POLITECNICO DI TORINO

Master's Degree in Ingegneria Matematica

Master's Degree Thesis

Dynamic Financial Index Tracking

Supervisor

Prof. Edoardo FADDA

Candidate

Edoardo VAY

October 2025

Summary

This thesis investigates the index tracking problem through a novel approach based on the principles of dynamic programming, with a focus on modern deep-reinforcement learning techniques. Given the increasing popularity of passively managed funds like exchange-traded funds, finding innovative ways to replicate a selected financial index efficiently is a significant challenge in today's computational finance. This approach comes with its own set of difficulties, primarily related to data requirements, as the method's great flexibility comes at the cost of significant computational time and the large volume of data necessary for training. Nevertheless, building such a model allows for the use of a highly realistic transaction cost model that is even capable to incorporate taxation: this makes the model also suitable for private investors, who are usually the most affected by the model simplifications typically found in the literature.

The work begins by laying out the mathematical formulation of the problem and transitioning to the dynamic programming approach, where the fixed-point Banach theorem is used to calculate the correct value of the transition costs. Subsequently, the reinforcement learning framework is explored, studying the proximal policy optimization method as a training strategy to enable the actor to learn the correct policy. Finally, the computational experiments are carried out in both a controlled synthetic environment, where it is possible to inspect in detail if the actor is learning the intended policy correctly, and with real-world data.

Acknowledgements

Table of Contents

List of Tables

List of Figures

Acronyms

ETF
Exchange Traded Fund

RTE
Return Tracking Error

DP
Dynamic Programming

RL
Reinforcement Learning

DLR
Deep Reinforcement Learning

PPO
Proximal Policy Optimization

TRPO
Trust Region Policy Optimization

NN
Neural Network

GAE
General Advantage Estimator

Chapter 1

Introduction

The fundamental problem of portfolio management has always been a cornerstone of quantitative finance. Optimizing a portfolio by balancing expected return against risk has been a central topic of mathematical interest since the field's inception in finance and economics.

Within portfolio management, two distinct philosophical approaches prevail: active management and passive management. Active management aims to outperform a market benchmark by strategically selecting assets. This approach relies on the manager's skill in analyzing market information to identify and exploit perceived inefficiencies or asymmetric information, thereby gaining an "edge" over the market.

Conversely, passive management does not seek to outperform the market. Instead, its objective is to replicate the performance of a specific benchmark as closely as possible. This strategy is theoretically grounded in the Efficient Market Hypothesis, which states that asset prices already reflect all available information, implying that consistently achieving excess returns through skill is statistically unlikely. Index funds are the primary instruments that fall into this latter category.

Index funds have gained a considerable amount of attention and popularity in recent years. Their primary objective is to replicate the performance of a benchmark index while minimizing the discrepancy between the fund's return and that of the index, a metric commonly known as tracking error. These funds are mainly structured as mutual funds or Exchange-Traded Funds (ETFs) and to achieve their goal, two main replication strategies are employed:

- **Full Replication**: this strategy consists in holding every asset in the benchmark index, matching the exact weights of the index composition. It is highly effective and widely used for indices composed of liquid securities, such as the S&P 500, and is adopted by major funds like the iShares Core S&P 500 UCITS ETF. The main advantage of this strategy is precision, but it can

become impractical or costly when dealing with less liquid assets or when frequent rebalancing leads to significant transaction costs.

- **Representative Sampling**: to mitigate the disadvantages of full replication, this approach involves holding a carefully selected subset of the assets from the index that is designed to mimic its overall characteristics (e.g., sector allocation, risk profile). While this method reduces transaction costs and simplifies portfolio management, it inherently carries a higher potential for tracking error. Nevertheless, it is the preferred strategy for very broad or less liquid indices and is used by some of the largest funds by AUM, such as the Vanguard Total Stock Market Index Fund (with AUM of more than $1.9 trillion) (Peng et al. 2024).

Due to the dynamic nature of financial markets and periodic changes in the index composition itself, the portfolio of an index fund cannot remain static. Fund managers must periodically rebalance the portfolio weights to minimize the tracking error. While rebalancing could theoretically occur at a high frequency, in practice, its periodicity is limited by transaction costs, which have a direct negative impact on returns.

1.1 Literature review

Despite this intrinsically dynamic process, a significant and large portion of the literature on the Index Tracking Problem, with some notable exceptions such as Yao et al. (2006), approaches it from a static perspective. This simplification, however, presents its own set of challenges. Static formulations cannot take into account the long-term trade-off between transaction costs and the tracking error and moreover, the vast majority of the literature does not include non-proportional transaction costs as is often the case in reality or completely ignores transaction costs all together. The static model inability to directly incorporate complex transaction costs, combined with the fact that, in a static setting, it is not possible to include a multi-period analysis, leads to the necessity of artificial cardinality constraints that restrict or penalize the number of assets in the portfolio beyond a set limit. Unfortunately this kind of constraints have a significant problem: there is no clear theoretical reason for imposing them (e.g., a realistic model would not require any cardinality constraint if the transaction costs are low enough) and they are not suitable in a multi-period context where a specific subset of assets could be optimal for a period but not for the next one.

1.2 Problem Description

In this thesis we consider the model introduced by Peng et al. 2024 as it aims to solve most of the problems that emerge from the static formulation. The objective of this formulation is to minimize the tracking error of the managed portfolio over multiple periods of time and, at the same time, include in the model the most realistic transaction cost possible. By using the dynamic programming paradigm, it is possible to reformulate the problem: the model is discrete-time with an infinite horizon, i.e., the index fund is treated as if it must continue existing and performing indefinitely.

The dynamic formulation of this kind of financial problems is very difficult to implement especially due to the high dimensionality of the state and decision variables that, in this case are also continuous-valued. For this reasons the solving method proposed is Deep Reinforcement Learning (DRL), an extension of reinforcement learning that uses neural networks to learn and model the value function and/or the policy distribution. In our setting the RL agent dynamically controls the weight of each asset in the portfolio according to the state variable observed. The RL method is particularly well-suited for adapting to major structural shifts in financial markets, such as the COVID-19 crisis. In fact, the framework enables the agent to adapt its policy in response to market changes, utilizing all relevant information, including trading volumes and treasury bill rates, which can be fully exploited thanks to this approach.

A key challenge of this approach is tied to the financial setting in which we find ourselves: usually RL problems are used in situations where a very large amount of data is available to train the algorithm. Since this is not the case, the thesis adopts the novel approach proposed by Peng et al. (2024), which relies on a rolling window to increase the apparent size of the available data for training and that incorporates the value function at the terminal state of each training episode to implement the infinite horizon.

The conceptual framework presented in this section establishes the key parts of the dynamic index tracking problem, including the handling of transaction costs and the management of portfolio weights via a DRL agent. In the following chapters, these ideas will be formalized using rigorous mathematical notation, defining the state variables, decision rules, portfolio dynamics, and objective function. This formalization lays the foundation for the implementation of the reinforcement learning methodology described later in the thesis.

Chapter 2

Index Tracking Problem

The primary objective of this section is to provide a comprehensive and rigorous mathematical description of the index tracking problem as it is approached in this thesis. We will systematically define all the key components of the model, including the state variables, the agent's available actions, the dynamics of the market environment, and the objective function to be optimized. A clear and unambiguous formulation is essential, as it lays the foundation for the dynamic programming and reinforcement learning methodologies developed in the following chapters.

To ensure clarity and maintain consistency with the relevant literature, a deliberate choice has been made regarding the notation. This thesis will adopt the notation conventions in the work of Peng et al. 2024. This approach facilitates a direct comparison of our contributions with this foundational study.

2.1 Mathematical notation

In this section is discussed the mathematical notation used to describe the index tracking problem. The fund manager wants to track the performance of a given index using N assets. At the start of the period t he will choose the new value for the weights w_t to maintain during a period of length M days so that he can dynamically adapt to the market conditions. During the M days of that period t the manager can choose with which frequency to rebalance the portfolio in order to maintain the actual portfolio weights w_t as close as possible to the decision taken at the start of the period t. In order to keep the notation less complicated the following formulation will assume that the rebalancing will happen every day in order to correct the discrepancies due to the asset's price movements. Each trading day inside a time period $[t, t+1]$ is indexed as $t + \frac{k}{M}, k = 0, ..., M-1$. For every asset $i = 1, ..., N$, the price of the stock is denoted by $p_{i,t+\frac{k}{M}}$.

To leave no ambiguity, the minus sign $-$ after the time index is used to indicate whether the value of what is indexed is before or after the rebalancing has happened. Indeed two different value can happen in the same day: for example $w_{i,(t)-}$ is the portfolio weight of the asset i at time t before the new weight is chosen, and $w_{i,t}$ in the weight of the asset i at time t after the new weight is chosen. Likewise, using the same notation, we will refer to $V_{(t+\frac{k}{M})-}$ and $V_{(t+\frac{k}{M})}$ as the value of the whole portfolio before and after the rebalancing happens and the variable $x_{i,t+\frac{k}{M}}$ will represent the number of share of asset i at a specific time.

Finally the $r^{TP}_{t+\frac{k}{M}}$ and $r^{I}_{t+\frac{k}{M}}$ are the return of the tracking portfolio and the Index respectively and they are as

$$r^{TP}_{t+\frac{k+1}{M}} = \frac{V_{(t+\frac{k+1}{M})-}}{V_{(t+\frac{k}{M})-}}, \qquad r^{I}_{t+\frac{k+1}{M}} = \frac{I_{t+\frac{k+1}{M}}}{I_{t+\frac{k}{M}}}. \tag{2.1}$$

This definition follows the formulation in Chiam et al. 2013 and it is important to note that the value of the tracking portfolio return at time $t + \frac{k+1}{M}$ is only influenced by the transaction cost observed at time $t + \frac{k}{M}$ because it includes the value of the portfolio before the rebalancing takes place; this is very important in shaping in the correct way the reward for the RL agent.

2.2 Return Tracking Error Formulation

This thesis adopts the **Return Tracking Error** (**RTE**) as its performance metric. This metric measures the average daily difference between the returns of the tracking portfolio and those of the index and is defined as

$$\mathbf{RTE}^{q}_{(t_1,t_2)} = \left[\frac{1}{M(t_2-t_1)} \sum_{k=1}^{M(t_2-t_1)} \left| r^{\mathrm{TP}}_{t_1+\frac{k}{M}} - r^{\mathrm{I}}_{t_1+\frac{k}{M}} \right|^{q}\right]^{\frac{1}{q}} \tag{2.2}$$

where $t_1, t_2 \in \{t + \frac{k}{M} : t = 0, 1, 2, \ldots, k = 0, 1, ..., M-1\}$ and $t_1 < t_2$ and the parameter q is usually 1 or 2.

No single metric is perfect for every situation: as noted in Boyde 2021, the usefulness of this particular metric heavily depends on the investors' investment horizon (e.g. a value based metric is more suitable for an investor with a very long horizon while an short investment horizon prefers that the fund movements follows as close as possible the daily oscillations). While this thesis focuses on the return tracking error, the proposed methodologies are flexible and can be easily adapted for other metrics with little to any modification.

Chosen the selected metric, the found manager problem formulation is the following:

$$\min_{\{w_t \in \mathcal{F}_t : t=0,1,\dots\}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{RTE}^q_{(t,t+1]}\right] \tag{2.3}$$

$$\text{s.t.} \quad w_{i,t} \geq 0, i = 1, ..., N, \sum_{i=1}^{N} w_{i,t} = 1, t = 0,1, ..., \tag{2.4}$$

$$V_{(t+\frac{k}{M})-} - c_{t+\frac{k}{M}} = V_{t+\frac{k}{M}}, t = 0,1, ..., k = 0, ..., M-1, \tag{2.5}$$

$$x_{i,t+\frac{k}{M}} p_{i,t+\frac{k}{M}} = w_{i,t+\frac{k}{M}} V_{t+\frac{k}{M}}, i = 1, ..., N, t = 0,1, ..., k = 0, ..., M-1, \tag{2.6}$$

$$w_{i,t+\frac{k}{M}} = w_{i,t}, i = 1, ..., N, t = 0,1, ..., k = 0, ..., M-1, \tag{2.7}$$

where in the Equation 2.3 the objective function is the expected value of the **Return Tracking Error**.

To precisely analyze the model, the role of every constrain is analyzed:

- constraints 2.4 ensure that all weights w_i are non-negative (meaning no short positions are allowed) and that their sum equals 1 at every time period (i.e. the portfolio is fully invested at all times with no cash position);
- constraints 2.6 links the number of shares held for each asset and their respective prices to the total value of the portfolio at each point in time;
- constraints 2.7 implements the manager's decision on the target weights made at the start of each period t. In this base formulation, the target allocation is reapplied daily to compensate for price fluctuations. However, the model in this thesis introduces flexibility, allowing the manager to choose the desired rebalancing frequency within each period $[t_s, t_{s+1}]$ allowing for a more precise analysis of the impact of transaction costs;
- constraints 2.5 formalizes the impact of transaction costs, connecting the portfolio's value before and after the rebalancing procedure. The quantity $c_{t+\frac{k}{M}}$ quantifies the total cost incurred during this operation. In practice, transaction costs are typically non-linear functions that depend on multiple factors, including traded volumes, and their precise structure varies according to the financial intermediary. Given this complexity, this thesis implements and analyzes two distinct models for the cost function:
  1. **Linear Model**: this includes a fixed cost and a component proportional to the traded value. Such a formulation is widely adopted in the academic literature due to its simplicity and analytical tractability; in this thesis this model will often be used just as a time saving measure being that the computational power available was limited;

2. **Realistic Non-Linear Model**: this function is designed to provide a more faithful representation of the costs an investor actually incurs. This model incorporates key features such as minimum commission fees and capital gains taxation. It is particularly relevant when evaluating the strategy's effectiveness from the perspective of a private investor that cannot bear the risk of disregarding a precise transaction cost.

The complete details on how the transaction costs are computed will be explained in Section 3.1 using the Banach Fixed Point theorem to solve the constraint efficiently.

Chapter 3

Dynamic Formulation: Reinforcement Learning

In this chapter, the thesis follows the approach used in Peng et al. 2024 applying reinforcement learning to solve the **dynamic formulation** of the Problem 2.3. Firstly, we analyze the conditions under which it is possible to solve the transaction cost equations; secondly, we focus on using the dynamic programming paradigm to create a suitable environment on which to train the RL agent.

3.1 Transaction costs: Fixed Point Iteration

In this section, we analyze the two transaction cost functions used in this work. After describing both, we focus on how to apply Banach's theorem to solve the equations. The precise description of the algorithm is done in Appendix A.

3.1.1 Linear cost function

The simple linear cost function used is

$$c_{t+\frac{k}{M}} = \alpha + \sum_{i=1}^{N} \beta_i \cdot |x_{i,(t+\frac{k}{M})-} - x_{i,t+\frac{k}{M}}|, \tag{3.1}$$

where α is a fixed cost for every transaction that occurs, and β_i is the proportional cost linked to the number of shares of asset i that are bought or sold during the transaction. For simplicity, we assume $\beta_i = \beta$ for every asset.

It is easy to show that the number of shares can be rewritten sing Equation 2.6

as

$$x_{i,t+\frac{k}{M}} = w_{i,t+\frac{k}{M}} \quad \frac{V_{i,t+\frac{k}{M}}}{p_{i,t+\frac{k}{M}}}, \tag{3.2}$$

$$x_{i,(t+\frac{k}{M})-} = w_{i,t+\frac{k}{M}} \quad \frac{V_{i,(t+\frac{k}{M})-}}{p_{i,t+\frac{k}{M}}}. \tag{3.3}$$

Replacing them into Equation 3.1, we obtain

$$c_{t+\frac{k}{M}} = \alpha + \beta \sum_{i=1}^{N} \frac{1}{p_{i,t+\frac{k}{M}}} \left| V_{i,(t+\frac{k}{M})-} - w_{i,t+\frac{k}{M}} V_{i,t+\frac{k}{M}} \right|. \tag{3.4}$$

At time $t + \frac{k}{M}$, when rebalancing occurs, $V_{i,(t+\frac{k}{M})-}$, $p_{i,t+\frac{k}{M}}$, and $w_{i,t+\frac{k}{M}}$ are all known variables, leaving $V_{i,t+\frac{k}{M}}$ as the only unknown. Therefore, we can rewrite the constraint 2.5 as:

$$V_{(t+\frac{k}{M})-} - c_{t+\frac{k}{M}}(V_{t+\frac{k}{M}}) = V_{t+\frac{k}{M}}. \tag{3.5}$$

As shown in Peng et al. 2024, the previous rebalancing equation, thanks to the contraction property, has a unique solution $V_{t+\frac{k}{M}}$. This allows us to apply the Banach Fixed Point Iteration algorithm to exactly determine the next portfolio value and then calculate the exact number of shares held in the portfolio after rebalancing.

3.1.2 Realistic non-linear cost function

To fully exploit the approach of this thesis, a more sophisticated transaction cost function it is included. The base point of this new cost function formulation is widely adopted in most existing studies such as Canakgoz and Beasley 2009, Strub and Baumann 2018, Benidis et al. 2018, and Peng et al. 2024. The transaction cost at time $t + \frac{k}{M}$ is specified as:

$$c^{\text{base}}_{t+\frac{k}{M}} = \sum_{i=1}^{N} \min \left\{ \max \left\{ \xi_{1i} \left| x_{i,(t+\frac{k}{M})} - x_{i,t+\frac{k}{M}} \right|, \xi_{3i} \right\}, \xi_{2i} \left| V_{i,(t+\frac{k}{M})} - V_{i,t+\frac{k}{M}} \right| \right\}. \tag{3.6}$$

In words, the cost per share of asset i is ξ_{1i}, with a minimum fee of ξ_{3i} and a maximum expense limited to ξ_{2i} of the traded value in that asset. In this work, we set $\xi_{1i} = 0.005$ USD, $\xi_{2i} = 0.5\%$, and $\xi_{3i} = 1.00$ USD. This values are intrinsic of the broker chosen buy the fund manager and we chose to adopt this values as they are used by one of the largest brokerage firms in the US.

To further enhance the accuracy of our representation of transaction costs, we include the FINRA and SEC fees, which apply only to sold stocks during the transaction:

$$\mathrm{SEC}_{t+\frac{k}{M}} = \nu_1 \sum_{i=1}^{N} \max\left\{V_{i,(t+\frac{k}{M})-} - V_{i,t+\frac{k}{M}}, 0\right\}, \tag{3.7}$$

$$\mathrm{FINRA}_{t+\frac{k}{M}} = \nu_2 \sum_{i=1}^{N} \max\left\{x_{i,(t+\frac{k}{M})-} - x_{i,t+\frac{k}{M}}, 0\right\}. \tag{3.8}$$

with $\nu_1 = 2.29 \cdot 10^{-5}$ and $\nu_2 = 1.30 \cdot 10^{-4}$.

However, for a private investor, this is still not enough, since each time the portfolio is rebalanced, the state tax on capital gains must also be computed and taken into account and in this thesis, we include this computation. For reference, we adopt the Italian capital gains taxation, which is fixed at 26%, with the additional feature that capital losses ($CapL$) can be carried forward for four years to offset future capital gains, providing a tax break on significant losses. To compute the value of the capital gain ($CapG$), it is necessary to store information about the price at which the asset was purchased. Being one of the most popular methods used by the Italian banking system, we chose to adopt the average buying price ($AvgP_i$) method instead of the alternative queue-based method. Implementing a FIFO system would have introduced additional computational overhead, unnecessarily slowing down the learning process. The taxation is calculated as follows:

$$CapG_{t+\frac{k}{M}} = 0.26 \cdot \max\left\{\sum_{i=0}^{N} (x_{i,(t+\frac{k}{M})-} - x_{i,t+\frac{k}{M}})^+ \cdot (p_{i,t+\frac{k}{M}} - AvgP_i)^+ - CapL, 0\right\}, \tag{3.9}$$

where $(\cdot)^+$ denotes the positive part, and $CapL$ represents the available capital losses that the portfolio has accumulated. The amount of capital losses used to provide a tax break is then removed from memory.

In the same way we did with the simpler version in Section 3.1.1 we can see that the complete transaction fee formula:

$$c_{t+\frac{k}{M}} = c^{\text{base}}_{t+\frac{k}{M}} + \mathrm{SEC}_{t+\frac{k}{M}} + \mathrm{FINRA}_{t+\frac{k}{M}} + CapG_{t+\frac{k}{M}} \tag{3.10}$$

can all be written as a function of $V_{t+\frac{k}{M}}$ and re-conducted to the 3.5 were it is possible to solve it using the algorithm 1.

The importance of using such a realistic function is that the model can fully understand the implications of having a large number of asset to manage and can consciously chose if the best policy is to reduce to a minimum the number of assets, adopt a full replication strategy or something in between.

Unfortunately, given the limited amount of computational power at our disposal, during testing we are not able to always deploy this version of the transaction cost function in favor of the simpler linear one. Nevertheless the capability of this method to sustain such a complex function should not be underestimated.

3.2 Dynamic Formulation and PPO

In this section we transform the formulation 2.3 to adapt it to the Reinforcement Learning approach using the dynamic programming paradigm. The goal is to rewrite the problem as a series of smaller recursive subproblems that are easier to solve. The DP principle is not a universal one, but it can be applied when the structure of the problem is Markovian: this is exactly the case in our problem, where the manager uses the state variable, representing the system current state, to determine the action to be taken. Since this is a discrete-time infinite-horizon formulation, Bellman's Equation allows us to write

$$\mathcal{V}(s_t) = \max_{a_t \in \mathcal{A}_t} \left\{ -\beta \cdot \mathbf{RTE}[t, t+1)^q + \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot|s_t, a_t)} [\gamma \cdot \mathcal{V}(s_{t+1})] \right\}, \tag{3.11}$$

where $\mathcal{V}(s_t)$ is the value function of the state s_t; $\mathcal{A}_t$ is the set of all possible actions that the agent can take at time t.

The value $\beta \cdot \mathbf{RTE}[t, t+1)^q$ is the reward given to the agent as a consequence of choosing the action a_t, and β is just a rescaling factor: this means that the only transaction cost impact is due to the action taken at time t, not $t+1$, and this is fundamental in creating the correct reward landscape for the agent to learn properly. This factor can be shown as a function of the current state, the current action and the future state as such:

$$r_t = \mathcal{R}(s_t, a_t, s_{t+1}) = -\beta \cdot \mathbf{RTE}^q_{[t,t+1)} \tag{3.12}$$

where the minus sign is necessary to use this metric inside a maximization formulation.

Unfortunately, this problem cannot be solved using conventional techniques in RL such as Q-Learning because of two major difficulties: the index tracking problem has a high-dimensional continuous state space and also a high-dimensional and continuous action space so for this reason it is not feasible to discretize them.

In the RL context, the agent observes at time t the state of the environment s_t and then chooses an action a_t that follows a policy distribution $\pi_\theta(\cdot, s_t)$, where with θ we indicate all the parameters of the said distribution.

The approach that will be used to solve this problem is the same proposed in Peng et al. 2024, which is the **Proximal Policy Optimization** algorithm (PPO)

proposed by Schulman, Wolski, et al. 2017. This method is an extension of the Trust Region Policy Optimization algorithm (TRPO) proposed by Schulman, Levine, et al. 2015. PPO solves a variant of the problem addressed by TRPO by introducing a constraint on the probability ratio between the probability distribution of the old policy and the new one. This means in practice that when the algorithm updates the parameters of the policy θ_{old}, it solves the following optimization problem:

$$\max_{\theta} \mathcal{L}^{\text{CLIP}}(\theta) := \tag{3.13}$$
$$\mathop{\mathbb{E}}_{s\sim\rho_{\theta_{\text{old}}},a\sim\pi_{\theta_{\text{old}}}} \left[\min \left\{ \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_{\theta_{\text{old}}}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon \right) A_{\theta_{\text{old}}}(s,a) \right\} \right]$$

This equation looks complex but it has the simple goal of updating the agent policy parameter θ in a stable way; the overall objective is to find the new parameter θ that maximizes the objective function, which is an approximation of the RL objective function.

$A_{\theta_{\text{old}}}(s,a)$ is the **Advantage Function** that estimates how much better it is to take action a in state s rather than following the average action proposed by π_θ: this is formally expressed by $A_\theta(s,a) = Q_\theta(s,a) - \mathcal{V}_\theta(s)$, where $Q_\theta(s,a)$ is the state-action function that evaluates the state s with action a and after that continues with policy π_θ, while $\mathcal{V}_\theta(s)$ is the value function of policy π_θ. For this reason a positive advantage means that action a is an improvement over what the policy would have chosen.

The ratio between $\pi_\theta(a|s)$ and $\pi_{\theta_{\text{old}}}(a|s)$ measures how likely action a is in the new policy distribution π_θ compared to the old one $\pi_{\theta_{\text{old}}}$; this means that if the ratio is ≥ 1, the new policy is more likely to choose that action over the old one.

The real innovation of the PPO algorithm is the presence of the minimum and the **clip**$(\cdot, min, max)$ function. The clip function clamps all the ratio values between a min and a max: this makes it very difficult to choose a new θ that drastically changes the policy, because the ratio will remain stable for every change over a certain threshold, making the improvement step during the update much more conservative; this happens also in the case of counterproductive actions where the advantage is negative, preventing an over-correction in the opposite direction.

We define ϕ as the set of parameters that define the value function, and its update is managed by the problem:

$$\min_{\phi} \mathcal{V}_{\text{loss}}(\phi) := \mathop{\mathbb{E}}_{s\sim\rho_{\theta_{\text{old}}}} [\mathcal{V}_\phi(s) - \mathcal{V}_{\phi_{\text{old}}}(s)] \tag{3.14}$$

where $\mathcal{V}_{\phi_{\text{old}}}(s)$ is estimated from the sample path of rewards. Being the policy stochastic, it is very important to include in the objective a component related to

the level of exploration in the distribution; for this reason, before the update, the entropy loss is computed as:

$$\mathcal{U}(\theta) = - \mathop{\mathbb{E}}_{s\sim\rho_{\theta_{\text{old}}}} [\text{Entropy}(\pi_\theta(\cdot, s))] \tag{3.15}$$

In conclusion, the PPO algorithm updates the value of the parameters θ, ϕ using multiple steps of stochastic gradient descent of the following problem:

$$\min_{\theta,\phi} \mathcal{L}(\theta, \phi) := -\mathcal{L}^{\text{CLIP}}(\theta) + e_1 \cdot \mathcal{V}_{\text{loss}}(\phi) + e_2 \cdot \mathcal{U}(\theta) \tag{3.16}$$

where e_1 and e_2 are positive coefficients.

3.3 State Description

One of the most important advantages of using such a flexible model-free approach is that we can fully utilize the market information that are much more rich then asset prices and returns alone. For this reason, in this work, in addition to all the prices information at time t for each asset and the value of the index, the agent receives as state variables also the VIX (Ticker symbol for the popular index that measures the expectation of volatility based on S&P 500 index options), T-Bill rates (interest rates of US treasury bills) and the trading volumes for each asset (some data example in Figure 3.1).

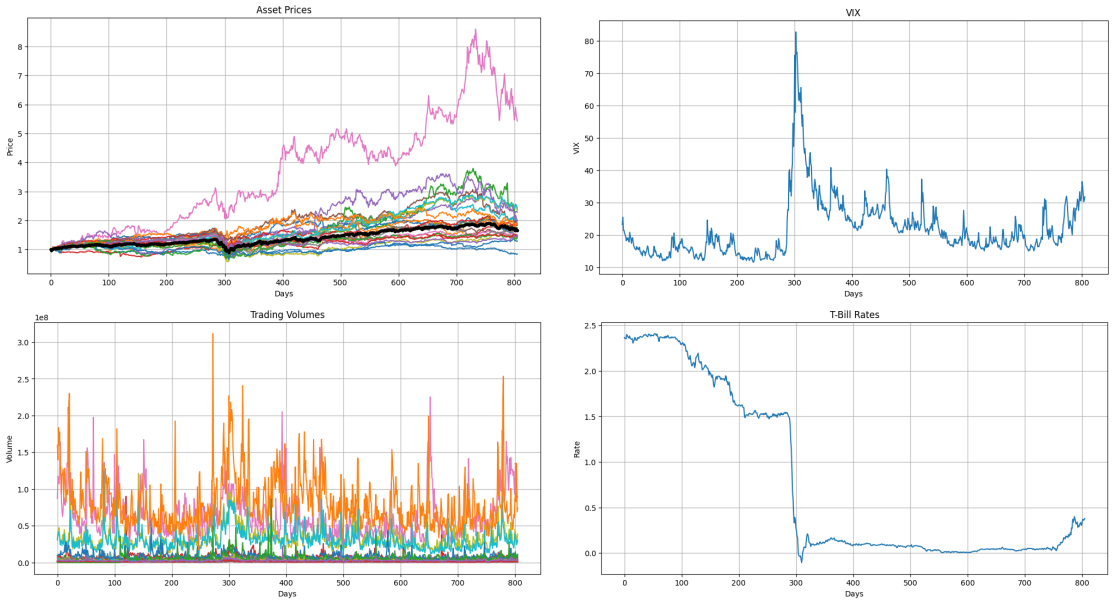

Figure 3.1: State variable information

To enable the model to learn about the cross-sectional correlations, in the state variable all of this data is given in a time window of length n_s days. This means that at time t, since the time period $[t, t+1]$ has M trading days, in the state variable the information from $[t - \frac{n_s}{M}, t]$ is included.

Differently from what was done in Peng et al. 2024, in this thesis the state variable includes the current weights of the portfolio for each asset before the rebalancing happens. Indeed, we want the agent to choose what is the best policy also given the current condition of the portfolio: there is no clear theoretical reason why the best policy would not depend on the current status. For example, if two distinct assets are very similar in their ability to track the index, with the first one being the best and the second still very good but objectively slightly worse, to decide the best course of action the agent should consider whether the second asset is already present in the portfolio before advising to shift everything to the first one. Transaction costs should have a very significant impact on this decision, and not considering the current portfolio allocation could potentially ruin the desired tracking performance.

3.4 Policy and Value function: Network Design

As already discussed, the state space and the action space state are both continuous and high-dimensional, and for this reason are implemented by 2 different Neural Networks. Both network designs are a slight modification of the ones used in Peng et al. 2024 and are discussed in the following section.

3.4.1 Policy Distributions

The first stochastic policy used in this thesis is modeled by a **truncated diagonal Gaussian distribution** (later in Chapter 4 we will discuss some modification that were tried in order to improve the algorithm performance). The truncated diagonal Gaussian distribution is created by firstly combining a diagonal Gaussian distribution with $\mu_\theta(s_t)$ as the mean vector and $\sigma_\theta(s_t)$ as the standard deviation vector; after sampling the vector $z_t \sim Normal(\mu_\theta(s_t), \sigma_\theta(s_t))$, the sample is clipped obtaining:

$$a_t = \text{clip}(z_t, -b, b) \qquad \text{with} \qquad b > 0 \tag{3.17}$$

where the clip function is the same used in Equation 3.13. The clipping is necessary to help the numerical stability of the learning process.

Both functions, $\mu_\theta(s_t)$ and $\sigma_\theta(s_t)$, are N-dimensional and are modeled by two separate Neural Networks (NNs) with the same architecture. The networks receive as input the state vector s_t, which has been flattened to one dimension, and it

passes through a *Batch Normalization layer* that helps the network amplify the differences between different states. Numerous tests without the *Normalization layer* consistently showed that the agent learned a stationary policy, meaning that each input to the network was treated as it was the same exact input. After normalization, the network design consists of a Linear layer, five hidden layers which contain 128 nodes and tanh activation function, and a *Linear output layer* with N nodes for the mean NN, while the standard deviation NN uses an exponential activation function with N nodes. To improve the stability of the efficiency of the algorithm during training, the *Batch Normalization layer* is only active while the agent is learning and locked afterwards.

Still, the RL agent cannot directly use a sample from this distribution to act on the environment. The action a_t has N dimension but its sum is not 1 and it could have negative components so it is impossible to use them as portfolio weights. To transform a_t in w_t we use the softmax function $g : \mathbb{R} \to \mathbb{R}$:

$$w_t = g(a_t) = \left(\frac{e^{a_{1,t}}}{\sum_{j=1}^{N} e^{a_{j,t}}}, ..., \frac{e^{a_{N,t}}}{\sum_{j=1}^{N} e^{a_{j,t}}} \right) \tag{3.18}$$

so that w_t now satisfies constraints 2.4 in the return tracking problem 2.3

3.4.2 Value Function

Similarly to the policy, the value function $\mathcal{V}_\phi(s_t)$ is modeled by a NN which consists in a *Batch Normalization layer, Linear input layer*, four hidden layers with 128 nodes and tanh activation function and a *Linear output layer* with one node.

3.5 Training Scheme

This thesis adopts the training methodology proposed in Peng et al. 2024, which addresses the problem of a limited dataset, that is arguably one of the most challenging issues to overcome in the application of RL to the financial domain. Typically, RL is well suited to settings in which a virtually infinite amount of data or experiments is available for training the agent: common examples include simulated environments and games. In the financial context, however, this assumption does not hold: the task is complicated by the fact that the agent has access to only a single sample path from the prices distribution, and the overall quantity of available data is severely restricted. For this reason, a novel technique is required to enable effective learning.

The core idea underlying this new procedure is to artificially make the dataset bigger by treating each day as a potential starting point. For instance, if the algorithm is trained on weekly time periods, instead of restricting the intervals

to Monday–Friday, a random day of the week is selected as the starting point of the first period in each training episode: in this way, the amount of data available becomes virtually five times greater (even bigger for larger time periods).

Firstly the dataset is spit into two parts: a Training Set and a Test Set. This two sets are kept completely separated so that no data leakage is possible during training.

To train the agent using the PPO algorithm, in each epoch, episodes are collected using the current version of the parameters θ, ϕ to sample trajectories using the policy π_θ. During the episode of training each step is saved inside a buffer:

$$\{(s_t, a_t, s_{t+1}, \pi_{\theta_{\text{old}}}(s_t|a_t), \sigma_{\theta_{\text{old}}}(s_t))|t = t_0, t_0 + 1, ...\} \tag{3.19}$$

After each episode, at every element of the buffer is added the computation of $\hat{A}_t$ and $\hat{\eta}_t$ that are the advantage estimator and the cumulative discounted reward respectively. This values are computed using the procedure described in Peng et al. 2024 that uses the General Advantage Estimator (GAE) (Schulman, Moritz, et al. 2015). To correctly calculate the discounted rewards it is important to note that, being an infinite horizon problem, the value of the terminal state s_{t_n} of the episode (n in the length of the episode) is not fixed as it happens in most RL applications; instead the current approximation of the value function $\hat{\eta}_{t_n} = \mathcal{V}_{\phi_{\text{old}}}(s_{t_n})$ is used.

Chapter 4

Performance Optimization

This thesis proposes also three different refinements to better improve the algorithm efficiency and compare them to the base model most similar to the one used in Peng et al. 2024

4.1 Benchmark Deviation Approach

The idea behind this modification is that, in some way, the portfolio manager always has a rough estimate of what the portfolio weights will be. In this case, it is plausible that the portfolio weights will be similar to those used in computing the S&P 500 index. Instead of relying on the PyTorch implementation of the initialization procedure of the NN, we use the information available at the start of training, namely the market capitalization MarketCap_i of each asset, to determine a starting point $w_i^{\text{benchmark}}$ from which the agent can begin learning:

$$w_i^{\text{benchmark}} = \frac{\text{MarketCap}i}{\sum_{j=1}^{N} \text{MarketCap}_j} \tag{4.1}$$

To apply this baseline to the decision policy, the role of the policy NN is modified. To obtain w_t, the action a_t is sampled as before, and then added to the logit of the benchmark before applying the softmax function. Specifically:

$$w_t = \text{softmax}(\text{logit}(w^{\text{benchmark}}) + a_t). \tag{4.2}$$

During testing, this method will be evaluated to determine whether it offers a tangible advantage in terms of performance and efficiency. Furthermore this is not the only possible benchmark; this approach easily adapts for any preferred method the manager wants to implement depending on the index that the portfolio aims to track.

4.2 Tanh-squashed Gaussian

The truncated Gaussian approach is very simple to implement but it comes with its complications. To correctly compute the value of the objective function 3.13 we need to be able to calculate exactly the value of $\pi_\theta(a_t, s_t)$. It is not possible to directly use the probability density of the Gaussian distribution $N(\mu_\theta(s_t), \sigma_\theta(s_t))$ because the clipping function concentrates all the probability density of the tails at the boundaries $-b, b$. For this reason, each time a new action is sampled, the probability calculation needs to be adjusted, thereby making the computation more expensive. Moreover, this makes the probability distribution not continuous and, at the boundaries, can cause problem with the gradient computation during the update phase.

For this reason, we tested a new approach. Instead of clipping z_t, we use the tanh function to restrict its value to the interval between -1 and 1. Formally this translates to:

$$a_t^{\tanh} = b \cdot \tanh(z_t) \qquad \text{with} \qquad z_t \sim N(\mu_\theta(s_t), \sigma_\theta(s_t)), \tag{4.3}$$

where in this case b is a rescaling factor to amplify the network decision. The main advantage of this approach is that the $\tanh(\cdot)$ function is bijective between $\mathbb{R} \to (-1, 1)$, differentiable, and its Jacobian never vanishes inside (−1,1): for this reason tanh satisfies the strong conditions of the **Change of Variables Theorem** and the transformed random variable distribution remains absolutely continuous.

Using this transformation instead of the truncated diagonal Gaussian distribution, offers two main advantages: the computation needed to correct the probability density calculation is much faster to compute and, being continuous and differentiable, the gradients are much more likely to be stable and not collapse to zero.

4.3 PCA for covariance directions

The other issue we propose to tackle is a limit of the diagonal Gaussian distribution: every component is completely independent of the others. The main reason for using this simplification is that, in this way, the agent only needs to learn to predict N standard deviations, one for each asset. Making the NN learn the complete covariance matrix would dramatically increase the number of parameters to predict and would also make the whole network design larger: for this reason, this is not a feasible approach.

Even though this approach is not possible, it is simple to imagine many situations in which making the agent explore the action space where all the assets are treated as independent is not optimal for training performance, wasting time experimenting

with assets that have historically very similar price movements. For this reason, in this thesis we test a new procedure:

1. the sample variance-covariance matrix $\hat{\Sigma}$ is precomputed before the training using all the historical returns of each asset available in the training data.

2. the eigenvector decomposition is computed to find all the covariance directions, as in PCA:
$$\hat{\Sigma} = Q\hat{\Lambda}Q^T \tag{4.4}$$
where Q is a $N \times N$ orthogonal matrix that contains all the principal component directions that we will use to construct the covariance matrix $\Sigma_\theta(s_t)$ for the Gaussian distribution. The diagonal matrix $\hat{\Lambda}$ contains all the eigenvalues that represent how much variance is explained by the corresponding direction.

3. the standard deviation FNN is repurposed from predicting the value of $\sigma_\theta(s_t)$ to predict the best N eigenvalues to use, $\text{diag}(\Lambda)_\theta(s_t)$.

4. when the action needs to be sampled from the policy distribution, we now use the Gaussian distribution as follows:
$$a_t^{\tanh} = b \cdot \tanh(z_t) \qquad \text{with} \qquad z_t \sim N(\mu_\theta(s_t), \Sigma_\theta(s_t)), \tag{4.5}$$
where $\Sigma_\theta(s_t) = Q\Lambda_\theta(s_t)Q^T$.

This new method allows the agent to explore the action space in a more sophisticated way, fully exploiting the historical knowledge of the correlation between assets without making the task more computationally difficult: in fact in both methods the NN has to predict N parameters for the distribution shape.

Using this approach is only possible if, after sampling from the distribution, the tanh-squashed Gaussian transformation is used (Section 4.2) instead of the truncated distribution discussed in Section 3.4.1: this is because it is not possible to analytically compute the necessary probability correction for the distribution if the starting Gaussian is not diagonal. If instead we use the tanh transformation, the probability correction computation done by the Jacobian remains the same.

Chapter 5

Results

In this chapter we analyze the algorithm performance in two different sections: firstly, to verify if the RL agent is capable of adapting and learn in simple scenarios, we test the algorithm with a synthetic dataset where it's possible to understand if the learned policy is effectively the optimal one. Secondly, once the performance has been validated, the RL agent is trained on a real market dataset.

The in-sample and out-of-sample performance are presented using a figure each containing four graphs. All of them record the agent decision and the resulting outcomes after the training. For the Normalized Value, Rolling Return Tracking Error and the Transaction Costs graphs the gray lines show the agent's progression during training. To correctly read the Normalized Value graph (i.e., the graph in the top left) it is important to remember that, being the performance metric the Return Tracking Error (RTE), for the tracking to be good, the most important thing that the portfolio has to replicate is the shape of the index trend and not how closely the portfolio value and the index value stay together.

During testing, every setting is evaluated using four different the versions of the algorithm:

- **V1**: is the base version of the algorithm (i.e., the most similar to the one proposed by Peng et al. 2024), the action is sampled using the truncated diagonal Gaussian described in Section 3.4.1;
- **V2**: this version implements the benchmark explained in Section 4.1 as the only modification to **V1**.
- **V3**: in this version, the policy distribution is changed as explained in Section 4.2 using a tanh-squashed Gaussian distribution;
- **V4**: extension of **V3** that implements the PCA direction as described in Section 4.3.

During the training phase, the agent relies on the stochastic policy to sufficiently explore the action space and avoid getting stuck in local minima. This exploration is crucial for learning; however, when evaluating the policy, exploration is not required and may even lead to better or worse performance purely by chance. For this reason, to compute the RTE we employ the deterministic version of the policy. This is done by taking the distribution's mean as the sample and then using it as before. For example, for version **V1** this means that the action taken at time t is

$$w_t = \text{softmax}\left(\text{clip}(\mu_\theta(s_t), -b, b)\right), \tag{5.1}$$

where the output of the policy NN is $\mu_\theta(s_t)$,that is the mean function used for the Gaussian distribution of Equation 3.17. All the experiments presented in this chapter were executed on a personal laptop (with 24 GB of RAM, 10 CPU cores and running macOs 15). For this reason, it was not possible to run more computationally demanding simulations.

5.1 Synthetic Dataset

In this section we discuss the algorithm results trained using a synthetically produced dataset. This type of testing environment is fundamental when discussing such complex learning algorithms: before deploying this method to real-world data, is critical to validate it under controlled conditions. Using an artificial dataset is a very easy way to verify if, for example, the RL agent is able to learn a policy that we can determine beforehand.

5.1.1 Data Generation

The data used to validate the algorithm has been procedurally generated. Each asset price $p_{i,t}$ is derived from the following stochastic process

$$p_{i,t+\frac{k+1}{M}} = p_{i,t+\frac{k}{M}} \cdot r_{i,t+\frac{k+1}{M}}, \tag{5.2}$$

where $r_{i,t+\frac{k+1}{M}}$ is the daily return of the asset calculated as

$$r_{i,t+\frac{k+1}{M}} = \exp\left((\mu + \epsilon_i) \cdot \delta t + \sigma \cdot \sqrt{\delta t} \cdot z_{i,t+\frac{k+1}{M}}\right), \tag{5.3}$$

where:

- ϵ_i is an intrinsic deviation of asset i form the general market percentage drift μ;
- $\delta t = \frac{1}{252}$ is the time step (252 are the usual trading days in a year);

- σ is the market volatility;
- $z_{i,t+\frac{k+1}{M}}$ is a random variable that comes form a standard normal distribution.

This model is the classic Geometric Brownian Motion stochastic process.

One important characteristic of this choice is that there is no correlation between asset prices and returns, and the each asset has constant drift during the entirety of the dataset. Critically, the intrinsic drift is the only information that the agent can infer from the synthetic data. All the other information used in the state variable described in Section 3.3 is randomly generated without any financial model and is not correlated with the asset prices. This is done to keep things simple and observe if and how well the RL agent is able to understand that much of the information is not relevant for the solution.

To create a dataset in which the best policy is known, the index values are created using the already generated asset prices. A vector of composing weights w_{Index} is chosen and then the index value is calculated as

$$I_{i,t+\frac{k}{M}} = w_{Index}^T \cdot p_{t+\frac{k}{M}}. \tag{5.4}$$

In this way, using a very simple set of weights, we expect that the agent policy will coincide as much as possible with w_{Index} after training.

5.1.2 Testing

To test the algorithm, using the same synthetic dataset of 2520 simulated days, three different testing settings T1, T2, and T3 are built using different types of environment's parameters that are explicated in the following table:

Parameter	**T1**	**T2**	**T3**
N	10	10	50
w_{Index}	[1,0,0, ...]	$[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, ...]$	$[1, 0, 0, ...]$
M	5	5	21
n_s	15	21	126

Table 5.1: Synthetic testing experiments

These experiments aim to test the algorithm in three different aspects. **T1** mainly tests the agent's ability to isolate the first asset from the other one since the index is entirely composed of it. The agent operates in periods of 5 days, corresponding to a trading week, using three weeks of previous data as the state variable. **T2** setting tests if the agent is able to choose multiple assets in its assessment of the policy. Differently than before, the state variable includes more

information using a month of data: this tests how well the agents can recognize a pattern in price movements. The **T3** setting is a stress-test of the various algorithm when the dimensionality of the problem is increased: both the number of assets and the look-back windows are significantly larger.

These three settings are tested using all the different versions of the proposed methods, to validate the effectiveness of each one. To see if the benchmark has any effect in version **V2**, the benchmark weights are set equal to the wanted policy (this is obviously "cheating" but can be useful to verify that the modification has the intended effect), so it is important to keep this in mind while reading **V2** results.

Params	Value	Description
V_{0-}	$1.0e10$	Initial portfolio value
$w_{i,0-}$	$\frac{1}{N}$	Starting portfolio weights
q	2	Power in Equation 2.2
`P-lr`	$1.0e{-}4$	Learning Rate of the policy NN
`V-lr`	$1.0e{-}4$	Learning Rate of the value function NN
`H-dim`	64	Dimension of each hidden layer in every NN
b	5	Parameter of Equations 3.17 and 4.3
ϵ	0.2	Parameter of Equation 3.13
β	500	Parameter of Equation 3.12
e_1	0.5	Coefficient of the value loss in Equation 3.16
e_2	0.05	Coefficient of the entropy loss in Equation 3.16
H	320	Number of training epochs
$\hat{n}$	32	Mini-batch size in RL training
l_e	50	Episode maximum length
n_e	4	Number of episodes per epoch
$\hat{k}$	4	Number of CPU cores for parallel computing

Table 5.2: RL parameters for synthetic dataset

Each one of the versions is tested using the same training parameters listed in Table 5.2, so that every comparison is as fair as possible. The used learning rates are set voluntarily very high in order to speed up the validation phase, given the fact that the synthetic data is much more simpler to process in comparison with Real Market data.

The results shown in Table 5.3 are produced, by testing the performance of each agent after training, by sampling 10 successive periods t of M trading days from the training dataset for the in-sample RTE and from the testing dataset for the out-of-sample RTE.

The proposed approach behaves as intended and capable of achieving a very

Scenario	In-Sample RTE			
	V1	**V2**	**V3**	**V4**
T1	$4.3010e{-8}$	$4.3010e{-8}$	$4.3010e{-8}$	$4.3010e{-8}$
T2	$3.2496e{-6}$	$3.2458e{-6}$	$3.2496e{-6}$	$3.2599e{-6}$
T3	$6.8409e{-4}$	$4.1103e{-8}$	$8.7087e{-3}$	$8.6914e{-3}$
	Out-of-Sample RTE			
	V1	**V2**	**V3**	**V4**
T1	$3.6232e{-8}$	$3.6232e{-8}$	$3.6232e{-8}$	$3.6232e{-8}$
T2	$6.3221e{-5}$	$6.3189e{-5}$	$6.3221e{-5}$	$6.3248e{-5}$
T3	$2.9649e{-5}$	$3.9835e{-8}$	$9.0628e{-3}$	$8.1822e{-3}$

Table 5.3: Synthetic experiments training results

good performance in the majority of the conducted tests. For **T1** and **T2** every version of the algorithms that we tested took approximately 3 minutes to complete the training and testing; **T3** was a lot more computationally heavy and took an average of 7 minutes to complete, with version **V1** being the quickest at 6 minutes and 28 seconds.

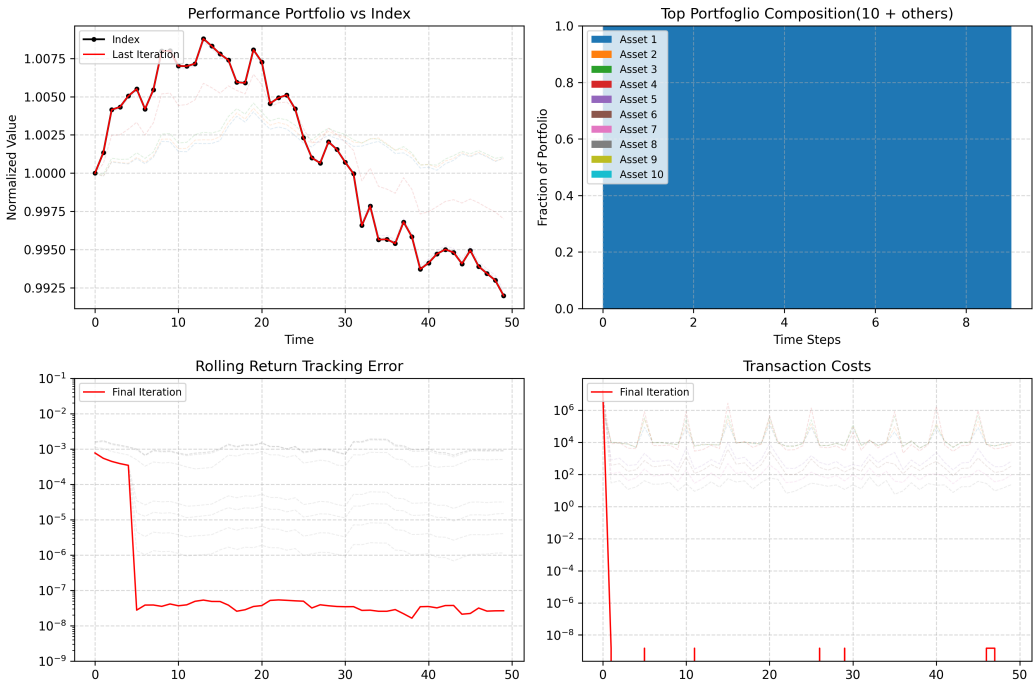

Figure 5.1: Out-of-Sample performance in **T1** using **V1**

As expected, **V2** holds a little advantage over the other versions, having the

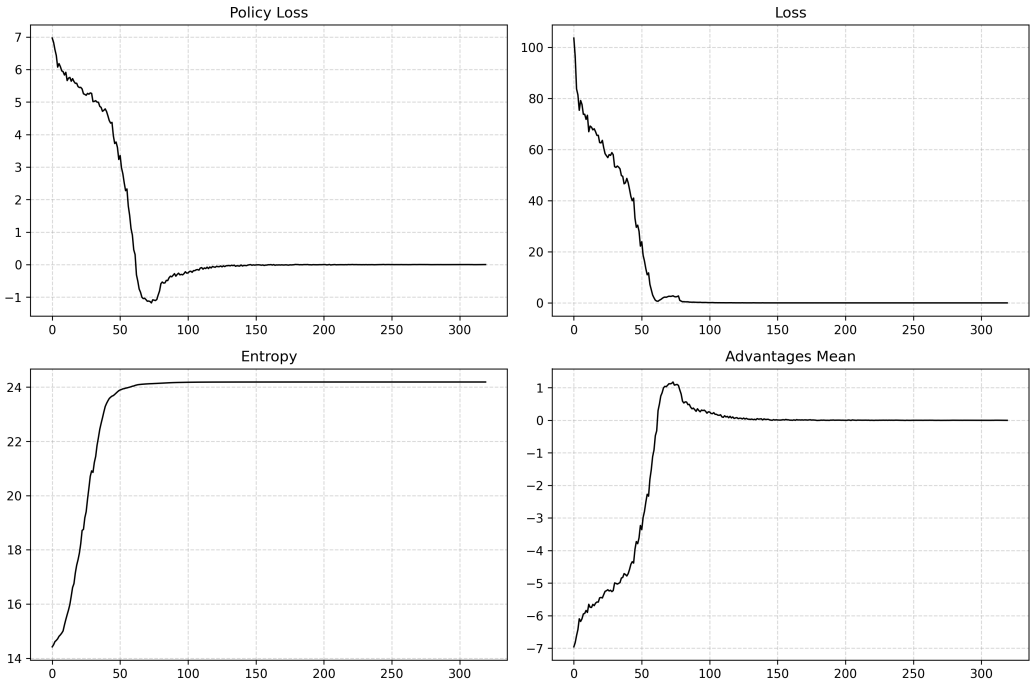

Figure 5.2: Training performance in **T1** using **V1**

benchmark weights set equal to the optimal policy: this means that the proposed modification works as intended, speeding up the agent's learning.

Specifically, in **T1** all the versions behaved in the exact same way, meaning that, in such a simple setting, all the approaches can find the optimal policy. Figure 5.1 shows the out-of-sample performance of the **V1** version: the four graphs are very clear in showing that the agent correctly found that the optimal policy is investing the totality of the portfolio in Asset 1, which is the only one composing the artificial Index. The rolling RTE graph shows us that the resulting portfolio follows extremely closely the behavior of the index, except at time $t = 0$, where there is a tracking problem but, in reality, the spike in the error is perfectly reasonable: in fact, the agent starts with a uniformly distributed portfolio, leading to high transaction costs in the first rebalancing to deploy the optimal policy.

In Figure 5.2 we can observe that the algorithm converges to the solution without any difficulties, steadily lowering the policy loss and with it the total loss.

In the **T3** setting, versions **V3** and **V4**, both employing the tanh-squashed Gaussian, did not achieve performance on par with the other versions. This is because the optimal policy is highly concentrated and, together with the large number of assets, the tanh transformation of the Gaussian makes it harder for the agent to drive the policy toward selecting a single asset over the others within

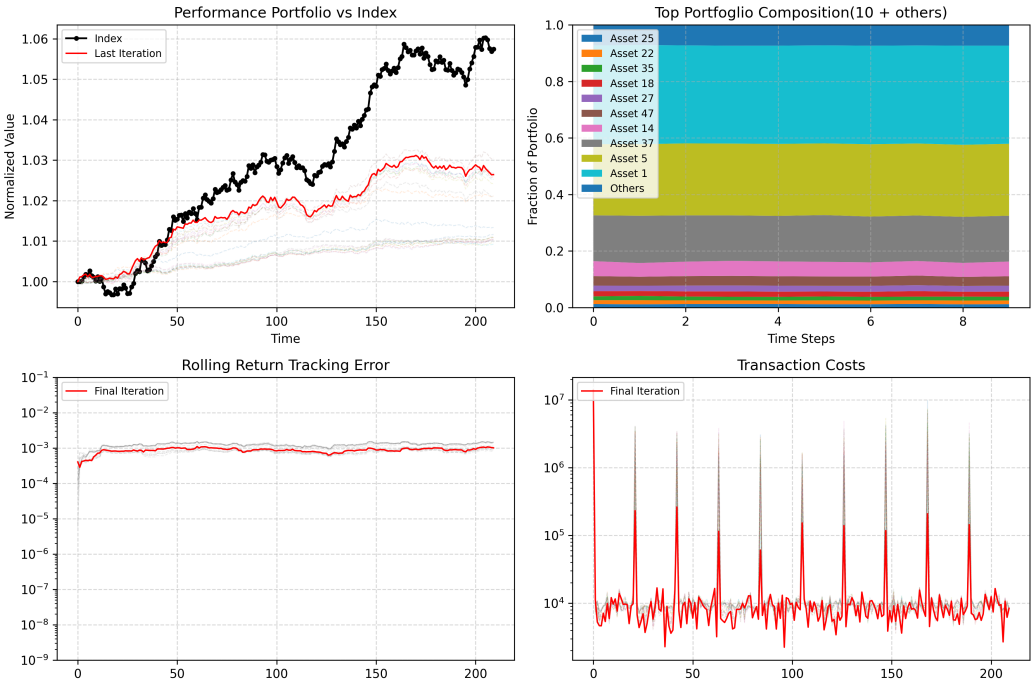

Figure 5.3: Out-of-Sample performance in **T3** using **V3**

the limited number of training epochs. In Figure 5.3 we can see that the agent is learning that Asset 1 is the most important, but it is not yet able to completely invest in it. Figure 5.4 shows that something similar happens with version **V1**: with the clip function, the Policy NN does not have to push the components' means as far apart to create a dramatic policy, so it converges more quickly, but we can see that the policy is not yet complete because at the start of 3 periods it does not choose the optimal policy. In this case, we only tested **R1** and **R2** using all the algorithm's versions, and **R3** was only tested using version **V3** because it performed better in the similar setting.

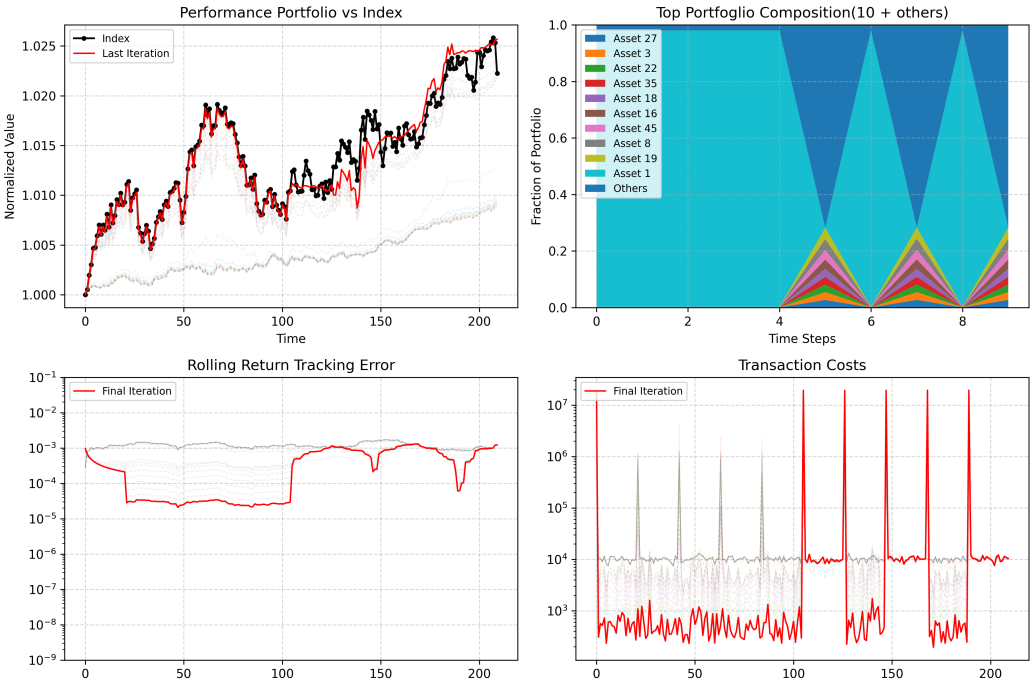

Figure 5.4: In-Sample performance in **T3** using **V1**

5.2 Real Data

In this section, the discussed approaches are tested on real financial datasets retrieved using the Python API of Yahoo Finance. The dataset is composed of all the prices and trading volumes of the 500 assets composing the S&P 500 Index, the Index itself, the VIX Index, and the T-Bill rates. As before, we created two different testing settings that are described in Table 5.4. Unfortunately, given the computational power at our disposal, it was not possible to test the full-scale problem using all the 500 assets in the S&P 500 Index.

All of these settings test the algorithm using a period of a month and only two months of history as the state variable. This is done for two main reasons: the first is that using a smaller state variable considerably reduces the NN size, thus reducing computational time. Secondly, this tests how flexible the algorithm can really be. The real difference between them is the time partitioning. This choice was made is such a way because: **R1** now in his training portion does not include the COVID-19 crisis, which only appears in the out-of-sample testing segment; conversely, in **R2** and **R3** the shock is included in the training portion, whereas during the out of sample portion the market is mostly stable. The **R3**, given the dimensionality increase compared to the other two, will only be tested using the

Parameter	R1	R2	R3
N	50	50	200
Start Training	2015-01-01	2020-01-01	2020-01-01
Start Testing	2019-10-20	2023-08-30	2023-08-30
End Testing	2020-12-31	2024-07-30	2024-07-30
M	20	20	20
n_s	40	60	60

Table 5.4: Real Market testing experiments

best-performing version.

For settings **R1** and **R2** we used the training parameters in Table 5.5.

Params	Value	Description
V_{0-}	$2.0e10$	Initial portfolio value
$w_{i,0-}$	$\frac{1}{N}$	Starting portfolio weights
q	2	Power in Equation 2.2
P-lr	$2.5e{-}5$	Learning Rate of the policy NN
V-lr	$2.5e{-}5$	Learning Rate of the value function NN
H-dim	128	Dimension of each hidden layer in every NN
b	5	Parameter of Equations 3.17 and 4.3
ϵ	0.2	Parameter of Equation 3.13
β	1000	Parameter of Equation 3.12
e_1	0.5	Coefficient of the value loss in Equation 3.16
e_2	0.05	Coefficient of the entropy loss in Equation 3.16
H	2000	Number of training epochs
$\hat{n}$	64	Mini-batch size in RL training
l_e	12	Episode maximum length
n_e	50	Number of episodes per epoch
$\hat{k}$	8	Number of CPU cores for parallel computing

Table 5.5: RL parameters for **R1** and **R2**

The learning rates are much smaller than in the synthetic case to better help with convergence in these more complex conditions. Also, the NNs are bigger with 128 nodes in each hidden layer, making them suited to this new data. Another important change is the number of episodes in a mini-batch, which is double the size of the synthetic case: this is because the real data is noisier and the agent needs more data points so that the GAE estimators have less bias. Moreover,

to further increase the number of sample paths evaluated, using all the available computational power at our disposal, the number of CPU cores is increased to 8.

The experimental results are displayed in Table 5.6. Each of the experiments shown took an average of 55 minutes to complete (laptop performance was remarkably weakened due to thermal throttling).

	In-Sample RTE			
Scenario	**V1**	**V2**	**V3**	**V4**
R1	$2.7957e{-3}$	$2.1987e{-3}$	$2.1706e{-3}$	$4.1336e{-3}$
R2	$3.1472{-3}$	$3.3063e{-3}$	$2.4299e{-3}$	$7.1215e{-3}$
	Out-of-Sample RTE			
	V1	**V2**	**V3**	**V4**
R1	$2.5224e{-3}$	$3.0037e{-3}$	$1.9920e{-3}$	$4.2534e{-3}$
R2	$1.9688e{-3}$	$2.5443e{-3}$	$1.6125e{-3}$	$1.5884e{-3}$

Table 5.6: Real Dataset training results

As with the artificial dataset, the results are calculated as the RTE over 4 successive periods of length $M = 20$ days: this equates to successive 80 trading

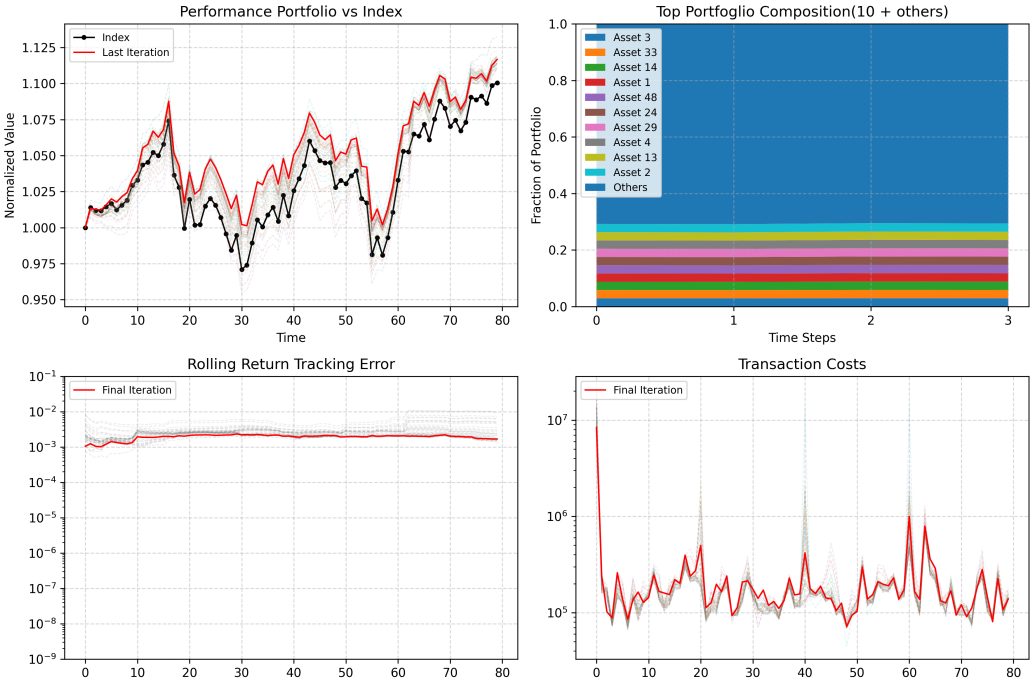

Figure 5.5: Out-of-Sample performance in **R1** using **V3**

days that are sampled from the training set for the in-sample analysis, and from the test set for the out-of-sample analysis.

For setting **R1** the proposed **V3** version outperforms all the other ones with slightly better performances across the in-sample and out-of-sample RTEs. All of the results are in line with the performance observed in the large scale experiments conducted in Peng et al. 2024 though conducted with the full set of assets, with a larger NN design and more training time.

In Figures 5.6 and 5.5 we can observe how the agent chooses an almost perfect stationary policy even if the chosen portfolio weights are different: this is a direct effect of including transaction costs in the model. It is possible to observe the agent reducing transaction costs while learning, as shown by the gray spikes at the start of each period of 20 days.

In contrast, Figures 5.7 and 5.8 show that these versions were not able to stabilize in time, meaning that the transaction cost spikes at the start of the period take a noticeable toll on the tracking error performance (especially visible for version **V4**).

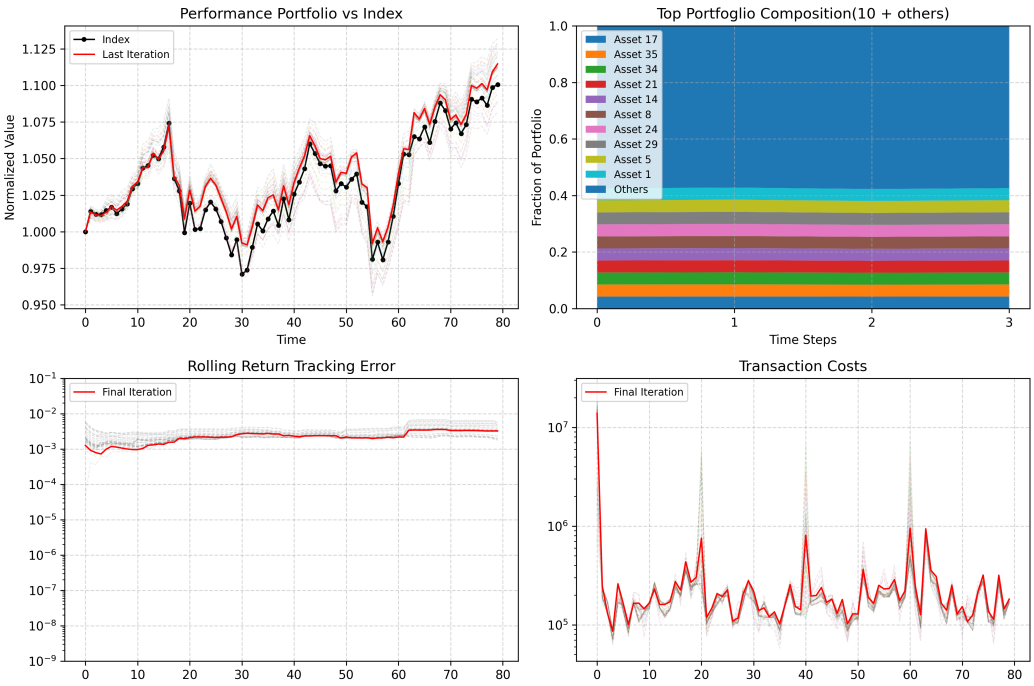

Figure 5.6: Out-of-Sample performance in **R1** using **V1**

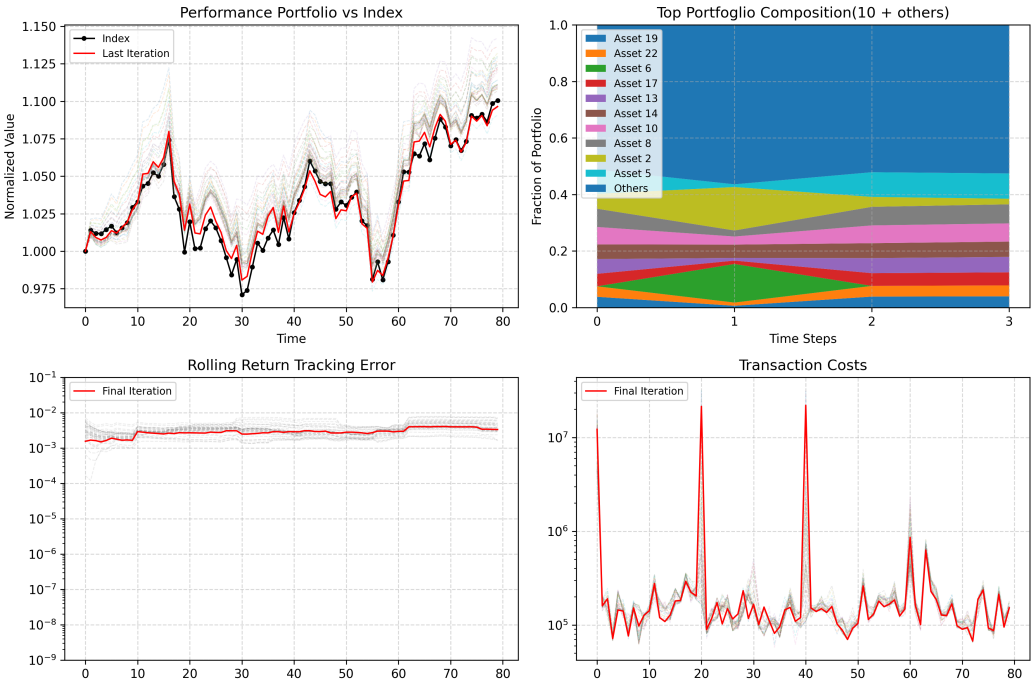

Figure 5.7: Out-of-Sample performance in **R1** using **V2**

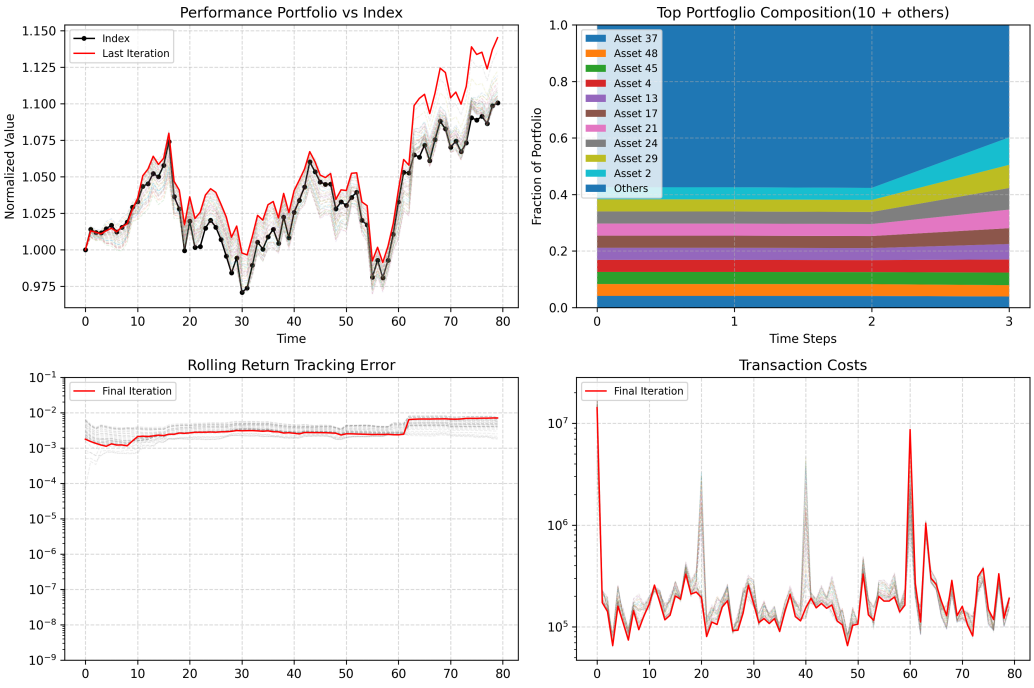

Figure 5.8: Out-of-Sample performance in **R1** using **V4**

Concerning the training process, all the versions behave practically in the same way. In Figures 5.9 and 5.10, we show the learning progression, and it is clear that the learning rate to allow for better convergence should still be lowered, ensuring that both the actor and the critic do not experience a dramatic "jumps" in their estimations. Nonetheless this choice of learning rates was necessary to keep the computational time contained to one hour while still showing some sign of convergence.

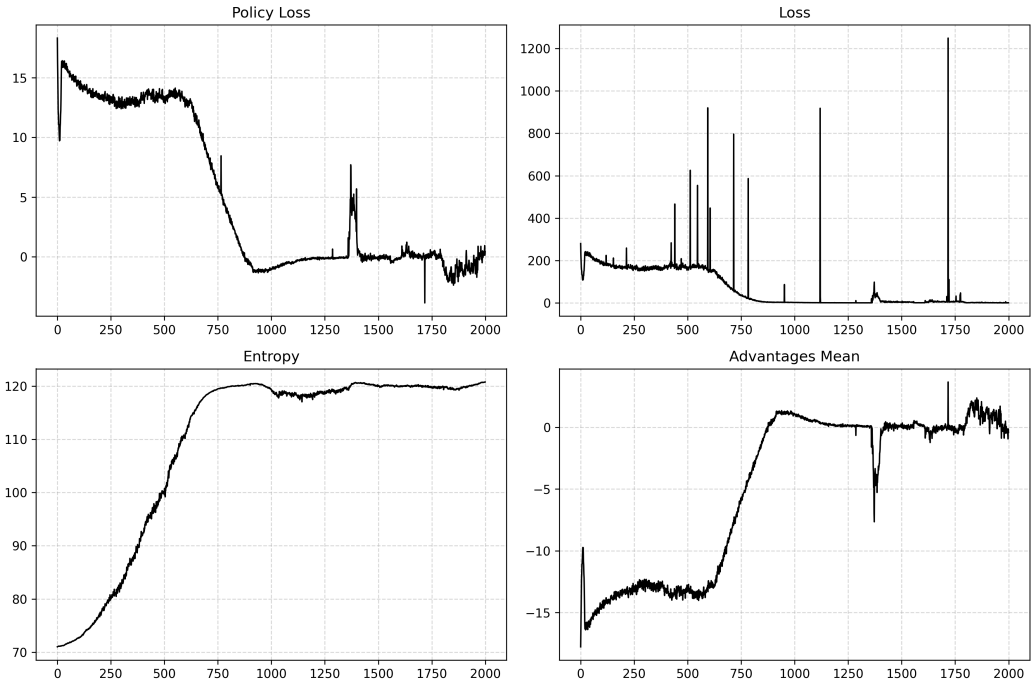

Figure 5.9: Training performance in **R1** using **V1**

A similar story happens in setting **R2** where version **V3** outperforms all the other versions by a similar margin, as we can see in Figure 5.11. **V1** is still second best considering both in and out of sample RTE (Figure 5.12) but, this time, **V4**, even if the in-sample performance is lacking compared with the others, it improves the out-of-sample performance of even over the **V3** version, as can be seen in Figure 5.14. What probably caused a worse in-sample performance is the presence of the economic shock in the training part of the dataset: this may indicate that the way the PCA directions are implemented in the algorithm leaves it less capable of abruptly adaption to market changes (in-sample performance of **V3** can be seen in Figure 5.14) and makes the convergence to the optimal policy a lot slower than the other methods.

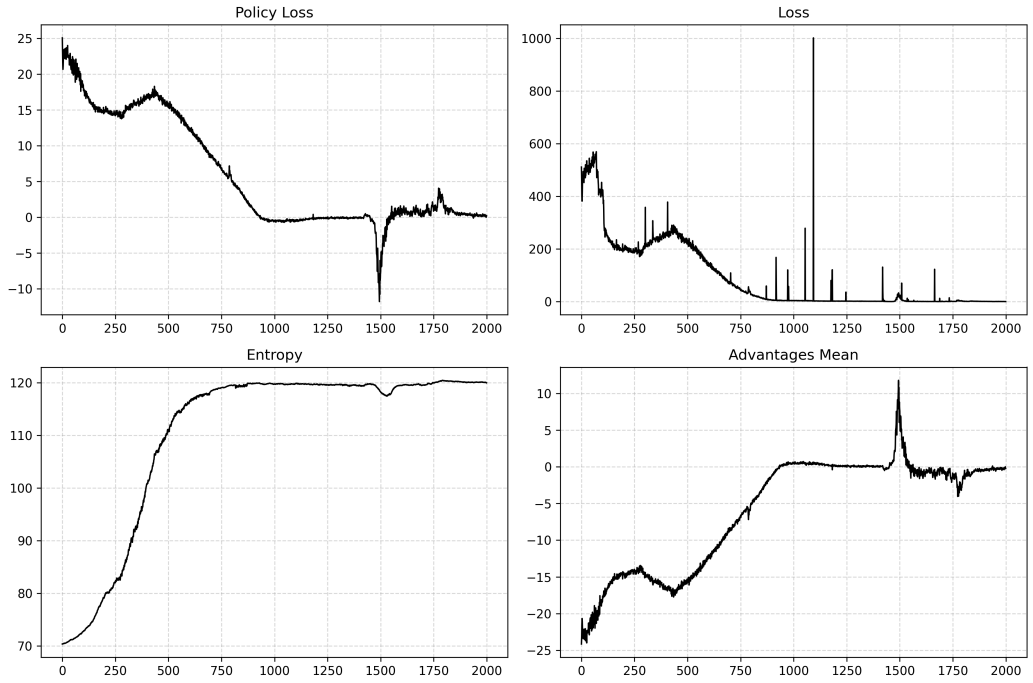

Figure 5.10: Training performance in **R1** using **V2**

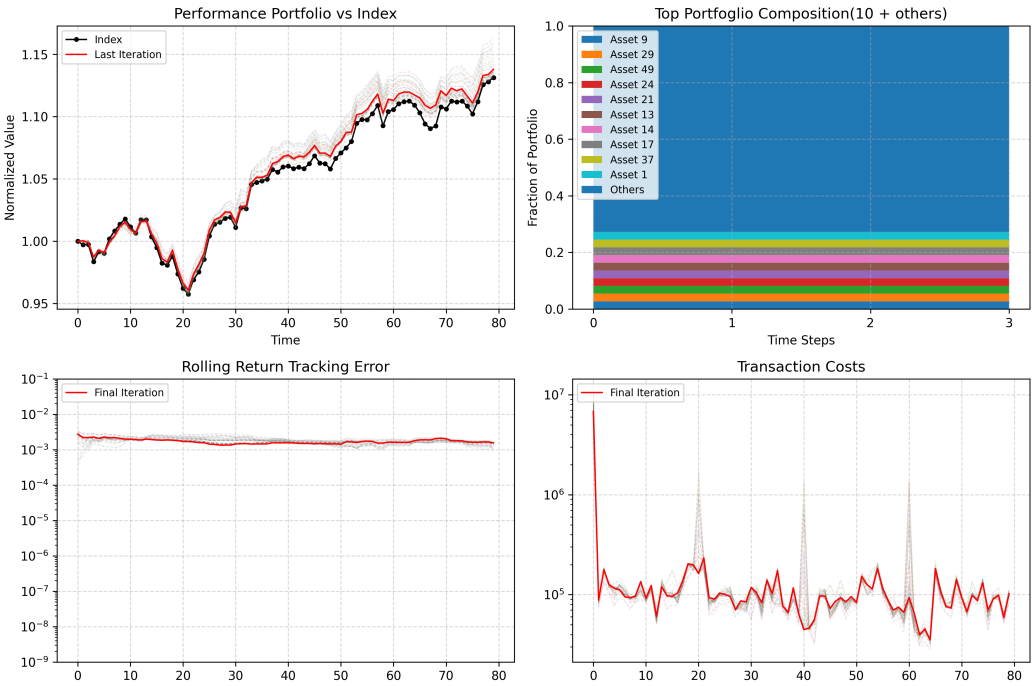

Figure 5.11: Out-of-Sample performance in **R2** using **V3**

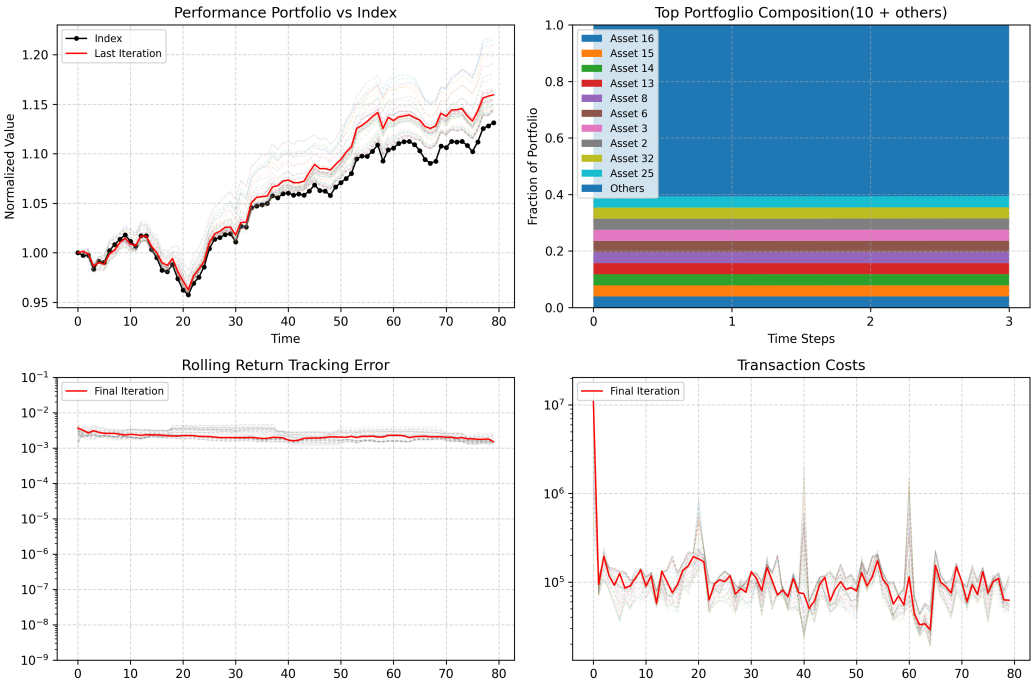

Figure 5.12: Out-of-Sample performance in **R2** using **V1**

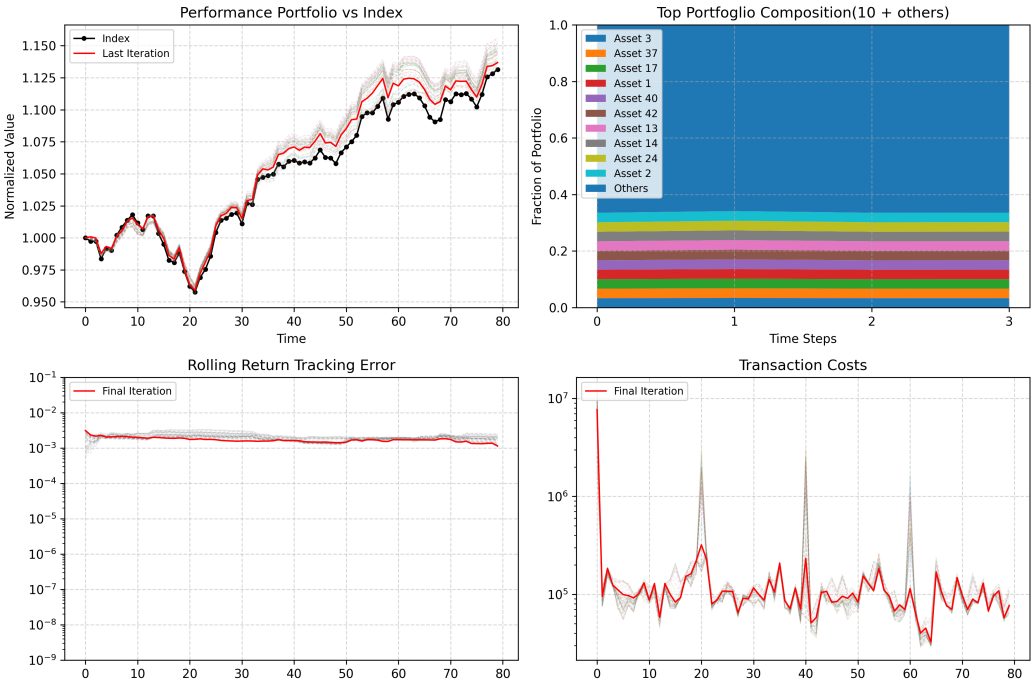

Figure 5.13: Out-of-Sample performance in **R2** using **V4**

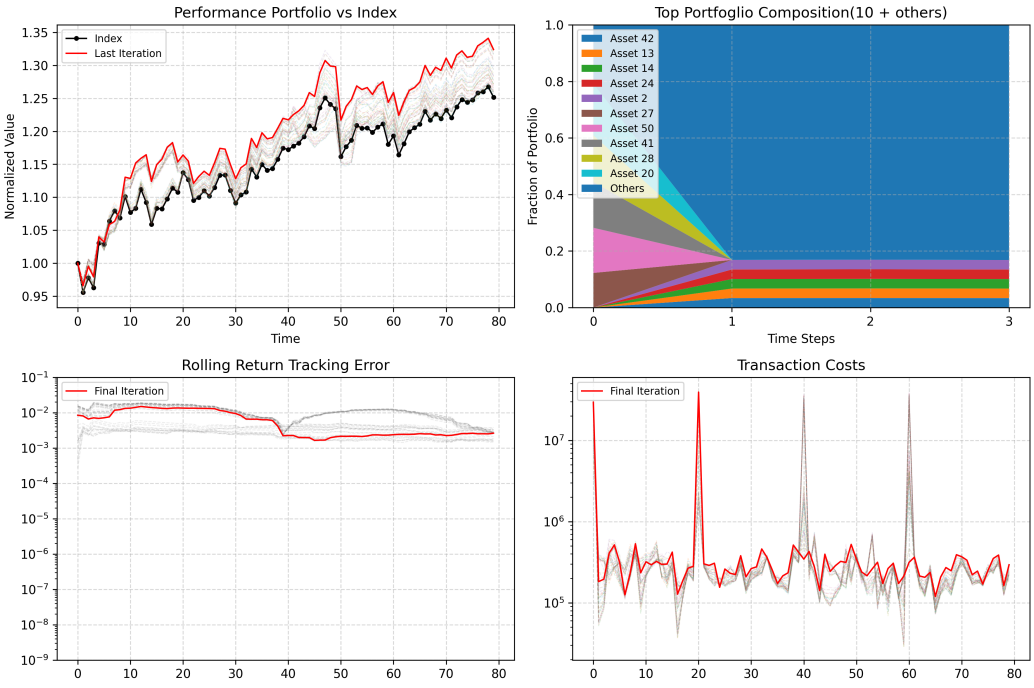

Figure 5.14: In-Sample performance in **R2** using **V4**

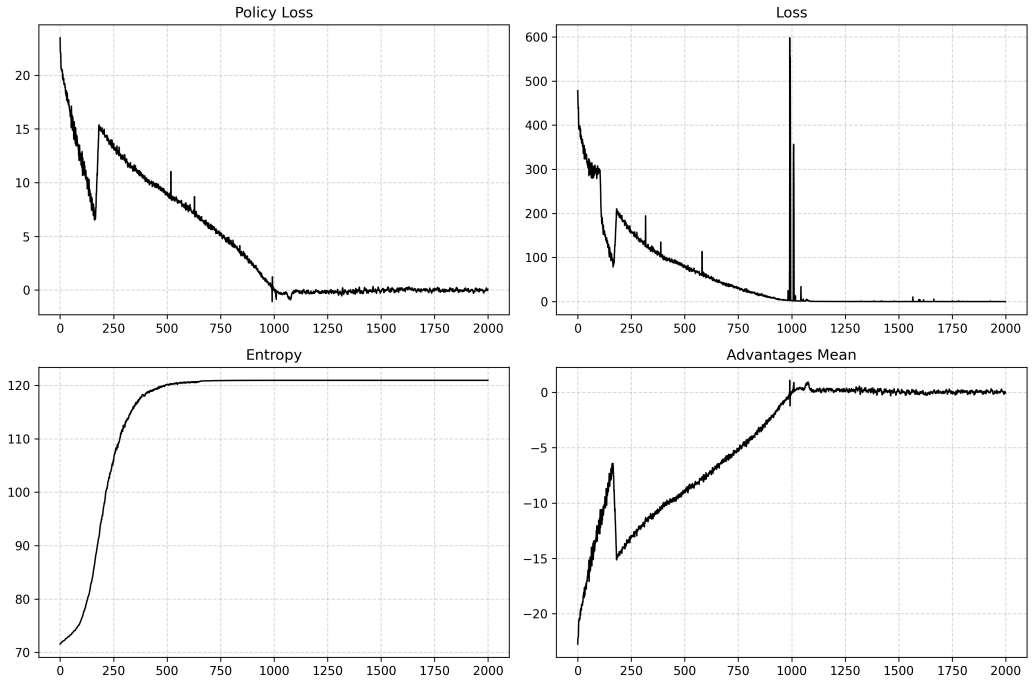

Figure 5.15: Training performance in **R2** using **V3**

Once again the training performance of all the versions is very similar, and, for the sake of brevity, we show in Figure 5.15 the performance of **V3**.

The **R3** setting it is tested only using version **V3** because it showed the most promising and robust results, and the agent was trained using the parameters displayed in Table 5.7. Most of the parameters remain the same used in the previous settings but some modifications were made to obtain results at an earlier training epoch. In this way we managed to get results in under 90 minutes of computational time.

Params	Value	Description
V_{0-}	$2.0e10$	Initial portfolio value
$w_{i,0-}$	$\frac{1}{N}$	Starting portfolio weights
q	2	Power in Equation 2.2
`P-lr`	$5e{-}5$	Learning Rate of the policy NN
`V-lr`	$5e{-}5$	Learning Rate of the value function NN
`H-dim`	128	Dimension of each hidden layer in every NN
b	5	Parameter of Equations 3.17 and 4.3
ϵ	0.2	Parameter of Equation 3.13
β	1000	Parameter of Equation 3.12
e_1	0.5	Coefficient of the value loss in Equation 3.16
e_2	0.05	Coefficient of the entropy loss in Equation 3.16
H	750	Number of training epochs
$\hat{n}$	64	Mini-batch size in RL training
l_e	12	Episode maximum length
n_e	50	Number of episodes per epoch
$\hat{k}$	6	Number of CPU cores for parallel computing

Table 5.7: RL parameters for **R3**

	V3	
Scenario	In-Sample RTE	Out-of-Sample RTE
R3	$2.0892{-}3$	$1.4310e{-}3$

Table 5.8: R3 training results

The agent performances, after training, which are calculated in the exact same way as before, are shown in Table 5.8 and we can clearly see that it is able to improve on the best result obtained in setting **R2**. The two results are directly comparable because they are tested on the same period length and on the same dataset.

In Figures 5.16 and 5.17 it is possible to observe how well the agent's policy is at replicating the shape of the index, achieving the best RTE. Unfortunately the graphs illustrating the portfolio composition become much more difficult to read due to the amount of assets, but as before we can see that it adopts an almost stationary approach to reduce transaction costs. Figure 5.18 describes the learning process as very stable, meaning that with more assets to choose from, the action policy can transition more smoothly between each update, improving the convergence.

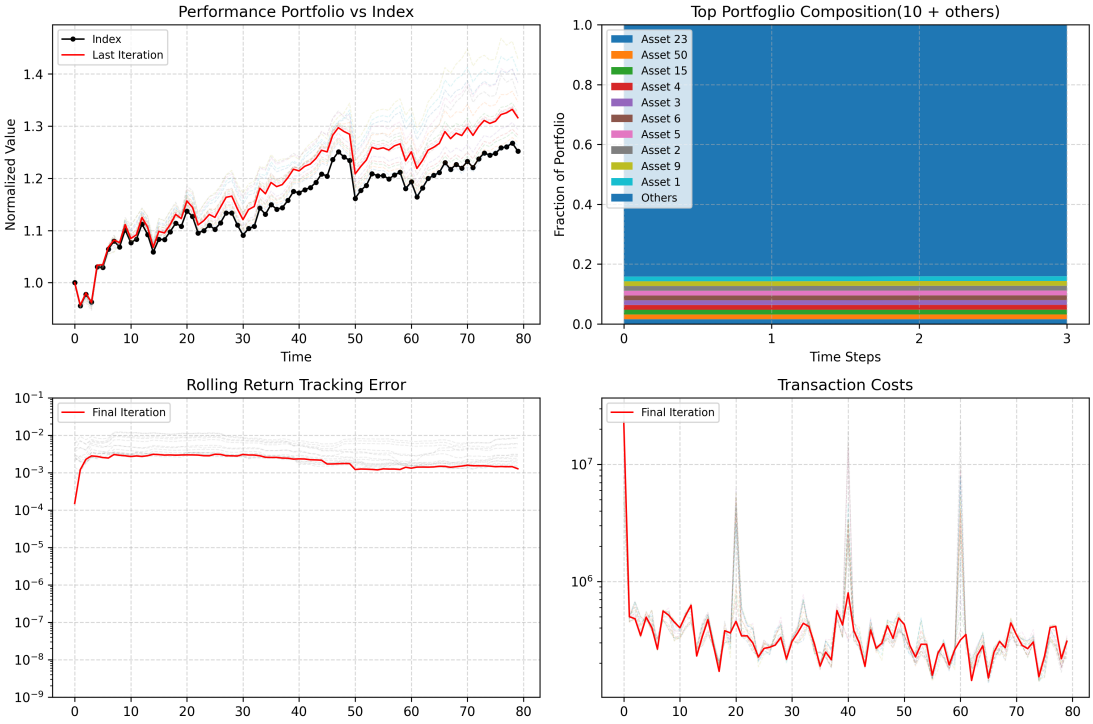

Figure 5.16: In-Sample performance in **R3** using **V3**

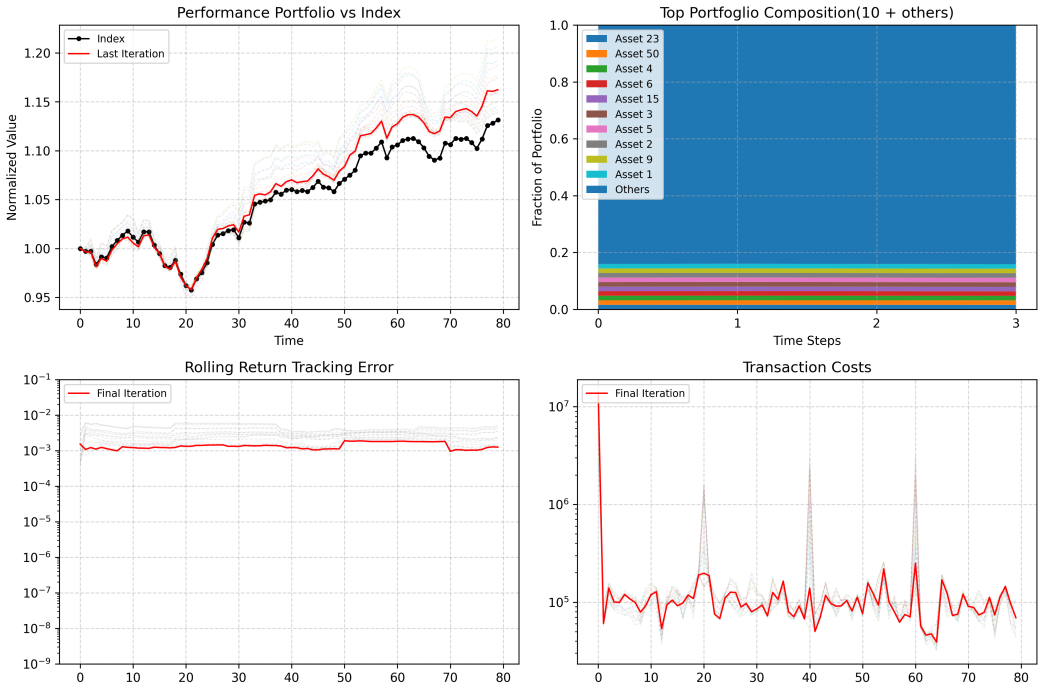

Figure 5.17: Out-of-Sample performance in **R3** using **V3**

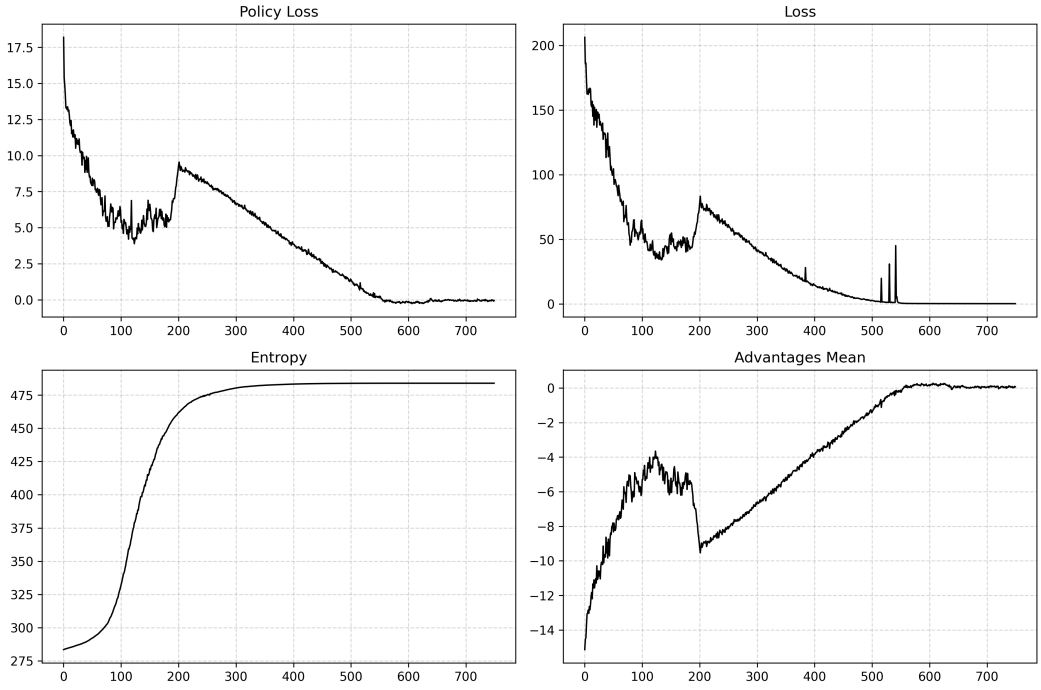

Figure 5.18: Training performance in **R3** using **V3**

Chapter 6

Conclusions

In this thesis we explored the Financial Index Problem through the paradigm of dynamic programming, exploiting the structure to implement the Reinforcement Learning method to solve the problem. This innovative framework has enabled us to tackle the problem in a new way, enabling the use of very realistic transaction cost function that are not limited to theoretical results but can accurately predict the policy behavior in the real world.

Thanks to the comprehensive testing and validation on the synthetic dataset and on the real-market dataset, we established how all the proposed versions of the algorithm work as intended in both controlled conditions, even reaching satisfactory performance in the presence of economic shocks, such as during the COVID-19 pandemic.

The benchmark method has shown to be very capable of reducing the amount of computational time needed to reach convergence on the optimal policy, especially in a high-dimensionality setting. This is subject to the quality of the benchmark, but it could still be a useful resource in the hands of the fund manager to introduce prior knowledge in the model.

The tanh-squashed Gaussian model proved to be the best performing among the tested versions, outperforming the base version (the most similar to the one proposed in Peng et al. 2024). This proves the importance of having a proper action engineering combined with smooth gradients in an high-dimensional setting.

Even if it is an extension of the tanh-squashed Gaussian model, the covariance directions version proved to be more challenging to work with in the limited amount of computational time at our disposal. Nonetheless the results are still very promising.

Overall, this thesis demonstrates that reinforcement learning provides a promising and already viable framework for dynamic financial index tracking.

Appendix A

Fixed Point Iteration algorithm

After collapsing the chosen cost function into a form only dependent by $V_{t+\frac{k}{M}}$ the algorithm used to solve Equation 3.5 is the Banach fixed point iteration algorithm that is described below:

Algorithm 1 Banach Fixed Point algorithm used to solve Equation 3.5

$\epsilon \leftarrow 10^{-15}$ ▷ Setting the tolerance to stop the iterations
$V^{\text{old}}_{t+\frac{k}{M}} \leftarrow V_{(t+\frac{k}{M})-}$ ▷ Initialize to the portfolio value to improve convergence
$V^{\text{new}}_{t+\frac{k}{M}} \leftarrow V_{(t+\frac{k}{M})-} - c_{t+\frac{k}{M}}(V^{\text{old}}_{t+\frac{k}{M}})$ ▷ Initialize the iteration
while $\left|V^{\text{old}}_{t+\frac{k}{M}} - V^{\text{new}}_{t+\frac{k}{M}}\right| \geq \epsilon$ **do**
    $V^{\text{old}}_{t+\frac{k}{M}} \leftarrow V^{\text{new}}_{t+\frac{k}{M}}$ ▷ Update the estimate
    $V^{\text{new}}_{t+\frac{k}{M}} \leftarrow V_{(t+\frac{k}{M})-} - c_{t+\frac{k}{M}}(V^{\text{old}}_{t+\frac{k}{M}})$
end while
return $V^{\text{new}}_{t+\frac{k}{M}}$

Appendix B

Python code structure

In this chapter, the structure of the python code used in this thesis is explained. This explanation is mean to be a broad overview of the code logic and underlying structure so that reading the python code would be more understandable.

B.1 main.py and main_synthetic.py

This are the python file to execute experiments from using the real market data or the artificial dataset respectively. This file imports the configuration file `config.json` or `config_synthetic.json`. This fils have the following structure:

Listing B.1: `config_synthetic.json` for **T3 V3**

```
{
"genConfig" :
{
"NUM_DAYS": 2520, "NUM_ASSETS": 50
},
"scenConfig":
{
"index_weights": [[0, 1]], "test_perc" : 0.20
},
"indexConfig" :
{
"M":21, "window":126, "rebalance_period":1, "initial_portfolio_value"
    :2e10, "obs_state_portfolio" : true, "transaction_cost" : "simple"
    , "eps1" : 2.5e-3, "eps2" : 1.0, "nu1" : 2.29e-5, "nu2" : 1.6e-4,
    "tax" : 26.0e-2
},
"agentConfig" :
{
```

```
"policy_lr":1e-4, "value_lr":1e-4, "eps_clip":0.2, "
    truncated_gaussian":true, "limit_func" : "tanh", "B": 5, "
    log_prob_correction" : true, "advantages_normalization":false, "
    benchmark":false, "benchmark_type":"marketcap","var-cov":false,"
    device":"cpu", "hidden_dim":64
},
"trainConfig" :
{
"DEBUG" : false, "num_epochs": 320, "sample_learnig" : 16, "
    batch_size":32,"episode_len": 50, "num_episodes": 12, "
    entropy_coeff": 0.1, "value_coeff": 0.5,"beta" : 5e2, "gamma" :
    0.99, "lam" : 0.95, "num_cores" : 4, "mode" : "parallel_spawn"
},
"evalConfig" :
{
"episode_len_eval" : 10
}
}
```

In this way, every parameter of each experiment can be changed from this file without the need to edit the code.

After loading the configuration, the main loads the chosen dataset and feeds the data to two objects of the `IndexTrackingEnv` class: the first instance is used during training and the second one is used in testing to prevent any data leakage. Successively the `PPOAgent` and the `Trainer` object are created using the user settings.

After selecting the appropriate training function, the agent is trained ad all the training logs are saved. Training logs also include the agent parameters at different moments during the training phase, so that it is possible to show the agent leaning progression in the performance graphs. After testing, all the figures are saved ad a PDF report is saved with all of the relevant information of the experiment.

B.2 IndexTrackingEnv class

This class is the responsible for simulating the market environment and the tracking portfolio. The RL agent only interacts with it through the `step()` method and the `__get__state()` method that will be used by the `Trainer` class. The `IndexTrackingEnv` object stores all the relevant market information for the simulation to behaves like the market keeping track of all the agent decisions and its implication.

A `TrackingPortfolio` object is created during initialization. This object is used by the `IndexTrackingEnv` to manage the tracking portfolio after an action as been taken by the agent. The `step()` makes the `TrackingPortfolio` interact with the `action` taken by the `PPOAgent` by computing transaction cost using the

`fixed_point_banach()` function and then, it simulates each M days in the period. The function returns the reward to the agent ad all the relevant logging information contained in a dictionary.

B.3 Trainer class

This is the class responsible for training the `PPOAgent` and making it interact with the selected `IndexTrackingEnv`. It has been implemented to support multiprocessing in order to take full advantage of a modern CPU. For this reason the `train()` and `collect_rollouts()` functions have also the multiprocessing equivalent with the same underlying functioning.

The `train()` function is the main training loop in which the agent is trained. At the start of each training epoch a new `RolloutBuffer` object is created: this object will collect all the relevant information for each step in the episode using the class method `collect_rollouts`(). At the end of each episode, the `RolloutBuffer` method `compute_returns_and_advantages`() computes the correct advantage estimators for each new entry on the buffer and adds it to the collected information. The training cycle ends with the call of the `update`() method of the class `PPOAgent` that returns the update logs. At a specified frequency, during the training loop, the program saves the state of each parameter inside the value NN and policy NN so that, at the end of training, the agent performance can be tested at various stages of the learning process.

B.4 PPOAgent and EvaluationAgent classes

This classes contain all the information structure of the RL agent, including the NNs for the policy and value functions. They are inherit from the same `BaseAgent` class that define the base structure. The two classes are used to keep separate the training from evaluation process in order to avoid data leaks and improve performance during the evaluation phase or the collection of episodes. The policy network is a `GaussianPolicy` object while the value function is a `ValueNetwork` object; both class inherit from the `nn.Module` of the `torch` python library. This classes contain the NN design as described in section 3.4.1 and all the methods necessaries to sample from the distribution an admissible action on the portfolio or calculate the probability density. The most important method of the `PPOAgent` class in the `update()` one: this method, after randomly shuffling the entries of the buffer, it updates, using a single step of the stochastic gradient descent, the NNs' parameters using batches of a chosen dimension.

Bibliography

Benidis, K., Y. Feng, and D. P. Palomar (2018). «Optimization methods for financial index tracking: From theory to practice». In: *Foundations and Trends in Optimization* 3.3, pp. 171–279 (cit. on p. 19).

Boyde, Emma (Apr. 2021). «ETFs: why tracking difference usually matters more than tracking error». In: *Financial Times*. URL: https://www.ft.com/content/44b08a6c-55d7-4841-8e8e-aad6451a4cc3 (cit. on p. 15).

Brandimarte, Paolo (2021). *From Shortest Paths to Reinforcement Learning: A MATLAB-Based Introduction to Dynamic Programming*. Springer.

— (2022). *Ottimizzazione per la ricerca operativa*. Torino: CLUT. ISBN: 978-88-7992-482-5.

Canakgoz, N. A. and J. E. Beasley (2009). «Mixed-integer programming approaches for index tracking and enhanced indexation». In: *European Journal of Operational Research* 196.1, pp. 384–399 (cit. on p. 19).

Chiam, S. C., K. C. Tan, and A. Al Mamun (2013). «Dynamic index tracking via multi-objective evolutionary algorithm». In: *Applied Soft Computing* 13.7, pp. 3392–3408 (cit. on p. 15).

Hambly, Ben, Renyuan Xu, and Huining Yang (2023). «Recent advances in reinforcement learning in finance». In: *Mathematical Finance* 33.2, pp. 437–503. DOI: 10.1111/mafi.12382.

Peng, Xianhua, Chenyin Gong, and Xue Dong He (2024). *Reinforcement Learning for Financial Index Tracking*. arXiv:2308.02820v2 [q-fin.PM]. arXiv: 2308.02820 [q-fin.PM] (cit. on pp. 12–14, 18, 19, 21, 24–27, 30, 40, 49).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). «Trust Region Policy Optimization». In: *International Conference on Machine Learning*. PMLR, pp. 1889–1897 (cit. on p. 22).

Schulman, John, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel (2015). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv: 1506.02438. URL: https://arxiv.org/abs/1506.02438 (cit. on p. 26).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347. URL: https://arxiv.org/abs/1707.06347 (cit. on p. 22).

Staden, Pieter M. van, Peter A. Forsyth, and Yuying Li (2018). «Beating a Benchmark: Dynamic Programming May Not Be the Right Numerical Approach». In: *SIAM Journal on Scientific Computing* 40.3, A1353–A1381. DOI: 10.1137/17M1131151.

Strub, O. and P. Baumann (2018). «Optimal construction and rebalancing of index-tracking portfolios». In: *European Journal of Operational Research* 264.1, pp. 370–387 (cit. on p. 19).

Yao, David D., Shuzhong Zhang, and Xun Yu Zhou (2006). «Tracking a Financial Benchmark Using a Few Assets». In: *Operations Research* 54.2, pp. 232–246 (cit. on p. 12).