However, prioritized replay introduces bias because it doesn't sample experiences uniformly at random due to the sampling proportion corresponding to TD error. We can

correct this bias by using importance-sampling (IS) weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$

These weights will be folded into the Q-learning update by using $w_i\delta_i$ instead of δ_i. And it fully compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$. In typical reinforcement learning scenarios, the unbiased nature of the updates is most important near convergence at the end of training [17]. Therefore we can define a schedule on the exponent β that reaches 1 only at the end of learning, adjusting the amount of importance-sampling correction over time.

Dueling networks: Dueling networks [18] explicitly separates the representation of state values and action advantages. The dueling architecture features two streams of computation, the value and advantage streams, sharing a convolutional encoder, and are merged by a special aggregator [15], as is shown in Figure 6. The dueling architecture represents both the value $V(s)$ and advantage $A(s, a)$ functions with a single deep model whose output combines the two to produce a state-action value $Q(s, a)$. The value function V measures how good it is to be in a particular state s, and the advantage function A measures the relative advantage of action a. This final Q value can be computed by:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s, a'; \theta, \alpha)\right),$$

where θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers, and $|\mathcal{A}|$ is the shape of action space.

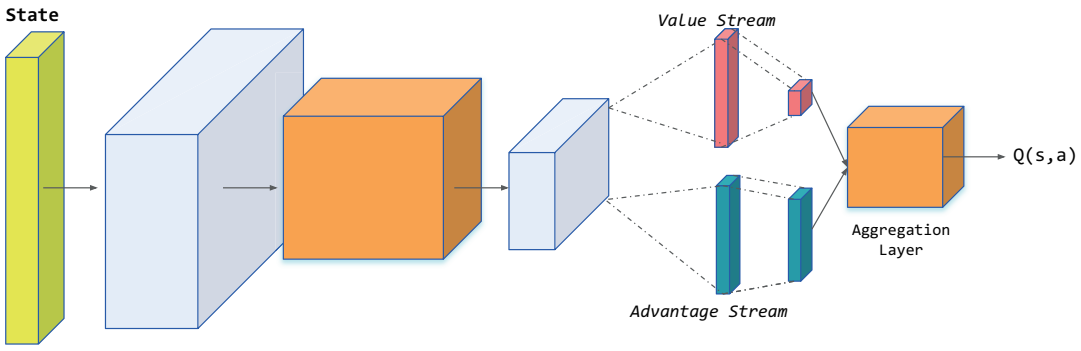


Figure 6: The dueling architecture

Multi-step learning: Multi-step learning [19] proposed to use n-step return rather than using 1-step return to calculate Q values so that the target value does not rely on just the current reward and can be more accurate. We define the truncated n-step return from a given state s_t as

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} r_{t+k+1}.$$

A multi-step variant of DQN is then defined by minimizing the alternative loss:

$$\left(r_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q\left(s_{t+n}, a'; \bar{\theta}\right) - Q\left(s_t, a_t; \theta\right) \right)^2$$

Distributional RL: Distributional RL introduces the use of distribution of Q values instead of using average estimated Q values since these values can be diverse in a different situation and hence causing the value to be inaccurate [15].

In [20], the authors argued the importance of learning the distribution of returns instead of the expected return. In detail, they model the value distribution using a discrete distribution parameterized by $N \in \mathbb{N}$ and $V_{\text{MIN}}, V_{\text{MAX}} \in \mathbb{R}$, and whose support is the set of atoms $\{z_i = V_{\text{MIN}} + i \triangle z : 0 \leq i < N\}, \triangle z := \frac{V_{MAX} - V_{MIN}}{N-1}$ [20]. A distributional variant of Q-learning is then derived by first constructing a new support for the target distribution, and then minimizing the Kullbeck-Leibler divergence between the distribution d_t and the target distribution

$$d_t' = (r_{t+1} + \gamma_{t+1} z, p_{\hat{\theta}}(s_{t+1}, \hat{a}_{t+1}^*)),$$

$$D_{KL}(\phi_z d_t' \| d_t).$$

Here ϕ_z is a L2-projection of the target distribution onto the fixed support z, and $\hat{a}_{t+1}^* = \arg\max_a Q(s_{t+1}, a; \hat{\theta})$ is the greedy action. Due to the space limitation, we refer readers to [20] for more details.

The parameterized distribution can be represented by a neural network, as in DQN, but with $atom\_size \times action\_dim$ outputs. A *softmax* is applied independently for each action dimension of the output to ensure that the distribution for each action is appropriately normalized. To estimate Q-values, we can use the inner product of each

action's softmax distribution and the set of atoms $\{z_i\}$:

$$Q(s_t, a_t) = \sum_i z_i p_i(s_t, a_t),$$

where p_i is the probability of z_i, i.e, the output of softmax.

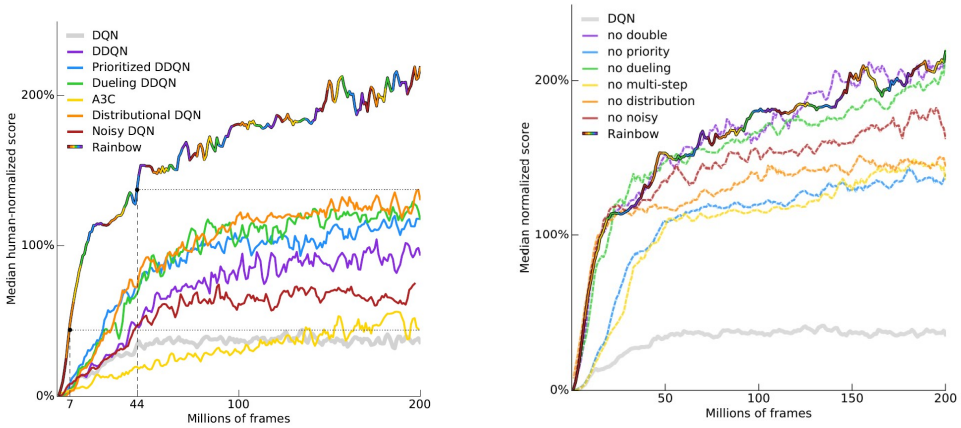
Noisy Nets: Noisy Nets [21] were introduced to add noise to the network parameters so that it explores more. Such noise is applied to the output streams of the network, which is typically a liner layer. A normal linear layer of a neural network with p inputs and q outputs is represented by $y = wx + b$, where $x \in \mathbb{R}^p$ is the layer input, $w \in \mathbb{R}^{q \times p}$, and $b \in \mathbb{R}$ the bias. Then the corresponding noisy linear layer is defined as:

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b,$$

where $\mu^w + \sigma^w \odot \epsilon^w$ and $\mu^b + \sigma^b \odot \epsilon^b$ replace w and b in the first linear layer equation, and $\odot$ denotes the element-wise product. The parameters $\mu^w \in \mathbb{R}^{q \times p}, \mu^b \in \mathbb{R}^q, \sigma^w \in \mathbb{R}^{q \times p}$ and $\sigma^b \in \mathbb{R}^q$ are learnable, whereas $\epsilon^w \in \mathbb{R}^{q \times p}$ and $\epsilon^b \in \mathbb{R}^q$ are noise random variables. Over time, the network learns to ignore the noisy stream at different rates in different parts of the state space, allowing state-conditional exploration with a form of self-annealing [15].

Rainbow DQN: The Rainbow DQN [15] makes a combination of the aforementioned extensions and provides very good performance in many missions. The performance of Rainbow DQN across 57 Atari games is shown in Figure 7, which shows that all of the extensions contribute to the final performance and prioritized replay buffer, multi-step learning, and distributional RL are more important. Each extension and its contribution is summarized in Table 3.

Since each of these extensions has been proven useful, in our project, we add all of these extensions to the basic DQN. In detail, the network structure is constructed with the thoughts of dueling network and distributional RL, while some of the linear layers are replaced with noisy linear layers. We use two networks to do the selection and evaluation separately. The replay buffer is changed to the prioritized buffer and we use the multi-step return to calculate Q values. We will detail the network structure and hyperparameters in Section 5.



(a) Rainbow DQN and other baselines

(b) Rainbow DQN and ablations

Figure 7: The performance of Rainbow DQN [15]

Extension	**Description**	**Contribution**
Double Q learning	Use two networks to decouple the selection from the evaluation	Tackle overestimation bias
Prioritized Replay	Replay buffer that gives more priority to important experiences based on TD error	Prioritize important transitions; help agent learn faster
Dueling Networks	Use two streams to separately estimate state-value and the advantages for each action	Lead to stable and fast learning
Multi-step learning	Use n-step return instead of 1-step return to calculate Q-value	Help agent learn faster
Distributional RL	Use the distribution of Q values instead of using average estimated Q values	Enable accurate Q-value estimation
Noisy Nets	Add noise to the network parameters	Lead to state-conditional exploration

Table 3: The summary of the extensions

4 Reinforcement Learning Environment

4.1 Overview of the Environment

As stated before, the core idea of RL algorithms is to interact with the environment, update iteratively, and eventually obtain a trading strategy to maximize the expected return. So it is vital to set up an environment for any RL mission. OpenAI Gym [22] provides standardized environments for various DRL tasks. In this project, we build a standard OpenAI gym-style environment named *CryptoEnv*. In addition to implementing the basic functions, such as reset and step, we also implement some functions to facilitate the training and evaluation process. The main APIs are given in Table 4. More details are elaborated as follows.

Table 4: The main APIs in CryptoEnv

Function	Description
env=CryptoEnv(data)	Return an environment instance of the CryptoEnv class with price data and default t_{window} and T_{cycle}.
env.reset()	Reset the environment and return a random episode
env.step()	Move a step, return the reward and next state
env.getPrice()	Obtain the original and normalized price list of the current cycle

4.2 State

When applying reinforcement learning to financial problems, the agent's state is primarily modeled with price-based features [23]. In our scenario, we also adopt a price-based feature with the form of $s_t = [remaining\ time, relative\ price, window\ prices]$. On day t, we obtain the state s_t as follows:

1. **remaining time:** this is computed by $(T-t)/T$, where T is the investment cycle and t indicates which day are we in this cycle.

2. **relative value:** to get the relative price, we first select the price on the day before the start of the current investment cycle as reference price p_r, and then subtract the reference price from today's price p_t. After that, we use a *sigmoid* function to normalize the difference and get the relative value:

$$relative\ price = sigmoid(p_t - p_r)$$

3. **window price:** window price is a list of price data for the previous t_{window} days

from today. And a Max-Min normalization is applied so that the price data should range from 0 to 1.

In conclusion, our state is a $1 \times (t_{window} + 2)$ vector with all of its elements ranging from 0 to 1.

4.3 Reward

The reward is fully governing the agent's behavior; thus, a good choice of reward function is critical for the desired type of strategy and its performance. Since we attempt to implement an agent that can find the lowest priced investment day within an investment cycle, it is intentional to reward our agent if it can get a low price to invest or hold our money when faced with a high price, and punish the agent if it does the opposite.

In detail, our reward scheme is based on the *logit* function $\mathcal{F}(x) = ln\frac{x}{1-x}$, which is actually the inverse of the sigmoid function. The image of this function is shown as Figure 8. In our scenario, the reward scheme is as follows:

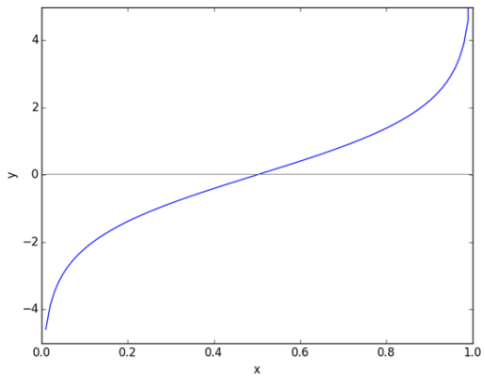


Figure 8: The image of the logit function

1. First we get the prices of the current investment cycle and do a Max-Min-Normalization so that each price is normalized to 0 to 1. Let p_i is the normalized price of $i-th$ day.

2. If the agent decide to buy on the $i-th$ day or i equals to investment cycle T, we calculate the reward through: $r = -\mathcal{F}(p)$.

3. If the agent decide to hold on the $i-th$ day, we calculate the reward through: $r = \alpha\mathcal{F}(p)$, where $\alpha \in [0, 1]$. The factor α is to prevent dilution of the final

reward, and we set it as 0.5 based on the empirical results.

We now give an example to show our reward scheme:

Example: Let's say the agent is in an episode with the following original price list p:

$$p = [1200.37, 1175.95, 1187.87, 1187.13, 1205.01, 1176.9, 1169.28, 1167.54, 1172.52],$$

where p_1 is the first day's price, p_2 is the second day's price, and so on. After the Max-Min-Normalization, we get the normalized price list p':

$$p' = [0.876, 0.224, 0.543, 0.523, 0.999, 0.250, 0.046, 0.001, 0.133].$$

Note that we adjust the normalized price from 1 to 0.999 and 0 to 0.001 in order to avoid the divide-by-zero error. Let's assume our agent holds on the first day and decide to buy on the second day in this investment cycle:

1. **Hold on the first day**: we have $p'_1 = 0.876$, which indicates the first day's price is a very high price during the current cycle. So if our agent decides to hold our money on this day, we should give a reward instead of punishment. The reward is: $r_1 = \alpha\mathcal{F}(p'_1) = 0.5 \times ln\frac{0.876}{1-0.876} = 0.978$
2. **Buy on the second day:** we have $p'_2 = 0.224$. Though not the best time to invest, this day is still a good choice. So we compute the reward as: $r_2 = -\mathcal{F}(p'_2) = -ln\frac{0.224}{1-0.224} = 1.243$

Since our agent decides to buy, this episode ends, and we can compute the score of this episode as $score = \sum_{i=1}^{t} r_i$, where $t = min\{t \in [1, \ldots, T, s.t, Q(s_t) = Buy\}$.

5 Experiment Setup

5.1 Hyperparameters

Based on the hyperparameters chosen by other reinforcement learning methods applied for financial problems and our empirical results, the hyperparameters for our mission are determined as shown in Table 5.

Hyperparameters		**Value**
Environment Parameters	t_{window}	30
	T_{cycle}	9
Basic Parameters	learning rate	5×10^{-4}
	memory size	10000
	batch size	128
	target update step U	100
	discounted factor γ	0.95
Prioritized Experience Replay	α	0.2
	β	0.6
	ϵ	1×10^{-6}
Distributional RL	V_{MIN}	0
	V_{MAX}	20
	atom size	51
N-step Learning	n step	3

Table 5: Hyperparameters of our experiments

5.2 Network Structure

The network architecture in our experiment is shown as Figure 9, where we use *Relu* for activation. In our scenario, the state input is a 1×32 vector. The feature layer is a

normal linear layer. The feature will be fed to two streams: the value stream consists of two noisy linear layers, and output the value function with a size of $1 \times atom\_size$, i.e, 1×51; the advantage stream has the symmetric architecture but outputs a vector of $1 \times (action\_dim \times atom\_size)$, i.e, 1×102. These two outputs are then aggregated to obtain the final $Q(s, a)$ as is mentioned in Section 3.2.4.

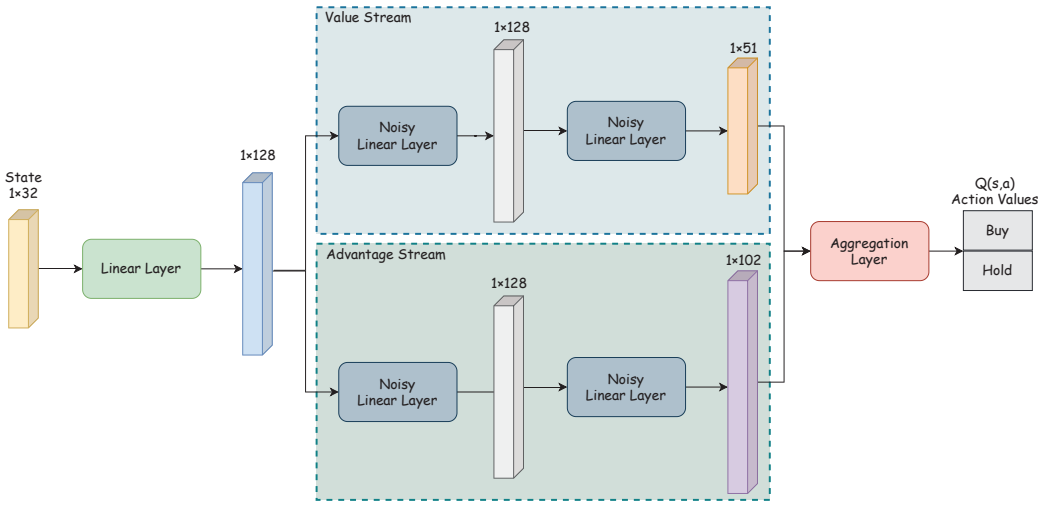


Figure 9: The network architecture

5.3 Data Splitting

Unlike missions of playing games, where reinforcement learning algorithms perform very well, there is a limit to the amount of data we can use in a financial problem. In some game environments, we could easily train for millions of steps and also test numerous times. However, most RL algorithms designed for financial problems take price-based features as the state, and the price data, in most cases, is limited. For our scenario, this limitation is even more apparent due to the short history of cryptocurrencies.

Since Bitcoin and Ethereum are two cryptocurrencies that attract a lot of attention in the cryptocurrency market, we conduct our experiments based on these two currencies. Bitcoin has a record price dating back as far as 2013, and we trace back 3472 days of price data from 2013.05.06 to 2022.11.07. However, early price changes are too old to be a good reference and even might harm the training if involved, so those early price data are discarded. The rest of the data is split into training and test data at a ratio of 0.85 to 0.15. At last, we split the price data of Bitcoin as follows:

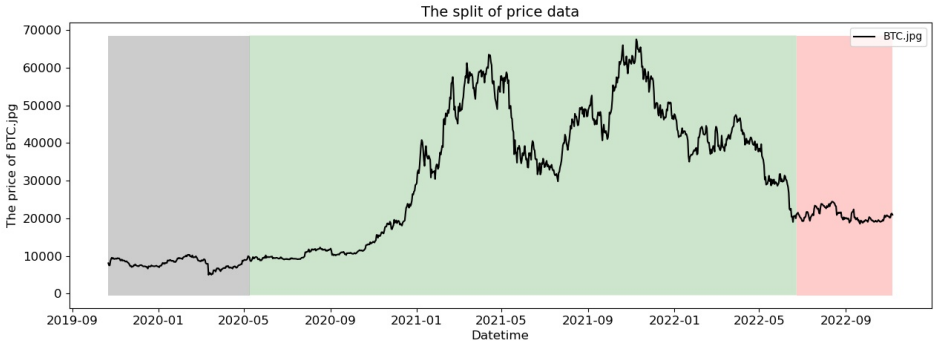
1. **Training:** from 2020.05.09 to 2022.06.23, 775 days
2. **Testing:** from 2022.06.24 to 2022.11.07, 136 days

The price data of Ethereum can be traced back to 2015.08.07. And we split the price data of Ethereum as follows:

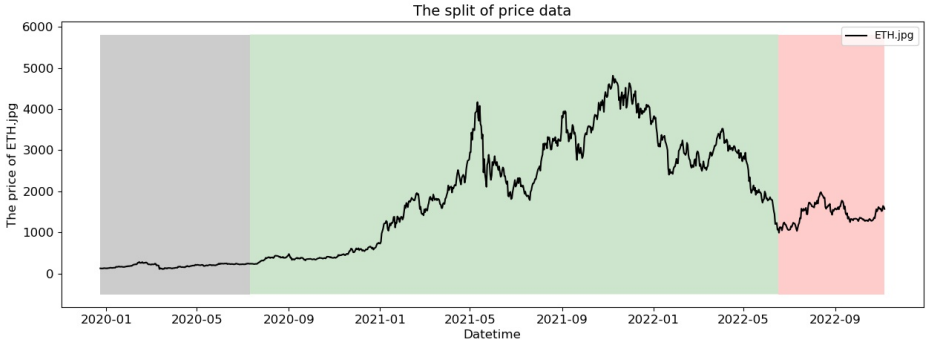
1. **Training:** from 2020.07.11 to 2022.07.02, 721 days
2. **Testing:** from 2022.07.03 to 2022.11.07, 127 days

In both cases, price data prior to the training period is not considered during training or testing.

The data splitting is shown in Figure 10, where the data shaded in green is used for training, and the data shaded in red is used for testing.



(a) For BTC price data



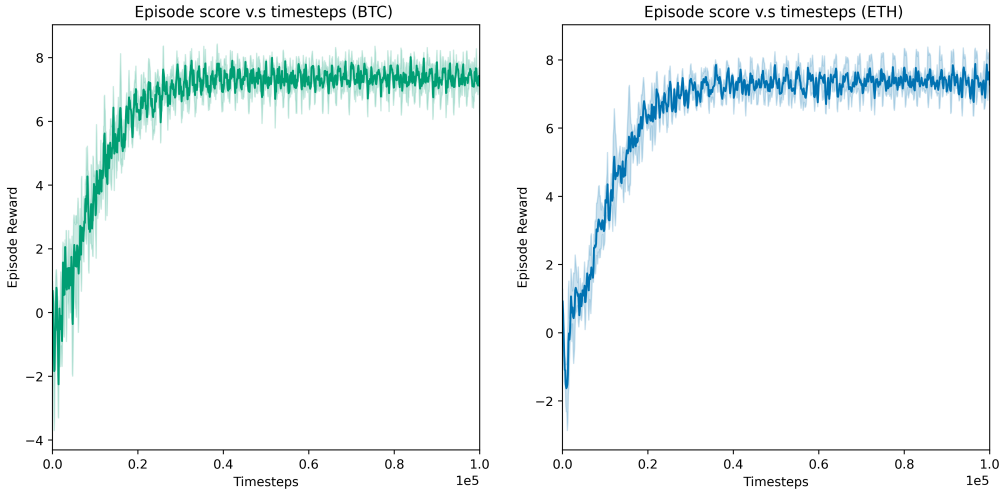
(b) For ETH price data

Figure 10: The data splitting for training and testing

6 Empirical Results

6.1 Training Results

As mentioned before, we train our Rainbow DQN agent for 100,000 steps, and the episode score is shown as Figure 11. We do 3 experiments with different seeds, and the line in the figure is the average of 3 experiments while the shade indicates the standard error. As is shown in the figure, in both environments, the agent begins to converge after around 30,000 steps and reaches an average episode score of around 7. And the volatility of the score in the BTC environment is larger than that in the ETH environment, which could be due to the greater range of price fluctuations in BTC.



(a) Score changes in BTC environment (b) Score changes in ETH environment

Figure 11: The change of episode score over the timesteps. Curves are smoothed with a moving average over 10 points

6.2 Testing Results

Then we apply the trained models to the test data respectively, assuming that we start to invest on the first day.

Test results on BTC: the best results on BTC is shown as Figure 12. The buying days selected by our agent are marked with dots, where a red dot indicates a price below the average price of the current cycle and a green dot indicates the opposite. Among 15 investments during the test period, there are 12 times we buy with a price lower than

the average price.

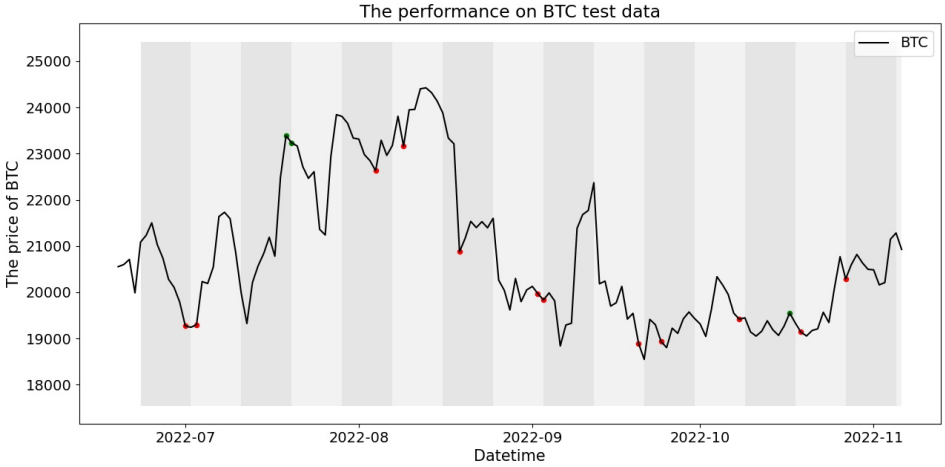


Figure 12: The performance of the trained agent on BTC test data

Starting on the first day and following our trained agent, the amount we end up earning after 15 cycles of investment, compared to always investing on the first or last day or investing at the average price (hypothetically), is shown in Table 6.

Table 6: The performance of the trained agent on BTC test data

Compared with	Improvement
Buy on the first day	2.29%
Buy on the last day	2.92%
Buy on a random day	2.64%
Buy on average price	1.65%

Then we use the test price data to generate a test environment, which will generate all the possible episodes, i.e, all 9 consecutive days in the test period. We test our trained agent on all the episodes, and for each episode, we compute the improvement compared with buying on the first day, last day, random day, and buying with average price. The average of the improvements over all the episodes is shown in Table 7.

Table 7: The average performance over all episodes on BTC test data

Compared with	Improvement
Buy on the first day	2.30%
Buy on the last day	2.25%
Buy on a random day	1.92%
Buy on average price	2.25%

Test results on ETH: similarly, the best results on ETH are shown as Figure 13. Among 14 investments during the test period, there are 10 times we buy with a price lower than the average price.

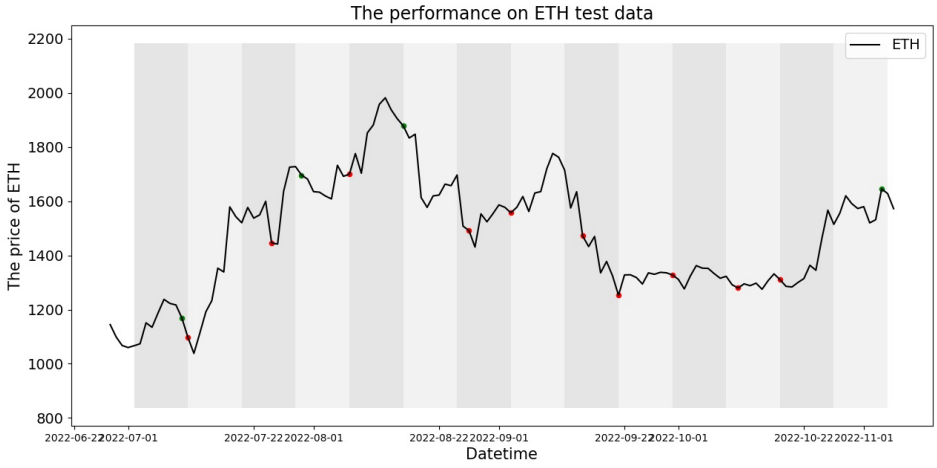


Figure 13: The performance of the trained agent on ETH test data

Again, starting on the first day and following our trained agent, the amount we end up earning after 14 cycles of investment, compared to always investing on the first or last day or investing at the average price (hypothetically), is shown in the Table 8.

Table 8: The performance of the trained agent on ETH test data

Compared with	Improvement
Buy on the first day	1.16%
Buy on the last day	6.47%
Buy on a random day	2.98%
Buy on average price	2.83%

Also, we use the ETH test price data to generate a test environment and test our trained agent on all the episodes. The average of the improvements over all the episodes is shown in Table 9.

Table 9: The average performance over all episodes on ETH test data

Compared with	Improvement
Buy on the first day	1.22%
Buy on the last day	6.71%
Buy on a random day	3.89%
Buy on average price	4.71%

Statistics show that our agents can earn over 2%-4% compared to investing at average prices, which is a decent performance. In conclusion, by applying our agent to the test price data that it had never seen before, we show that our Rainbow DQN agents have learned how to make profits among the weekly or monthly fluctuations of the cryptocurrency market, and also the models have generalization capability

6.3 Test with Different Investment Cycles

As mentioned in Section 4, our state is set as

$$s_t = [remaining\ time, relative\ price, window\ prices]$$

The remaining time is computed by $(T - t)/T$, where T is the investment cycle and t indicates which day are we in this cycle. Since the remaining time is normalized by dividing T, it is possible to conduct some experiments to test our trained agent's performance with different investment cycles.

Our agents are trained in the environment with the t_{window} of 30 days and T_{cycle} of 9 days. We test our agents on the test price data, but with an investment cycle different from 9 days. Specifically, the investment cycle ranges from 5 days to 14 days. The results are shown in Table 10. The statistics show that even though the investment cycle is different from that during the training process, our agents still earn more profits compared to investing at the average price of each cycle. However, the performance of the agent does degrade when it is applied to a different investment cycle.

One possible reason for this degradation is that the possible values of the remaining time are discrete and are determined as the cycle is determined. To explain it further, since

the remaining time is computed by $(T-t)/T$, all the possible values during the training process is $[0.89, 0.78, 0.67, 0.56, 0.44, 0.33, 0.22, 0.11, 0]$. However, the possible values of the remaining time change a lot even with a slight change of T, which might account for the degradation of the performance.

Test set	**Compared with**	$T=9$	$T=5$	$T=6$	$T=7$	$T=8$
BTC Test	Buying on first day	2.29%	1.09%	0.66%	1.46%	1.03%
	Buying on last day	2.92%	0.77%	0.77%	0.74%	1.85%
	Buying on random day	2.64%	0.95%	0.76%	0.74%	1.87%
	Buying on average price	1.65%	1.10%	1.26%	0.90%	1.05%
ETH Test	Buying on first day	1.16%	-0.55%	-1.08%	-0.48%	-1.06%
	Buying on last day	6.47%	1.72%	2.38%	1.76%	2.28%
	Buying on random day	2.98%	0.62%	-1.71%	-0.70%	0.70%
	Buying on average price	2.83%	0.90%	0.43%	0.41%	1.04%
Test set	**Compared with**	$T=10$	$T=11$	$T=12$	$T=13$	$T=14$
BTC Test	Buying on first day	1.77%	0.42%	0.60%	4.47%	1.90%
	Buying on last day	1.45%	0.18%	0.64%	2.47%	1.37%
	Buying on random day	4.84%	2.50%	-0.97%	1.18%	0.20%
	Buying on average price	2.42%	2.10%	1.06%	2.40%	0.38%
ETH Test	Buying on first day	-2.33%	-0.49%	-0.34%	-2.36%	-0.58%
	Buying on last day	2.47%	3.56%	4.94%	1.36%	3.68%
	Buying on random day	2.00%	5.00%	4.13%	3.00%	3.36%
	Buying on average price	1.62%	3.67%	3.85%	0.19%	1.35%

Table 10: Test agent with different investment cycles

6.4 Summary of Results

In this section, we have shown the effectiveness of reinforcement learning methods for solving optimal stopping problems. To be more specific, we demonstrate that our method is able to help investors to find a lower price to invest during each investment cycle when they are following the dollar cost averaging strategy, and thus help them to grab more profits. Combined with our RL method, DCA can be a strategy that can not only eliminate the timing risk but also improve the probability of earning profits.

7 Conclusions

In this report, we have done back-testing experiments of cryptocurrency investment with the dollar cost averaging method, demonstrating that this method has the ability to mitigate timing risk in cryptocurrency markets. We formalized the problem of finding a day with a lower price to invest as an optimal stopping problem and implemented a Rainbow DQN agent to solve this problem and conduct experiments based on Bitcoin and Ethereum price data. We trained and tested the agent on the historical price data, which shows a decent performance. We also tested the performance of our trained agents under different investment cycles. The results also outperformed buying at the average price of each cycle. This work will be implemented into *Kreek*, a startup by the research group. The future work includes training the model based on other cryptocurrencies, training the model with different investment cycles, and so on.

8 References

[1] J. Frankenfield, *Cryptocurrency*, Investopedia, May 2022. [Online]. Available: https://www.investopedia.com/terms/c/cryptocurrency.asp.

[2] W. Liu, 'Portfolio diversification across cryptocurrencies,' *Finance Research Letters*, vol. 29, pp. 200–205, 2019.

[3] A. Kajtazi and A. Moro, 'The role of bitcoin in well diversified portfolios: A comparative global study,' *International Review of Financial Analysis*, vol. 61, pp. 143–157, 2019.

[4] I. Makarov and A. Schoar, 'Trading and arbitrage in cryptocurrency markets,' *Journal of Financial Economics*, vol. 135, no. 2, pp. 293–319, 2020.

[5] K. Mannaro, A. Pinna and M. Marchesi, 'Crypto-trading: Blockchain-oriented energy market,' in *2017 AEIT International Annual Conference*, IEEE, 2017, pp. 1–5.

[6] Coinmarketcap, *Cryptocurrency market capitalizations | coinmarketcap*, CoinMarketCap, 2022. [Online]. Available: https://coinmarketcap.com/.

[7] X.-Y. Liu, H. Yang, J. Gao and C. D. Wang, 'Finrl: Deep reinforcement learning framework to automate trading in quantitative finance,' in *Proceedings of the Second ACM International Conference on AI in Finance*, 2021, pp. 1–9.

[8] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu and R. Brooks, 'A brief survey of cryptocurrency systems,' in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, 2016, pp. 745–752. DOI: 10.1109/PST.2016.7906988.

[9] M. J. Brennan, F. Li and W. N. Torous, 'Dollar cost averaging,' *Review of Finance*, vol. 9, no. 4, pp. 509–535, 2005.

[10] K. B. Leggio and D. Lien, 'An empirical examination of the effectiveness of dollar-cost averaging using downside risk performance measures,' *Journal of Economics and Finance*, vol. 27, no. 2, pp. 211–223, 2003.

[11] C. Herrera, F. Krach, P. Ruyssen and J. Teichmann, 'Optimal stopping via randomized neural networks,' *arXiv preprint arXiv:2104.13669*, 2021.

[12] *Bitw | bitwise 10 crypto index fund*, Bitwise Investments. [Online]. Available: https://bitwiseinvestments.com/crypto-funds/bitw (visited on 11/11/2022).

[13] A. Fathan and E. Delage, 'Deep reinforcement learning for optimal stopping with application in financial engineering,' *arXiv preprint arXiv:2105.08877*, 2021.

[14] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, 'Human-level control through deep reinforcement learning,' *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] M. Hessel, J. Modayil, H. Van Hasselt *et al.*, 'Rainbow: Combining improvements in deep reinforcement learning,' in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[16] H. Van Hasselt, A. Guez and D. Silver, 'Deep reinforcement learning with double q-learning,' in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[17] T. Schaul, J. Quan, I. Antonoglou and D. Silver, 'Prioritized experience replay,' *arXiv preprint arXiv:1511.05952*, 2015.

[18] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot and N. Freitas, 'Dueling network architectures for deep reinforcement learning,' in *International conference on machine learning*, PMLR, 2016, pp. 1995–2003.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[20] M. G. Bellemare, W. Dabney and R. Munos, 'A distributional perspective on reinforcement learning,' in *International Conference on Machine Learning*, PMLR, 2017, pp. 449–458.

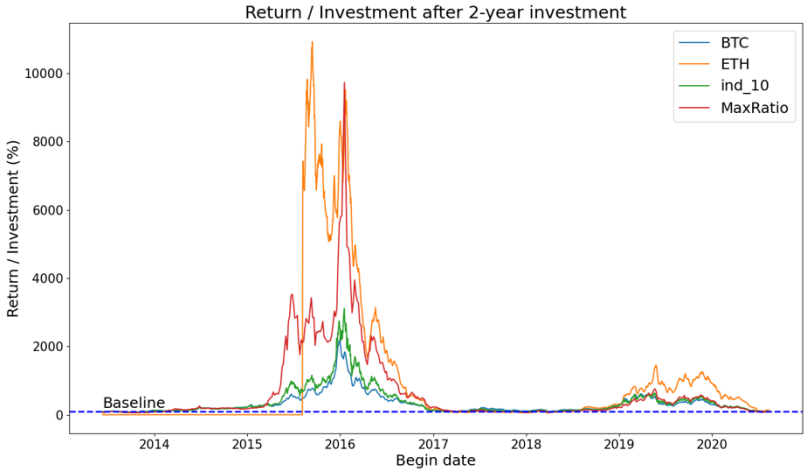
[21] M. Fortunato, M. G. Azar, B. Piot *et al.*, 'Noisy networks for exploration,' *arXiv preprint arXiv:1706.10295*, 2017.

[22] G. Brockman, V. Cheung, L. Pettersson *et al.*, 'Openai gym,' *arXiv preprint arXiv:1606.01540*, 2016.

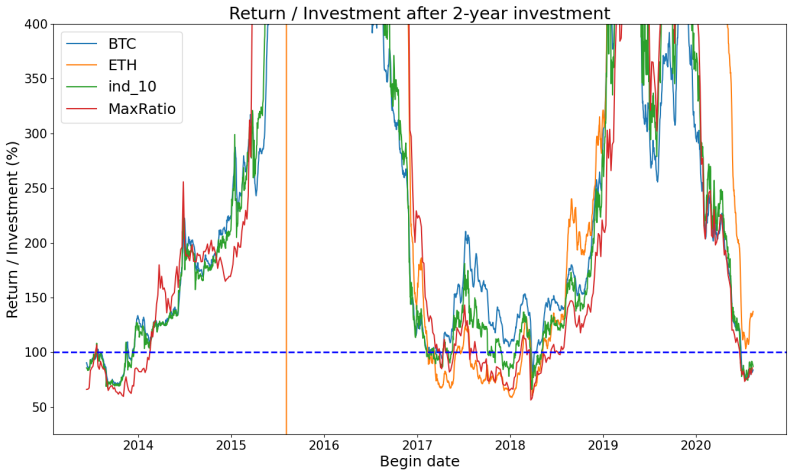
[23] T. G. Fischer, 'Reinforcement learning in financial markets-a survey,' FAU Discussion Papers in Economics, Tech. Rep., 2018.

A Back-testing Results for More Years

Back-testing results for more years are shown as follows. These results show that Dollar Cost Averaging (DCA) method is able to eliminate the timing risk in cryptocurrency investment and the longer the investment period, the higher the chance of obtaining high returns.

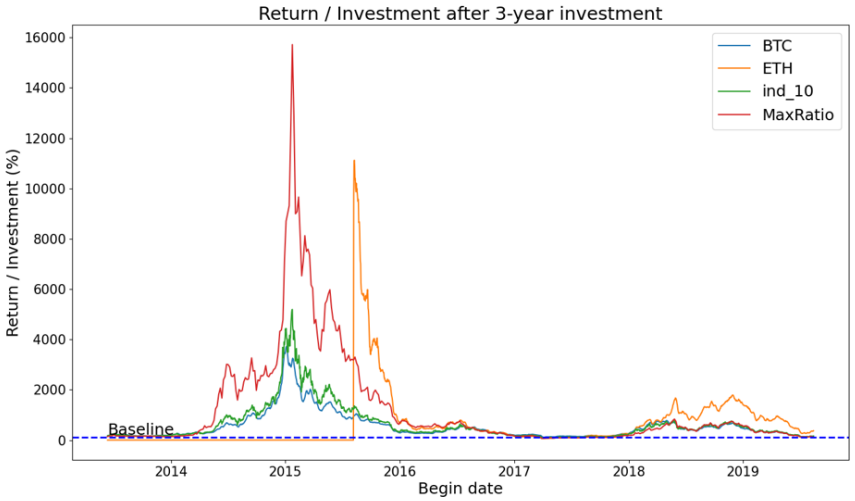


(a) Overview

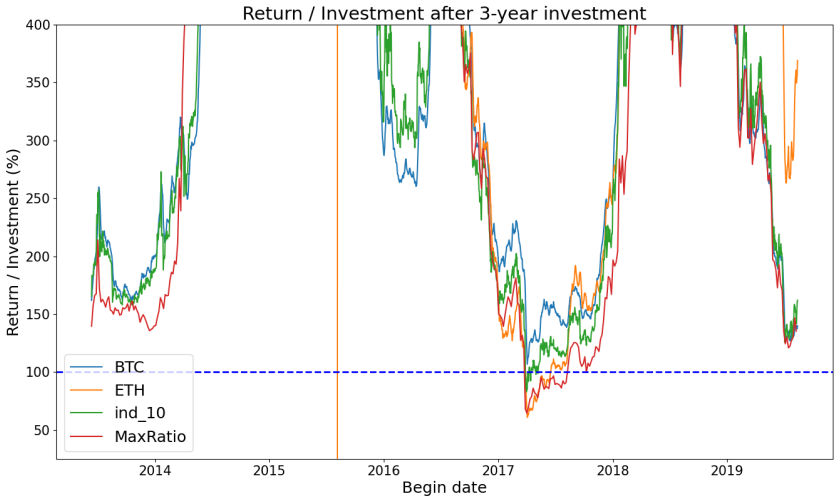


(b) Details around the baseline

Figure 14: Back-testing for 2-year investment on different portfolios

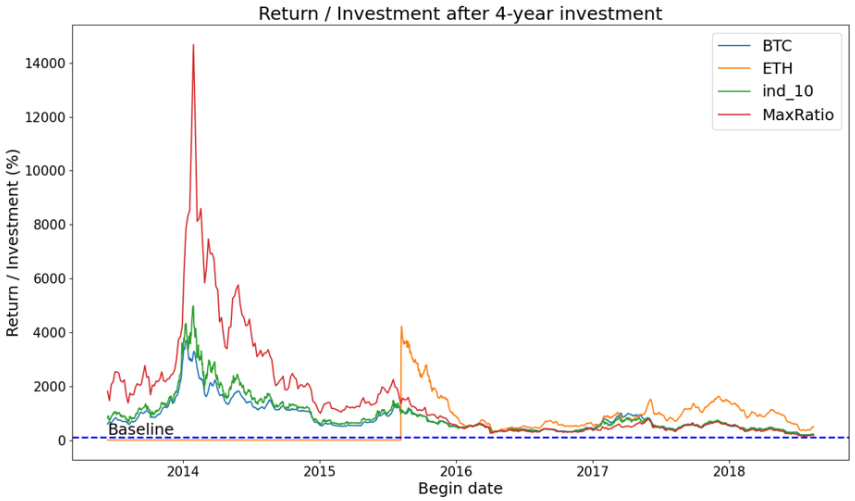


(a) Overview

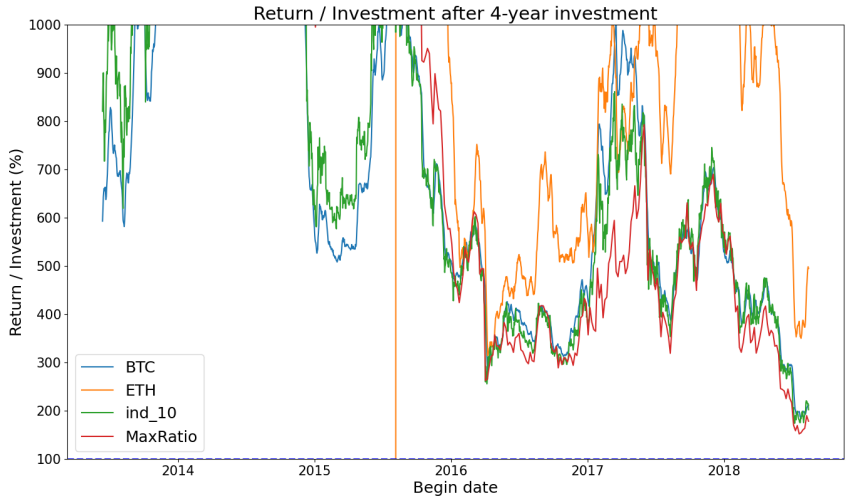


(b) Details around the baseline

Figure 15: Back-testing for 3-year investment on different portfolios

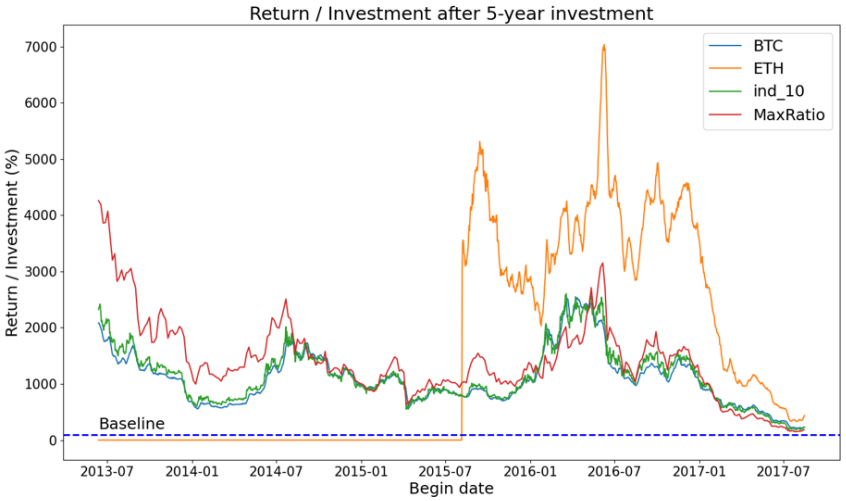


(a) Overview

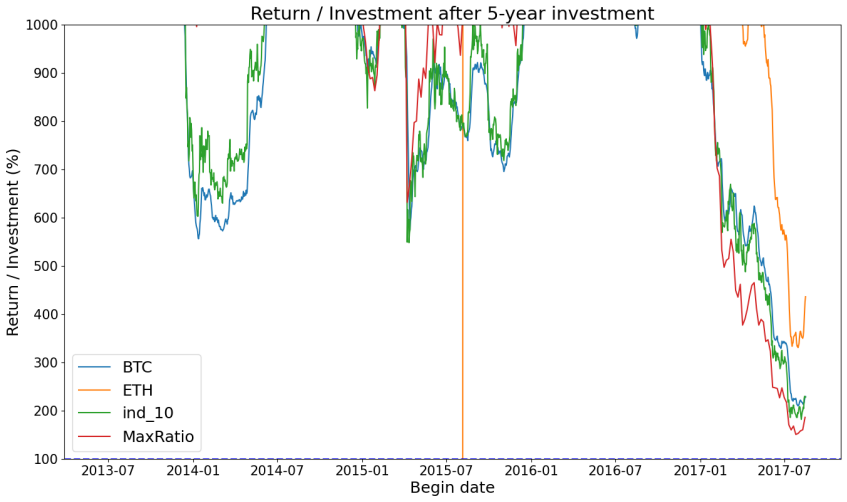


(b) Details around the baseline

Figure 16: Back-testing for 4-year investment on different portfolios



(a) Overview



(b) Details around the baseline

Figure 17: Back-testing for 5-year investment on different portfolios