ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Master Thesis Econometrics

December 19, 2022

Deep Reinforcement Learning for Goal-Based Wealth Management: a Model-Based Approach

Authors:

Afrasiab Kadhum (581164ak)

Supervisors:

Dr. Andrea Naghi - ESE

David Kroon - Ortec Finance

Prof.dr Marc Francke - Ortec Finance

Second assessor:

Dr. Eoghan O'Neill - ESE

Abstract

This research investigates the effectiveness of model-based Reinforcement Learning (RL) in solving a goal-based wealth management problem. This model-based approach provides the decision-maker, namely the agent, with one-step-ahead forecasts of asset returns, for which we incorporate various linear autoregressive models and compare their performance to that of a complex non-linear model used in prior literature. Furthermore, we train and test all agents using a set of simulated economic scenarios. We find that the model-based method consistently achieves a higher goal success rate than the model-free RL baseline in both a sparse and a dense reward environment. Moreover, we achieve similar performance gains over the model-free method with both the linear and nonlinear regressions, indicating that this model-based approach does not necessarily require estimating complex prediction models.

Contents

1 Introduction

Reinforcement learning (RL) is a branch of machine learning that has received a growing amount of attention in the past decade. It consists of teaching an intelligent agent how to strategically act within a specific environment by rewarding it for desirable behaviour, according to a user-specified reward function. As the agent seeks to maximize its total received reward, it learns how to optimally choose its actions through trial and error. Although this idea is relatively simple, RL has shown to be an effective approach for learning strategies in complex sequential decision-making problems. Recent years have seen the technique successfully be applied to solve a variety of tasks such as playing games (Berner et al., 2019; Ecoffet et al., 2019; Silver et al., 2018), robotics control (Buckman et al., 2018; Lillicrap et al., 2015; Vecerik et al., 2017) and other control tasks (Qi et al., 2019). Furthermore, RL methods have also gained the interest of researchers in the field of finance (Fischer, 2018). Particularly the task of dynamic asset allocation or portfolio optimization is well suited for RL, due to it being a sequential decision-making problem. However, research in this domain has been limited compared to the size of the literature dedicated to the study of other applications of RL (Fischer, 2018).

A relatively new branch of portfolio optimization that has received interest from investors lately is Goal-Based Wealth Management (GBWM) (Blanchett, 2015). GBWM is an asset allocation strategy that aims to directly maximize the probability of achieving certain goals of an investor, such as accumulating a set amount of wealth at the end of an investment period. Due to the presence of a clear goal, RL is an attractive candidate to solve GBWM problems. Recently, this application of RL has been studied and shown to be a promising replacement for the classic dynamic programming approach (Dasa and Varmaa, 2020; Dom et al., 2022). However, similar to the majority of RL literature on portfolio optimization, these studies only use so-called "model-free" RL methods, where no model of the environment is learned or used (Fischer, 2018). Although model-free methods are simpler to implement, they have been shown to not always reach the same level of performance achievable by model-based methods (Moerland et al., 2020).

In this research, we investigate the efficacy of a model-based RL method for solving GBWM problems and compare its performance to that of model-free RL. Our model-free baseline consists of the Proximal Policy Optimization (PPO) algorithm of Schulman et al. (2017), which is commonly used in practice due to its stable and efficient learning properties (Achiam, 2018). We apply the model-based framework introduced by Yu et al. (2019) to PPO by including prediction models that

provide the agent with forecasts of future asset returns. Furthermore, Yu et al. (2019) limit their analysis to a comparison between model-free and model-based methods, and only include complex nonlinear prediction models in their methodology. To gain insight into how the choice of the prediction model affects the performance of the model-based agent, we implement different prediction models, using both linear and nonlinear models, and compare their respective performances. For the linear models, we vary the complexity of the models from separate AutoRegressive (AR) models for each asset to a Vector AutoRegressive (VAR) model, as we forecast the future returns using only the past returns. Additionally, we include the nonlinear model used by Yu et al. (2019), namely the Nonlinear Gaussian Dynamic Boltzmann Machine (N-G-DyBM). This model can be interpreted as a VAR model that also includes exponential moving averages of dependent variables as regressors, and captures nonlinearity within the time series through a Recurrent Neural Network (RNN) layer (Dasgupta and Osogami, 2017).

Moreover, we examine the effectiveness of including behaviour cloning in both our model-based and model-free methods for GBWM, where the agent also adjusts its policy based on expert demonstration. Nair et al. (2018) show that behaviour cloning can positively affect the learning process of agents in environments where rewards are sparse, which tends to be the case in GBWM. Finally, we use both a sparse, goal-only reward function and a dense, shaped reward function in order to study the effect of the sparsity of reward signals on the performance of both the model-free and model-based methods. We train and test all our agents using a dataset consisting of 15,000 simulated economic scenarios, each representing a possible investment period.

Similarly to Yu et al. (2019), we find that the proposed model-based method consistently outperforms the model-free method, achieving a higher goal success rate in the GBWM problem setting. This result is consistent in both the sparse and the dense reward environment and also when we enforce additional constraints on the portfolio weights. Additionally, we find some evidence that the model-based method requires fewer training iterations to converge than the model-free method, especially when rewards become more sparse. This is in line with the general consensus that model-based methods tend to learn more efficiently than model-free methods (Moerland et al., 2020).

Furthermore, we find that the predictions of the simple linear models and the nonlinear model lead to similar performance gains when provided to the agent. This includes forecasts of models that are fitted using only historical data instead of the simulated scenario data, indicating that this model-based approach can already be effective when simple models are used.

Finally, we do not observe an increase in performance when behaviour cloning is included in the

learning process of the agent. Contrary to Yu et al. (2019), who find that behaviour cloning leads to a decrease in risk, we observe a substantial increase in risk levels of the learned strategies. We hypothesize that this decrease in performance stems from the fact that the expert actions, which here are the actions that maximize returns at each timestep, are not necessarily optimal in the setting of GBWM and thus provide the agent with conflicting learning signals.

Our work adds to the growing literature on RL in portfolio optimization. This literature largely studies the use of model-free RL algorithms due to the difficulty of modeling financial markets (Jiang et al., 2017; Liang et al., 2018). Nevertheless, Yu et al. (2019) show that their model-based RL framework achieves higher risk-adjusted returns than its model-free counterpart, demonstrating the potential of model-based RL in portfolio optimization. We build further on this result, where our main contributions can be summarised as follows. First, to our best knowledge, no previous research investigates the application of model-based RL in GBWM problems. This setting calls for a significantly different reward function than what is used in common risk-return optimization problems. The specification of the reward function is one of the most crucial parts in RL and has a substantial impact on the results (Sutton and Barto, 2018). Our work thus provides insight into how well this model-based approach translates to this new context. Second, we extend the analysis of the model-based framework of Yu et al. (2019) by comparing the use of different prediction models to see how this choice affects the final performance of the agent. Finally, we investigate the effect of adding a form of behaviour cloning in our RL algorithms, which is yet to be done in the current literature on RL in GBWM.

2 Problem setup

In this section, we describe the notation of a general GBWM problem and detail the specific problem setup that we consider in this research.

Let the wealth of an investor at time t be defined as W_t. This investor starts with an initial capital W_0 and their goal is to accumulate a final inflation-adjusted wealth of W_T^* after an investment period of T years. We denote the wealth at the end of this period as the terminal wealth W_T. The investor allocates their wealth among m different assets, where we denote the fraction of wealth that the investor has assigned to asset i at time t as the portfolio weight $a_{i,t}$. These weights can be changed on a yearly basis during the investment period while facing fixed transaction costs c. The GBWM objective can be described as

$$\begin{aligned} &\max_{a_t, t \leq T} \quad \mathbb{P}\left[W_T \geq W_T^*\right], \\ &\text{s.t} \qquad\quad \sum_{i=1}^{m} a_{i,t} = 1, \quad \forall t \leq T, \end{aligned} \tag{2.1}$$

where $\mathbb{P}\left[W_T \geq W_T^*\right]$ denotes the probability of achieving the terminal wealth goal of the investor.

The problem setting in this research is as follows. We assume an investor with a starting capital of \$100,000 and a goal terminal wealth of \$400,000, adjusted for inflation. The investment period spans 40 years and the following seven asset classes are available: equity (emerging markets), equity (world), fixed income (high yield), fixed income (investment grade), cash (USD), real estate (indirect) and hedge funds. All asset weights are bounded between 0 and 1, such that no leverage or short selling is allowed. Furthermore, we allow additional constraints to be applied to the asset weights, which are shown in Table 1. Finally, transaction costs of 0.5% for both buying and selling are used throughout all experiments.

Table 1: Asset category constraints

	Equity (combined)	Fixed income (combined)	Cash	Real estate	Hedge funds
lower bound	0.2	0.2	0.0	0.0	0.0
upper bound	0.8	0.8	0.3	0.2	0.1

Note: "combined" refers to the sum of the weights for all assets in a specific category, e.g equity - world and equity - emerging markets

3 Reinforcement learning background

This section aims to provide some background information on what RL exactly conveys and to introduce some terminology, where we follow the mathematical notation of Sutton and Barto (2018).

The goal of RL is to get an intelligent agent to learn how to solve a task. This agent lives and acts within a dynamic environment. At a certain point in time t, it observes this environment as a state s_t and interacts with it by performing actions a_t, causing the environment to transition to the next state s_{t+1}. The rule that the agent uses to decide which action to take is termed the policy π, which can be either deterministic or stochastic. Furthermore, at each timestep the agent receives a reward R_t according to a reward function $r(s_{t-1}, a_{t-1}, s_t)$ that must be specified by the researcher. This feedback loop is visualised in Figure 1. Essentially, the agent learns through trial and error an optimal policy π^* such that it maximizes its expected long-term cumulative reward.

The following sections describe the mathematical foundation behind a set of algorithms, namely

policy gradient methods, that aim to find π^*. Furthermore, we give a brief discussion on different classes of algorithms and on existing model-based RL methods in the literature.

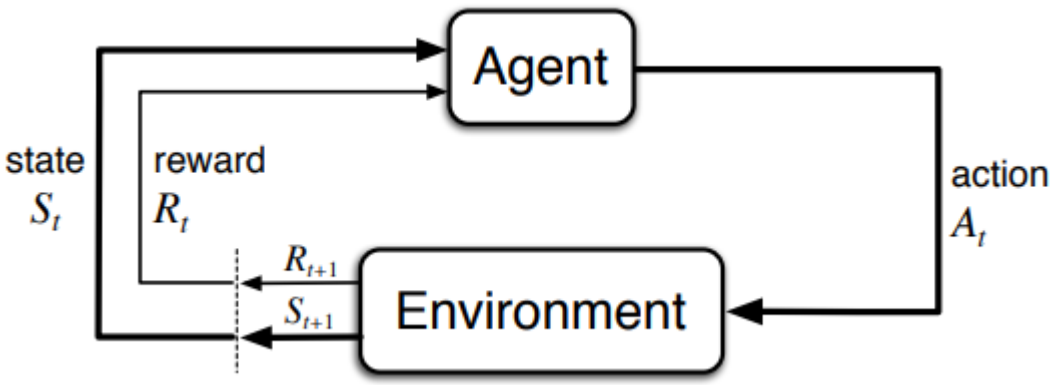


Figure 1: Feedback loop of reinforcement learning (Sutton and Barto, 2018)

3.1 Markov Decision Processes

The various sequential decision-making problems that RL is used for can be mathematically formulated using the Markov Decision Process (MDP) framework. This framework represents the problem using a 6-tuple $[\mathcal{A}, \mathcal{S}, r, P, \gamma, T]$, where $\mathcal{A}$ represents the action space, $\mathcal{S}$ the state space, and r the reward function that outputs scalar rewards R_t, such that $a_t \in \mathcal{A}$, $s_t \in \mathcal{S}$ and $R_t \in \mathbb{R}$. The policy is a mapping of $\mathcal{S} \to \mathcal{A}$ and the reward function r is a mapping of $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Furthermore, the (unknown) state transition kernel P maps $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, where we have

$$s_{t+1} \sim P\left(\cdot \mid s_t, a_t\right), \tag{3.1}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Furthermore, $\gamma \in (0, 1]$ defines the discount rate, where $\gamma < 1$ indicates that rewards in the distant future are worth less to the agent than immediate rewards. Finally, T is the decision horizon or trajectory length of the problem that the agent must solve. This can be infinite in some applications, but in our case, this is equal to the fixed 40-year investment period. Furthermore, the trajectory τ is a sequence of possible states and actions in the environment, $\tau = (s_0, a_0, s_1, a_1, \ldots)$, such that for the T-step trajectory we have

$$P(\tau \mid \pi) = \rho_0\left(s_0\right) \prod_{t=0}^{T-1} P\left(s_{t+1} \mid s_t, a_t\right) \pi\left(a_t \mid s_t\right), \tag{3.2}$$

where $P(\tau \mid \pi)$ is the probability of τ occurring while following policy π and $\rho_0\left(s_0\right)$ denotes the probability of the randomly sampled starting state s_0 (Achiam, 2018). Finally, the return $G(\tau)$ is

the total discounted reward obtained throughout the trajectory, defined as

$$G(\tau) = \sum_{t=1}^{T} \gamma^t R_t. \tag{3.3}$$

Using this notation, the central goal in RL can be formulated as finding the solution to the following optimization problem (Achiam, 2018):

$$\begin{aligned} J(\pi) &= \int_{\tau} P(\tau \mid \pi) G(\tau) = \mathrm{E}_{\pi}[G(\tau)], \\ \pi^* &= \arg\max_{\pi} J(\pi), \end{aligned} \tag{3.4}$$

where the objective function $J(\pi)$ is the expected return while following policy π and π^* the optimal policy for the MDP problem.

3.2 Policy gradient methods

In continuous action spaces, policy gradient techniques are typically used in order to solve (3.4) (Sutton and Barto, 2018). These methods parameterize a policy function with parameter vector θ and aim to estimate the policy gradient $\nabla_\theta J(\pi_\theta)$, which is the gradient of policy performance, and use it to update θ. Through the policy gradient theorem, whose proof can be found in the book of Sutton and Barto (2018), and following the derivation of Achiam (2018) we obtain the estimator for the policy gradient such that

$$\nabla_\theta J(\pi_\theta) = E_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a) \right], \tag{3.5}$$

where $\nabla_\theta \log \pi_\theta(s,a)$ is the gradient of the policy log-likelihood and $Q^{\pi_\theta}(s,a)$ is the state-action value function following policy π_θ. The state-action value function is defined as the expected return of the agent starting in state s after performing action a and following policy π_θ onwards (Achiam, 2018). It thus describes the inherent long-term desirability of a specific state-action combination. In mathematical terms, this becomes

$$Q^{\pi_\theta}(s,a) = \mathrm{E}_{\pi_\theta} \left[G(\tau) \mid s_0 = s, a_0 = a \right]. \tag{3.6}$$

Methods that only specify a policy function are so-called "actor-only" methods. An example of this is the REINFORCE algorithm, which uses the sampled returns as an estimate for $Q^{\pi_\theta}(s,a)$ to compute the gradient in (3.5) (Williams, 1992). Although such methods are viable in continuous

action spaces, they tend to be inefficient at evaluating their policy due to a large variance in their estimate for $Q^{\pi_\theta}(s,a)$ and typically converge to a local optimum instead of a global one (Silver et al., 2018).

Next to actor-only methods are critic-only algorithms, which aim to only estimate a state-action value function from which they derive the optimal actions in each state, and thus a policy (Sutton and Barto, 2018). However, these methods are restricted to discrete-action spaces and thus not viable in our problem setting.

Actor-critic methods combine the respective benefits of actor- and critic-only methods by specifying both a policy and a value function, where the policy (actor) is updated using information provided by the estimated value function (critic), which combats the high-variance problem in actor-only methods (Konda and Tsitsiklis, 1999). Note that most actor-critic methods approximate a state value function $V^{\pi_\theta}(s)$ instead of a state-action value function, where $V^{\pi_\theta}(s)$ gives the expected return when starting in state s and following policy π. $V^{\pi_\theta}(s)$ can also be interpreted as the expectation of $Q^{\pi_\theta}(s,a)$ for a random action a in state s. This allows for the advantage function $A^{\pi_\theta}(s,a)$ to be defined as

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s), \tag{3.7}$$

where $A^{\pi_\theta}(s,a)$ can be interpreted as the expected difference in return from performing action a in state s, as opposed to choosing a random action. This function is essential to many actor-critic methods since the state value function is a popular choice for the baseline function, which is a function that only depends on the state s and can thus be subtracted from $Q^{\pi_\theta}(s,a)$ in (3.5) without changing the expectation of the gradient (proof can be found in Achiam (2018)). This reduces the variance of the gradient and leads to a faster and more stable learning process. Thus, the final and most popular specification of the policy gradient in actor-critic algorithms becomes

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s))\right], \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right], \end{aligned} \tag{3.8}$$

where the actor and critic functions are often approximated using Artificial Neural Networks (ANN) in deep RL applications (Sutton and Barto, 2018; Qi et al., 2019).

3.3 On-policy methods

An important distinction between actor-critic algorithms is whether the agent learns on-policy or off-policy (Sutton and Barto, 2018). Whereas on-policy methods evaluate and improve the same policy that is used for the decision-making of the agent, off-policy methods improve a policy that is different from the one that is generating the training experience. The most popular off-policy algorithms in the literature are Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) and its successors Twin Delayed DDPG (TD3) (Fujimoto et al., 2018) and Soft Actor-Critic (SAC) (Haarnoja et al., 2018). For on-policy algorithms, these are Asynchronous Advantage Actor-Critic (A2C) (Mnih et al., 2016), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017).

The main idea behind choosing an off-policy method over an on-policy one is that off-policy algorithms do not need to explore the action space with the policy that they learn. This allows the learned policy to only concern itself with learning which actions lead to the most rewards, making the policy more "greedy". However, off-policy algorithms are less stable and more complex than on-policy methods, since they are not directly optimizing the policy with which they are performing the actions during the learning period (Sutton and Barto, 2018). Furthermore, Aboussalah et al. (2022) show in their research that likely due to this greediness, off-policy algorithms tend to construct more risky asset allocations than on-policy algorithms in portfolio optimization problems.

Regarding on-policy algorithms: unlike TRPO and PPO, A2C directly optimizes the objective function without clipping the loss, which can lead to destructively large updates (Schulman et al., 2017). Both TRPO and PPO limit large policy updates and thus tend to have more stable learning. Finally, since PPO is a simplified and more efficient version of TRPO, we opt to use PPO in our research for both our model-free and model-based algorithms.

3.4 Model-based RL

In this research, we apply the architecture of Yu et al. (2019) for our model-based method. A number of other model-based RL methods have also been developed over the past years that have seen success in their respective applications (Moerland et al., 2020). Although their design tends to be application-specific, these methods generally involve learning a model in order to let the agent predict and plan over future states. However, it is important to understand the fundamental difference between the environment in portfolio optimization problems and other environments in RL research. The majority of RL literature considers so-called "closed-loop" systems (Sutton and

Barto, 2018). These refer to problems where only the actions of the agent directly affect the next state of the environment, which in turn influences the next action taken by the agent. This is in contrast to the financial market environment, which can be seen as an open-loop system if we make the common assumption that the investments of the agent do not impact future asset returns (Feng et al., 2016). Furthermore, although they do take into account stochasticity in their models, most model-based RL methods in the literature have been developed to learn complex but largely deterministic dynamics (Buckman et al., 2018; Zhang et al., 2019). This adds more uncertainty to the question of how well these methods would translate to a more stochastic and noisy environment such as in our case. An example of this is the conventional model-based RL approach tried by Filos (2019). Here the author incorporates a VAR model and an RNN in order to predict and plan over future returns in a portfolio optimization problem, leading to an agent that is outperformed by non-RL benchmark strategies. We therefore decide to use a model-based framework that has been proven to be effective and capable of outperforming its model-free counterpart in a problem setting that is more closely related to ours.

4 Data

A major issue in applying RL to portfolio optimization problems lies in the fact that the amount of real-world financial market data tends to be too limited in order to train a robust agent. This is especially true when we are interested in learning long-term, low-frequency trading strategies such as in our case. To overcome this issue, we make use of synthetic data that is generated by the Dynamic Scenario Generator (DSG) of Ortec Finance (Steehouwer, 2016). This data comes in the form of 15,000 possible future economic scenarios, each containing eight yearly time series spanning forty years, starting in 2022. These eight series are composed of the annual returns of the seven asset categories mentioned in Section 2 and the annual inflation rate. Not only does the use of scenarios allow us to train the agent on a larger dataset (albeit artificially generated), but each scenario also replicates a possible 40-year investment period which the agent could encounter in the future. By letting the agent perform actions over a large number of possible investment periods while rewarding it accordingly, we are able to train an agent that learns a more robust trading strategy.

Finally, real-world historical data from 1991 to 2021 is available. This is used to initialize certain parts of the training algorithm and for the fitting of a number of prediction models. Ad-

ditionally, we also test some of the learned strategies using the historical data in order to observe their performances in a real-world setting.

4.1 Dynamic Scenario Generator

The DSG is an extensive model with many elements to it, but its general concept is as follows. It takes in over 600 financial and economic time series as input and decomposes each into a trend, business cycle and a monthly component (Steehouwer, 2016). Next, Dynamic Factor Models (DFMs) are constructed for each of these components, which consist of a number of dedicated factors to model the long-, medium- and short-term. The dynamics of these DFMs are the driving force behind the forecasted expectation and volatility of all variables, as they produce the scenarios for all the components. Finally, the separate component scenarios are merged in order to reconstruct the complete scenarios for all the input series. An overview of this process is shown in Figure 2. The full model consists of many other components such as the inclusion of a number of stylized facts, for which we refer to Steehouwer (2016).

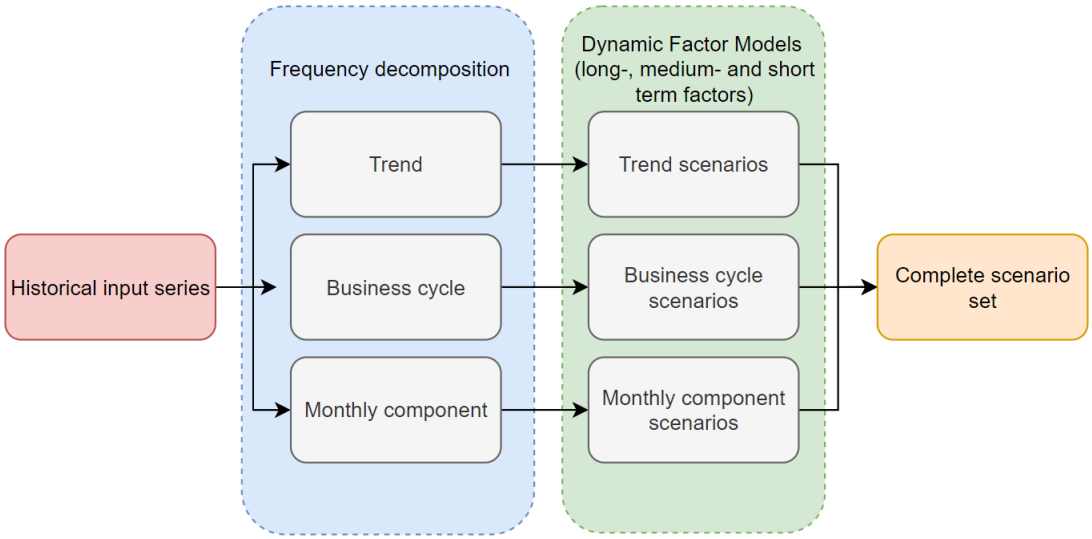


Figure 2: Overview of the frequency decomposition and use of dynamic factor models in the Dynamic Scenario Generator of Ortec Finance

4.2 Descriptive statistics

The dataset is split into a training, validation and test set using a 70/15/15 split, respectively. Furthermore, the splits are constructed such that the distribution of different types of scenarios is similar in each set. Table 2 presents a summary of the asset returns in the training scenario set

Table 2: Annual returns and rate of inflation [%] averaged across the training scenarios at various points in time

Year	Equity-World	Equity-Emerging markets	Fixed Income-High Yield	Fixed income - Investment Grade
1	6.155 (20.595)	9.242 (28.292)	0.806 (7.866)	1.432 (12.004)
20	7.054 (18.153)	9.587 (26.468)	3.497 (8.620)	5.524 (15.910)
40	7.371 (18.711)	9.619 (26.548)	4.948 (8.595)	6.714 (15.861)
Year	Cash USD	Real Estate Indirect	Hedge Funds-Fund of Funds	Inflation rate
1	0.246 (0.000)	5.143 (22.740)	2.844 (9.485)	3.474 (1.067)
20	1.742 (1.987)	7.301 (20.928)	3.291 (8.260)	2.208 (2.036)
40	2.423 (2.422)	7.247 (20.802)	3.391 (8.262)	2.103 (1.913)

Note: The values in brackets represent the standard deviation for each variable across the different scenarios.

at various points in time. We can see that there is a clear mix of assets with different risk levels available to the investor, with e.g equity having substantially larger standard deviations than assets in the fixed income category. Furthermore, the average return of all assets appears to increase over time, with the largest increase visible in the fixed income category. The standard deviation tends to generally remain the same among the assets, although a clear growth trend can be observed for Cash.

5 Methodology

5.1 State representation

When constructing the state that the agent will receive as input, it generally is the case that the more the state follows the Markov property, the higher the likelihood that the agent will learn a successful policy (Sutton and Barto, 2018). The Markov property holds when the future state transition depends only on the current state and not on the past history. We thus desire that the state vector contains all the relevant information such that

$$P\left(s_{t+1} = s' \mid s_t, s_{t-1}, \ldots, s_0\right) = P\left(s_{t+1} = s' \mid s_t\right). \tag{5.1}$$

Our model-free state representation is equal to that used by Dom et al. (2022), which is defined as

$$\mathbf{s_t} = [\mathbf{x_t}, \xi_t, \mathbf{a^*_{t-1}}, W_t, \mathcal{T}], \tag{5.2}$$

where $\mathbf{x_t}$ and ξ_t contain the first lag of the returns and the inflation rate at time t, respectively, $\mathbf{a^*_{t-1}}$ the current portfolio weights after they have been adjusted for the previous returns and $\mathcal{T} = T - t$ represents the time left until the terminal state is reached. This representation is similar to that of Yu et al. (2019), where we also include the inflation rate as it directly influences the inflation-adjusted wealth progression of the agent. Furthermore, including more lags of the asset returns in the state could increase the information available to the agent, but it is also important that the state remains concise. Ota et al. (2020) mention that generally, a state vector with a lower dimensionality will lead to a faster and overall better learning process for the agent. Yu et al. (2019) also only include one lag of the asset prices into their state, and Dom et al. (2022) found that adding more than one lag of the returns to the state space did not increase the performance of the agent. Furthermore, the inclusion of the current portfolio weights is necessary in order for the agent to take into account transaction costs (Jiang et al., 2017). Finally, it is natural that the time left and the current amount of wealth are informative for the agent regarding how close it is to reaching its goal. Pardo et al. (2018) argue that in time-limited tasks such as ours, the time left is an essential part of the state that must be included in order to not violate the Markov property.

Finally, Achiam (2018) recommends normalizing the state features whenever possible in order to enhance performance. Therefore, we scale the wealth and time left in the state such that

$$\begin{aligned} W'_t &= \frac{W_t - W^*_T}{W^*_T}, \\ \mathcal{T}' &= \frac{\mathcal{T} - \frac{1}{2}T}{T}, \end{aligned} \tag{5.3}$$

where W'_t and $\mathcal{T}'$ represent the scaled wealth and time left, respectively.

5.2 Reward function

The correct specification of the reward function is vital to successfully aid the agent in learning how to solve its task. Typically, we only want to reward the agent for achieving its exact goal in order to not influence its decision-making too much by over-engineering the reward function (Sutton and Barto, 2018). E.g, in chess, you would only reward the agent based on whether it loses, draws or wins the game, and not for any other achievements such as capturing pieces or improving board position. Applying this same reasoning to GBWM would make the specification of the reward

function relatively simple, as a clear goal is already defined by the investor. In our problem setting, we would thus only reward the agent based on whether or not it has achieved its wealth goal W_T^* at time T, which can be specified as

$$R_t = \begin{cases} 0 & \text{if } t < T, \\ \mathbb{1}_{W_T \geq W_T^*} & \text{if } t = T, \end{cases} \tag{5.4}$$

where $\mathbb{1}$ represents an indicator function. Although this function rewards the agent exactly for what we want it to achieve, the agent is likely to experience a large number of trajectories where it receives no rewards, making the number of nonzero rewards sparse. Andrychowicz et al. (2017) mention how in complex control tasks, reward functions that are too sparse can make it difficult for the agent to learn. This often leads researchers to perform reward shaping, where more elements are added to the reward function in order to provide the agent with supplementary rewards (Ng et al., 1999). In this research, we try both a shaped reward function and the simple specification of (5.4), which we refer to as the goal-only reward function.

For the shaped reward function, we use a specification similar to that of Dom et al. (2022). First, we add an element termed the Loss Given Failure (LGF) that penalizes the agent given that it has not reached its goal at the end of the trajectory. This makes it so that the agent is guaranteed to receive a nonzero reward in each trajectory. Furthermore, this penalization is scaled based on how far off the terminal wealth is from the goal wealth, such that large misses lead to a more negative reward. This is done in order to discourage overly risky strategies. The specification of the LGF then becomes

$$LGF = (\mathbb{1}_{W_T < W_T *}) \frac{W_T^* - W_T}{W_T^*}. \tag{5.5}$$

Additionally, at each timestep we include a reward based on a risk-adjusted return measure, namely the Downside Deviation Ratio (DDR), which is defined as

$$DDR_t = \frac{\mathbb{E}\left[r_t\right]}{dd_t}, \tag{5.6}$$

where r_t denotes the portfolio return at time t and dd_t is the downside deviation. The downside deviation is a risk measure that only takes into account the downside risk of assets. This makes it a more favourable risk measure than for example the Sharpe ratio, which is a symmetric measure of risk that also penalizes upside risk. This reasoning is also followed by Yu et al. (2019), who use

the DDR in their risk-adjustment module. Moody and Saffell (2001) define the downside deviation as the square root of the average of the squared negative returns, such that

$$dd_t = \left(\frac{1}{t} \sum_{s=1}^{t} \min \{r_s, 0\}^2 \right)^{\frac{1}{2}}. \tag{5.7}$$

Following Moody and Saffell (2001), we estimate the average return and downside deviation iteratively using exponential moving averages, such that

$$\begin{aligned} \widehat{\mathbb{E}}_{t+1}(r_t) &= \widehat{\mathbb{E}}_t(r_t) + \nu \left(r_{t+1} - \widehat{\mathbb{E}}_t(r_t) \right), \\ \mathrm{dd}_{t+1}^2 &= \mathrm{dd}_t^2 + \nu \left(\min \left\{ r_{t+1}^2, 0 \right\} - \mathrm{dd}_t^2 \right), \end{aligned} \tag{5.8}$$

where ν denotes the adaptation rate, which we set at the common value of 0.04 (Tsay, 2013). We initialize the moving averages with the entire historical data, where we assume a constantly rebalanced portfolio with a set of initial asset weights.

Bringing all these elements together leads to the shaped reward function that is formalized as

$$R_t = \begin{cases} \alpha_1 DDR_t & \text{if } t < T, \\ \mathbb{1}_{W_T \geq W_T^*} + \alpha_1 DDR_t + \alpha_2 LGF & \text{if } t = T, \end{cases} \tag{5.9}$$

where α_1 and α_2 are hyperparameters that let us scale the relative importance of each element in the reward function.

5.3 Actor and Critic network

Both the actor and critic network are specified using ANNs with two hidden layers, using Rectified Linear Unit (ReLU) activation functions in between hidden layers, similar to the work of Schulman et al. (2017). We follow the recommendation of Andrychowicz et al. (2020) to use separate layers for the actor and critic networks, as the authors show that this can lead to an improved overall performance in on-policy algorithms. Furthermore, since the actor network $\mu(s)$ must output a 7-dimensional vector that contains the desired asset weights, its output layer contains a softmax activation function such that $\sum_{i=1}^{7} \mu_i(s) = 1$. The critic network does not use a nonlinear activation function in the output layer, as it must simply output a scalar value $V(s)$. The widths of both networks are hyperparameters that are tuned.

During training, it is essential that the agent explores different parts of the action space in order for it to learn an effective policy (Sutton and Barto, 2018). However, if the output of the actor

network $\mu(s)$ is directly used as the action of the agent, we would have a deterministic policy. As a result, it is customary to use the output $\mu(s)$ as input to a random distribution from which we sample the actions. We specify a diagonal Gaussian policy function, which is commonly used in problems with continuous action spaces (Silver, 2015). Hence, the actor network outputs the mean of a Gaussian distribution, where the standard deviations σ_i can also be varied by the algorithm throughout the training process. We obtain the sampled weights $\tilde{\mathbf{a}}_t$ through

$$\tilde{\mathbf{a}}_t \sim \mathcal{N}\left(\mu(\mathbf{s}_t), \mathbf{\Sigma}\right). \tag{5.10}$$

where $\mathbf{\Sigma} = \text{diag}(\sigma_1^2, .., \sigma_m^2)$. To ensure that the sampled actions sum up to 1, we normalize them such that

$$\mathbf{a_t} = \frac{\tilde{\mathbf{a}}_t}{\sum_{i=m}^{7} \tilde{a}_{i,t}}, \tag{5.11}$$

where $\mathbf{a_t}$ denotes the corrected action vector containing the normalized asset weights. During testing we no longer need the agent to explore the action space and thus skip this sampling procedure, such that we have $\mathbf{a}_t = \mu(\mathbf{s}_t)$.

5.4 PPO

The model-free RL method that we use as a baseline in this research is the PPO algorithm introduced by Schulman et al. (2017). Our Python implementation is based on the code of Dom et al. (2022), which takes inspiration from the reliable PPO implementation of Raffin et al. (2021).

Firstly, the training set is split into a number of batches of scenarios, with the batch size N being a hyperparameter. During a training iteration, we let the agent perform actions throughout each scenario in the batch at every timestep t, where we store the trajectories containing the observed states, actions and rewards inside the memory of the algorithm. Furthermore, we also store the value estimates by the critic and the log of the probability of the actions using the density function of the action distribution (5.10).

Once these trajectories are collected, we perform an updating step. In order to calculate the objective function (3.8), we require the advantages A_t at each timestep t. These are estimated using a truncated version of General Advantage Estimation (GAE) such that

$$\begin{aligned}\hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t}\delta_T \\ &= \delta_t + \gamma\lambda\hat{A}_{t+1},\end{aligned} \tag{5.12}$$

where λ is a hyperparameter frequently set at 0.9 (Schulman et al., 2017). δ_t is computed using the stored values and rewards at time t, starting backwards from $t = T$ so that

$$\delta_t = R_t + \gamma V\left(\mathbf{s_{t+1}}\right) - V\left(\mathbf{s_t}\right). \tag{5.13}$$

With the advantages estimated, we can compute the clipped PPO objective function, which is defined as

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(p_t(\theta)\hat{A}_t, \text{clip}\left(p_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \text{with } p_t(\theta) = \frac{\pi_\theta\left(\mathbf{a_t} \mid \mathbf{s_t}\right)}{\pi_{\theta_{\text{old}}}\left(\mathbf{a_t} \mid \mathbf{s_t}\right)}, \tag{5.14}$$

where $p_t(\theta)$ is the probability ratio of a specific state-action pair in the new policy compared to the old policy. The clip function limits how much moving this probability ratio from 1 can influence the objective function and thus discourages large policy changes. ϵ is a hyperparameter that determines the interval $[1-\epsilon,\ 1+\epsilon]$ in which changing $p_t(\theta)$ still influences the objective function. Although the PPO objective appears complex at a first glance, it is relatively intuitive when we focus on what happens for certain signs of $\hat{A}_t$. In case $\hat{A}_t$ is negative, the action that we tried in a certain state had poor results and we increase the objective function by decreasing $p_t(\theta)$, resulting in the action becoming less likely in the new policy. However, when $p_t(\theta)$ is decreased below $(1-\epsilon)$, the $min(\cdot)$ function kicks in, meaning that decreasing this ratio any further has no more impact on the objective function. Similarly, if the advantage is positive, we increase the objective by increasing $p_t(\theta)$, but increasing $p_t(\theta)$ beyond $(1+\epsilon)$ does not affect the objective. Thus, PPO discourages large policy updates by clipping the objective from above when the advantage is positive and from below when it is negative.

In each updating step, we randomly sample mini-batches from the stored trajectories and update the policy parameter θ to maximize (5.14) using the Adam algorithm (Kingma and Ba, 2014). The same approach is used to update the critic parameters ϕ, where we use Adam to minimize the sum of squares differences between the estimated state values and the actual sampled returns, such that

$$L(\phi) = \hat{\mathbb{E}}_t\left[\left(V_\phi\left(\mathbf{s_t}\right) - \hat{G}_t\right)^2\right], \tag{5.15}$$

where $L(\phi)$ is the critic loss. We allow the actor and critic to be updated multiple times during a single learning step, where we sample new mini-batches from the memory. After each learning iteration, we clear the memory of the agent and collect new trajectories until we have looped a number of times over the entire training set. A full pseudo-code of this algorithm, including the model-based extension that we discuss in the next section, can be found in Appendix A.

5.5 Model-based PPO

For the model-based algorithm, we extend the regular PPO algorithm by including a prediction model that estimates the returns of the next timestep using lagged returns as regressors. The state space of the agent is then augmented with these forecasts, such that the new state becomes

$$\tilde{\mathbf{s}}_\mathbf{t} = [\mathbf{s}_\mathbf{t}, \hat{\mathbf{x}}_{\mathbf{t+1}}], \tag{5.16}$$

where $\tilde{\mathbf{s}}_\mathbf{t}$ represents the new augmented state and $\hat{\mathbf{x}}_{\mathbf{t+1}}$ the predicted asset returns. The reasoning behind this methodology is that we expect these forecasts to contain additional information concerning the next state, which would improve the Markov property of the state and in turn lead to performance gains. As for the possible prediction models, Yu et al. (2019) only investigate the use of complex nonlinear models. However, it is uncertain whether the estimation of such models is necessary, or if simple linear models could give similar or perhaps even better results. In this research, we investigate the effectiveness of both linear and nonlinear models as prediction models for model-based PPO.

5.5.1 Linear models

The first and most simple model that we estimate is an AR model of order 1 for each of the asset returns, such that we have

$$x_{i,t} = \psi_{0,i} + \psi_{1,i} x_{i,t-1} + \epsilon_{i,t}, \tag{5.17}$$

where $x_{i,t}$ denotes the returns of asset i at time t. These AR(1) models do not take into account correlations across different assets and are thus the simplest models that we can estimate. Although a common benchmark and an even simpler model that is used in the literature is a random walk (Henrique et al., 2018), it makes little sense to include it in our model-based comparison as we would simply be augmenting the state with duplicate values.

Furthermore, we estimate a VAR model that can capture correlations between the different assets and their lagged values. For a VAR of order p, the model specification becomes

$$\mathbf{x_t} = \mathbf{c} + \mathbf{A_1 x_{t-1}} + ... + \mathbf{A_p x_{t-p}} + \epsilon_\mathbf{t}, \tag{5.18}$$

where $\mathbf{A_1}$ to $\mathbf{A_p}$ are $(m \times m)$ coefficient matrices and $\mathbf{c}$ represents an $(m \times 1)$ vector of constants, in our case.

Aside from the different types of models, another important consideration that must be made is what data is used in order to estimate these models. If we are interested in utilizing as much data as possible, we can pool the training data and use this to estimate the model. However, it can be argued that since we know the data is simulated and not originating from real-world financial markets, we are simply estimating the Data Generating Process (DGP) of the scenario set and introducing positive model bias in our model-based approach. Although we know that the scenarios are generated with a vastly more extensive model than the prediction models that we estimate, this remains a valid concern when it comes to generalizing the results to real-world applications. A way to counteract this effect is to estimate the linear models using non-synthetic data, namely with historical data. Although this leads to a significantly smaller dataset, it also provides insight into how the agent performs when provided with perhaps less accurate predictions. We use both estimation methods in this research and compare their respective performances.

We use a lag length of 3 in the VAR model for the pooled data. As a robustness check, we also provide results in Appendix D for different lag lengths. Furthermore, we estimate an AR(3) model with the pooled data to study if capturing cross-correlations with the VAR(3) leads to an increase in performance. Finally, we cannot estimate a VAR(3) model on the smaller set of historical data due to the limited observations and hence estimate a VAR(1) using this dataset.

5.5.2 Nonlinear Gaussian Dynamic Boltzmann machine

The last model that we implement is the nonlinear model used in the framework of Yu et al. (2019), termed the Nonlinear Gaussian Dynamic Boltzmann Machine (N-G-DyBM).

At its foundation lie Restricted Boltzmann Machines (RBMs), which are stochastic ANNs that can be used to create generative models for binary data (Hinton, 2012). T. Osogami and Otsuka (2015) introduced the Dynamic Boltzmann Machine (DyBM), which uses a sequence of RBMs in order to be able to learn a temporal pattern and generate a sequence of binary time series.

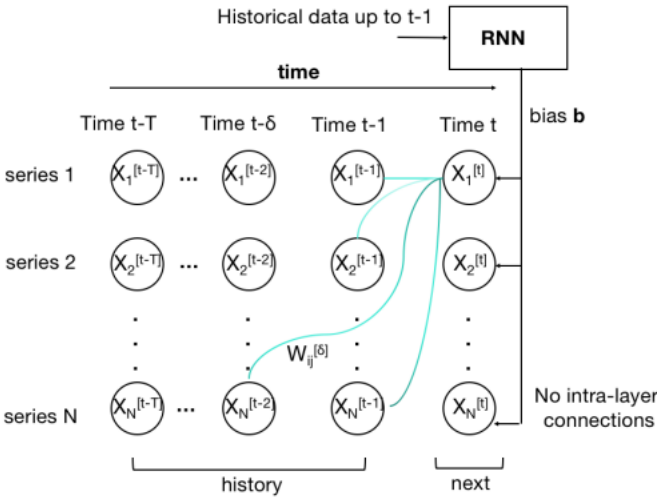


Figure 3: A Nonlinear Dynamic Boltzmann Machine network unfolded in time (Yu et al., 2019). Note that neurons within layers share no connections, which facilitates the conditional dependence assumption that is used to derive the update rule. The RNN layer updates the bias of the output and allows for nonlinear relationships within the time series to be captured

Finally, Dasgupta and Osogami (2017) extend the dynamic model to generate values based on a Gaussian distribution instead of only binary values, making the method suitable for real-valued time series generation. Furthermore, they incorporate a nonlinear component through an RNN output layer. Dasgupta and Osogami (2017) show that the N-G-DyBM can outperform standard VAR models when forecasting financial time series, while matching the performance of a Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) network at a much lower computational cost.

An overview of the network architecture is shown in Figure 3. There are $T+1$ layers of size N, where N equals the dimension of the time series. Each neuron at time t consists of three sorts of parameters, namely a bias term, variance and weight parameters. The model assumes that each unit $x_{i,t}$ follows a Gaussian distribution and is conditionally independent from other units at time t, given the complete history of the time series. Using this independency, the conditional probability density of $\mathbf{x_t}$ given this history becomes

$$p\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, .., \mathbf{x}_0\right) = \prod_{i=1}^{N} p_i\left(x_{i,t} \mid \mathbf{x}_{t-1}, .., \mathbf{x}_0\right), \tag{5.19}$$

where p and p_i represent the conditional probability density of $\mathbf{x}_t$ and $x_{i,t}$ given the history of $\mathbf{x}_t$, respectively. Furthermore, due to the assumption that each series follows a Gaussian distribution, the conditional probability distribution for each component $x_{i,t}$ is

$$p_i\left(x_{i,t} \mid \mathbf{x}_{t-1}, .., \mathbf{x}_0\right) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{\left(x_{i,t} - \mu_{i,t}\right)^2}{\sigma_{i,t}^2}\right), \tag{5.20}$$

where $\mu_{i,t}$ and σ_i^2 represent the mean of unit x_i at time t and its variance, respectively. Furthermore, in order to let the number of parameters be independent of time, only layers up to a lag of $d-1$, where d is termed the delay, are connected to the units at time t, as shown in Figure 3. Moreover, Dasgupta and Osogami (2017) introduce eligibility traces $\alpha_{i,k}$ for each component x_i, such that

$$\alpha_{i,k,t} = \sum_{\delta=d}^{\infty} \lambda_k^{\delta-d} x_{i,t-\delta} \tag{5.21}$$

where $\alpha_{i,k,t}$ represents the exponential moving average of x_i at time t, using a decay rate λ_k for $k = 1, .., K$. By perceiving the eligiblity traces as additional regressors, we can interpret the network as a VAR model with lag order $(d-1)$ such that we have

$$\mu_t = \mathbf{b}_t + \sum_{\delta=1}^{d-1} \mathbf{W}_\delta \mathbf{x}_{t-\delta} + \sum_{k=1}^{K} \mathbf{U}_k \alpha_{k,t-1}, \tag{5.22}$$

where $\mathbf{W}_\delta$ and $\mathbf{U}_k$ are coefficient matrices for the lags and the eligibility trace vector α_k, respectively, and $\mathbf{b}_t$ the bias vector at time t.

Furthermore, the RNN layer is used to update the bias vector such that

$$\mathbf{b}_t = \mathbf{b}_{t-1} + \mathbf{A}^\top \boldsymbol{\Psi}_t, \tag{5.23}$$

where $\mathbf{A}$ is the $M \times N$ learned output weight matrix and $\boldsymbol{\Psi}_t$ is the $M \times 1$ state vector at time t, for an M dimensional RNN. This state vector is updated at each timestep using the time series such that

$$\boldsymbol{\Psi}_t = \tanh\left(\mathbf{W}_{rnn} \Psi_{t-1} + \mathbf{W}_{in} \mathbf{x}_t\right), \tag{5.24}$$

where $\mathbf{W}_{rnn}$ and $\mathbf{W}_{in}$ correspond to the internal RNN weight matrix and the weight matrix used to project the time series input to the RNN layer, respectively.

Unlike the linear models, the N-G-DyBM is designed as an online learning model. By maximizing the log-likelihood, Dasgupta and Osogami (2017) derive an online stochastic gradient update rule. Hence, similarly to Yu et al. (2019), we train this model together with the agent, where its parameters are updated at each timestep of the scenarios. We also follow their work in the choices

for the different model settings, which are detailed in Appendix E. Furthermore, we set the delay parameter to 4 such that the model uses a similar amount of lags as the VAR(3) model. Finally, our implementation of the N-G-DyBM is based on the open-source code repository of Osogami and Young (2016).

5.6 Behaviour cloning

Yu et al. (2019) implement a dynamic form of behaviour cloning in their model-based framework, which they find to be effective in reducing the riskiness of the constructed asset allocations. We therefore also choose to investigate the use of behaviour cloning in our problem setting. However, we do separate this methodology from our previously discussed model-based approach, since behaviour cloning is a general learning technique that can be applied in RL and is thus not inherently model-based.

Behaviour cloning is a branch of imitation learning where an agent tries to learn how to solve a task by mimicking the actions of an expert (Abbeel and Ng, 2004). Imitation learning can be categorized into either passive learning or active learning. In passive learning, all expert demonstrations are obtained prior to the learning process of the agent, whereas in active learning, the expert demonstrations are inferred from the actions of the agent. The form of behaviour cloning that is applied in this research falls under the active learning paradigm.

Ideally, the actions of this expert are in line with the optimal policy that we want the agent to learn. Since in the case of Yu et al. (2019) the goal of the agent is to maximize the portfolio returns at each timestep, the expert action can simply be derived from this condition. Although maximizing returns is not the goal of our agent, we do know that in case the agent were to mimic the expert actions, it would always achieve its goal. This way of deriving the expert actions thus leads to a "pseudo-optimal" policy and could still have a positive influence on the learning process of the agent.

If the future returns and the current asset weights are known, the expert action is found by solving the following optimization problem:

$$\begin{aligned} &\max_{\mathbf{a}_t} \mathbf{a}_t \cdot \mathbf{x}_t - c \sum_{i=1}^{m} |a_{i,t} - a_{i,t-1}| \\ &\text{s.t.} \sum_{i=0}^{m} a_{i,t} = 1, \quad 0 \leq a_{i,t} \leq 1, \qquad \forall i \end{aligned} \tag{5.25}$$

where we maximize the returns while taking into account transaction costs. However, since the

objective function specified in (5.25) contains an absolute value term, it is not differentiable and hence not solvable using linear programming. This is fixed by introducing additional variables in order to linearize the objective function such that

$$
\begin{aligned}
&\max_{(\mathbf{a}_t, \Delta\mathbf{a_t}^+, \Delta\mathbf{a_t}^-)} \mathbf{a}_t \cdot \mathbf{x}_t - (\Delta \boldsymbol{a}_t^+ + \Delta \boldsymbol{a}_t^-) \cdot \boldsymbol{c} \\
\text{s.t. } &\sum_{i=0}^{m} a_{i,t} = 1, \\
&\mathbf{a_t} = \mathbf{a_{t-1}} + \Delta\mathbf{a_t}^+ - \Delta\mathbf{a_t}^-, \\
&a_{i,t} \geq 0, \qquad \forall i \\
&\Delta a_{i,t}^+ \geq 0, \qquad \forall i \\
&\Delta a_{i,t}^- \geq 0, \qquad \forall i
\end{aligned}
\tag{5.26}
$$

where $\Delta\mathbf{a}_{\mathbf{i,t}}^-$ and $\Delta\mathbf{a}_{\mathbf{i,t}}^+$ represent the negative and positive change in the portfolio weight for asset i with respect to the previous timestep. Furthermore, we also include the asset weight constraints specified in Table 1.

We implement the behaviour cloning procedure as follows. At each timestep t after the agent performs its action $\mathbf{a_t}$, the current asset weights and new returns are used in order to solve (5.26) and obtain the expert action $\bar{\mathbf{a}}_\mathbf{t}$, which is stored in the memory. When the actor network is updated, the log-loss $\bar{L}^\mu$ between the actions of the agent and those of the expert is calculated using the binary cross-entropy loss (Yu et al., 2019). This loss is defined as

$$
\bar{L}^\mu = -1 \times \frac{\sum_{i=1}^{N} \sum_{j=0}^{m} \bar{a}_{i,j} \log\left(\mu\left(\mathbf{s_i}\right)\right)_j + \left(1 - \bar{a}_{i,j}\right) \log\left(1 - \left(\mu\left(\mathbf{s_i}\right)\right)_j\right)}{N \cdot m}, \tag{5.27}
$$

where $\bar{L}^\mu$ represents the behaviour cloning loss. Gradients with respect to the actor network can then be calculated and used to update θ slightly, where we discount this gradient with a factor λ_{BC} to maintain a stable learning process, with λ_{BC} being a hyperparameter.

5.7 Enforcing portfolio constraints

In real-world applications, it is possible that an investor desires that certain asset weights stay within specific bounds. We therefore investigate how the model-free and model-based methods perform when such additional constraints are introduced. There are various methods in order to enforce bounds on the asset weight. A simple "brute-force" option is to apply quadratic programming in order to find the action vector that is the closest to the original action of the agent while also

satisfying all constraints. This comes down to solving the following optimization problem, where we use the Euclidean distance as a measure of closeness such that we have:

$$\begin{aligned} \underset{\boldsymbol{a}^*}{\text{minimize}} \quad & \|\boldsymbol{a}^* - \boldsymbol{a}\|_2^2, \\ \text{subject to} \quad & \mathbf{G}\boldsymbol{a}^* \leq \boldsymbol{h}, \\ & \boldsymbol{\iota}'\boldsymbol{a}^* = 1, \end{aligned} \tag{5.28}$$

where $\mathbf{a}^*$ is the optimized weight vector that satisfies all constraints, $\mathbf{a}$ the original weight vector, $\mathbf{G}$ and $\mathbf{h}$ a matrix and vector that contain the inequality constraints of Table 1, respectively, and $\boldsymbol{\iota}$ a vector of ones.

5.8 Interpretability

5.8.1 Shapley values

Once the agent is trained, we are left with a deterministic actor function $\mu_\theta(s)$ that represents the policy. Although we can determine whether the policy is successful or not, it is also valuable to understand what information in the state is most important to the agent, and how it uses this information in order to decide what action to take. However, the actor and critic functions of the agent are represented by ANNs, which makes them difficult to interpret.

In recent years, the SHapley Additive exPlanations (SHAP) method of Lundberg and Lee (2017) has become a popular tool for gaining insight into how the different input features in machine learning models influence the eventual prediction. SHAP is based on Shapley values which stem from coalitional game theory, where they are used as a method to distribute payouts within a coalition of players based on their respective contributions to the total payout (Shapley, 1953). When it comes to model explainability, the "players" are the individual features that are used to create the prediction, and the difference between a specific prediction and the average prediction is the payout.

SHAP specifies an explanation model as

$$g\left(\mathbf{z}'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i', \tag{5.29}$$

where $g(\cdot)$ is the linear explanation model that takes in binary values, $\mathbf{z}' \in \{0,1\}^M$ the coalition vector with M the maximum coalition size and ϕ_i the feature attribution for a particular feature i. ϕ_0 represents the output of the model where all features are absent, i.e the average prediction.

Within a coalition for a particular prediction, we refer to the features that are "present" (playing) with a 1, and features that are "absent" with a 0. The actual feature values for a particular observation are used for the present features, whereas absent features are seen as missing. Since most models cannot handle missing features, these values are replaced with randomly sampled feature values from the data. The mapping from a binary vector $\mathbf{z}'$ to an actual feature vector is done by a mapping function $h_x(\mathbf{z}')$. By sampling all possible coalitions for a certain prediction, we can estimate the marginal contributions to the payout of each feature, which represent the Shapley values.

In this research, we use the Deep SHAP algorithm to approximate the Shapley values, since this is the most computationally efficient approximation method for ANNs (Molnar, 2020). However, a major drawback of SHAP in ANNs is that it assumes that all features are independent of each other. This is certainly not the case in our problem setting, as this assumption clearly does not hold for e.g the portfolio weights. There are possible extensions to SHAP that aim to reduce its inaccuracy when features are dependent, but these extensions are difficult to implement (Aas et al., 2021). Since model explainability is not the main focus of this research, we remain using the somewhat flawed method of Deep SHAP and keep these limitations in mind when analyzing the results.

5.8.2 Perfect predictions

Although Shapley values give insight into the variable importance of each element in the state, they do not paint the full picture of how the agent interprets these different elements. For example, even though our model-based method augments the state with predictions of the future asset returns, there is no guarantee that the agent also understands that these values are meant to be indications of the future returns. In order to get an idea of how much information the agent infers from these predictions, we investigate the effect of replacing the model forecasts in the state with the actual returns of the next timestep, such that $\hat{\mathbf{x}}_{\mathbf{t+1}} = \mathbf{x}_{\mathbf{t+1}}$. The agent thus observes the state augmented with the forecasts from the prediction model during training, and then receives "perfect predictions" of the future returns during testing. This novel technique can show us how much trust the agent places in the predictions of each model, or if it even has some understanding that these features are predictions in the first place. For example, in case the agent places a large amount of trust into these predictions, we expect that giving it perfect predictions will lead to a significant increase in performance during testing.

5.9 Performance measures

When comparing different asset allocation strategies, the most important performance metric in GBWM is the expected probability of achieving the goal. We estimate this using the Goal Success Rate (GSR), which we define as the success rate of the strategy at achieving the goal of the investor in a given scenario set. The GSR is calculated as

$$GSR = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{W_{i,T} \leq W_T^*}, \tag{5.30}$$

where $W_{i,T}$ is the terminal wealth achieved in scenario i and N the number of scenarios that are used to test the agent. Furthermore, we confirm that a 100% success rate is theoretically possible by analyzing the performance of a strategy that always follows the expert actions described in Section 5.6, which comfortably reaches the goal wealth in all scenarios.

Although the success rate is the most important performance metric, it is also useful to include secondary measures of performance in order to have a more in-depth comparison between the learned strategies. An example of such a measure is the average terminal wealth, which indicates how successful the agent is at accumulating wealth. Since the average can be heavily influenced by large outliers, which are likely to be present in risky strategies, we also include the median terminal wealth as a more robust measure of wealth accumulation.

Moreover, we use the distribution of final wealth to calculate the Value at Risk (VaR) and Conditional Value at Risk (CVaR), where VaR_α is the (1-α)% percentile of the terminal wealth, and $CVaR_\alpha$ the average terminal wealth given that it is below VaR_α. These measures provide insight into the downside risk of the investment strategies, where low values for VaR_α and $CVaR_\alpha$ are indications of more risky strategies. Furthermore, we also evaluate the agents on their average returns and DDR across all test scenarios.

Aside from the performance of the learned investment strategies, another aspect that we compare between the model-based and model-free methods is the sample efficiency of the algorithm. Model-based RL methods are known to be significantly more sample efficient than model-free methods, requiring fewer training iterations in order to converge to a (local) optimum policy (Moerland et al., 2020). We study the sample efficiency of the algorithms by comparing how quickly the learning curves for both methods converge during training.

Finally, we use the Root Mean Squared Error (RMSE) to gauge the accuracy of the different prediction models, where we calculate the RMSE as

$$RMSE = \frac{1}{m}\sum_{j=1}^{m}\sqrt{\sum_{i=1}^{N}\sum_{t=1}^{T}(\tilde{\boldsymbol{x}}_{i,j,t} - \boldsymbol{x}_{i,j,t})^2}, \tag{5.31}$$

where $\tilde{x}_{i,j,t}$ and $x_{i,j,t}$ are the predicted and actual returns at time t for scenario i and asset j, respectively.

5.10 Hyperparameter tuning

Hyperparameter tuning is a crucial step in RL that can greatly affect the final performance of the agent (Achiam, 2018). In fact, Henderson et al. (2018) show that a large number of performance discrepancies between algorithms can be attributed to code optimizations and different hyperparameters, instead of actual conceptual differences. Thus, in order to perform a comparison between the model-based and model-free algorithms such that we can accredit differences in performance due to the (lack of) presence of a model, we opt to only perform hyperparameter tuning for the model-free algorithm. These tuned hyperparameters are then also used for the model-based algorithms.

Andrychowicz et al. (2020) provide in their research some recommendations for hyperparameters in on-policy RL algorithms. These are used together with the hyperparameter tuning results of Dom et al. (2022) to construct a search space for each hyperparameter. Restricted by the computational costs of training, we refrain from using k-fold cross validation and perform a random search over the search spaces. Furthermore, random search has shown to be more efficient than grid search when it comes to finding the optimal set of hyperparameters (Bergstra and Bengio, 2012). The results of the hyperparameter tuning can be found in Appendix C, where we choose the set of hyperparameters that achieve the largest GSR in the validation scenario set.

6 Results

6.1 Algorithm performance comparison

Table 3 displays the performance of the different model-free and model-based RL methods for both the goal-only and the shaped reward function, without enforcing additional asset weight constraints. The naming convention is such that the model-based methods are denoted by the prediction model that they incorporate, followed by the data used to estimate the model ("-H" for the historical data, "-P" for the pooled scenarios of the training data).

Within the goal-only reward setting, we find that all model-based strategies outperform the

Table 3: Performances of the RL methods in the test set for both reward functions

Metrics	Model-Free	AR(1)-H	AR(1)-P	AR(3)-P	VAR(1)-H	VAR(3)-P	N-G-DyBM
Goal-only Reward							
GSR	88.6	91.8	90.5	90.9	90.9	**92.1**	91.7
Expected wealth	913.6	865.0	840.5	879.4	871.1	**949.2**	925.2
Median wealth	864.1	804.6	791.2	840.0	814.0	**886.0**	879.7
$\text{VaR}_{0.95}$	208.7	329.5	296.9	288.4	306.4	**338.2**	312.7
$\text{CVaR}_{0.95}$	122.5	228.2	222.2	200.7	214.3	**245.1**	217.7
Expected returns	6.23	6.24	6.22	6.41	6.28	**6.75**	6.41
$D\bar{D}R$	1.006	1.099	1.080	1.096	1.100	1.113	**1.191**
Shaped Reward							
GSR	90.1	91.8	92.0	91.7	92.0	**93.2**	91.9
Expected wealth	1446.0	1403.3	1351.4	1312.7	1251.9	1456.3	**1539.3**
Median wealth	1155.1	1060.8	1063.4	1041.9	1077.7	1196.0	**1204.1**
$\text{VaR}_{0.95}$	280.0	326.8	335.1	332.7	329.7	**348.9**	321.8
$\text{CVaR}_{0.95}$	180.9	240.4	240.4	241.4	236.2	**250.3**	229.4
Expected returns	**7.61**	7.23	7.12	7.12	7.10	7.47	7.41
$D\bar{D}R$	1.092	1.247	1.278	1.276	1.270	**1.339**	1.320

Note: All the model-based agents are referred to by the prediction model that they use, where N-G-DyBM refers to the nonlinear model used in Yu et al. (2019). "-P" and "-H" denote if the linear model is trained using the pooled training data or the historical data, respectively. The GSR refers to the Goal Success Rate in %. The expected and median wealth and the VaR and $CVaR$ are all estimated at the final state and given in $ [x1000]. Finally, the expected returns and Downside Deviation Ratio (DDR) are averaged over time, with the returns shown in %.

model-free strategy in terms of GSR. Whereas the model-free baseline achieves a GSR of 88.6%, the model-based algorithms reach success rates of up to 92.1%, with the lowest being 90.5%. Furthermore, although the model-free agent attains a relatively high average terminal wealth, it does so with a considerably larger amount of risk, indicating that this agent likely accumulates such high terminal wealth by constructing more volatile portfolios. The risk metrics are also where the model-based agents appear to have the largest gain over the model-free agent, as can be seen in the considerably higher $\text{VaR}_{0.95}$ and $\text{CVaR}_{0.95}$ of the model-based strategies.

Moving on to the agents trained using the shaped reward function, we notice a slight increase in the GSR for the model-free agent (from 88.6% to 90.1%), combined with a nearly 60% increase in average terminal wealth and a reduction in the riskiness of the strategy. This is likely a result of the agent being rewarded for achieving high risk-adjusted returns at each timestep and receiving a penalty for large misses of its wealth goal. The substantial increase in average acquired terminal wealth is consistent across all strategies trained under the shaped reward function, coupled with an increase in average DDR. We also observe larger discrepancies between the average and median final wealth, which indicates that the use of the shaped reward function increases the number of outliers in the right tail of the final wealth distribution. Furthermore, we again find that all model-based

strategies achieve a higher GSR than the model-free baseline. Although the model-free strategy does have the highest average returns, it again performs the worst on all risk metrics.

Moreover, when we compare the model-based strategies to each other, we see that the agent using the VAR(3)-P prediction model achieves the highest performance on nearly all metrics in both the goal-only and shaped reward setting. The nonlinear model also seems to perform well, attaining the largest average final wealth with the shaped reward function and a relatively high GSR under the goal-only reward function together with the AR(1)-H model. However, we must note that running the algorithms with different random seeds can already lead to variations of up 1 to 1.5 percentage points in the GSR. Hence, while the difference in performance between the model-free and model-based methods appears to be substantial, differences in GSR between the various model-based strategies seem negligibly small. We thus find that even with forecasts obtained from simplistic linear regressions such as the AR(1)-H model, we can already achieve a substantial performance gain over the model-free baseline.

Next, we enforce the asset constraints described in Table 1 during the testing of the learned strategies. The performances under these conditions are shown in Table 4 for the goal-only reward setting. Aside from a general performance decrease across all strategies, which is consistent with the results of Dom et al. (2022), we see that the previous result of all model-based agents achieving a higher GSR than the model-free agent remains true.

Table 4: Performances of the RL methods with asset constraints enforced, using the goal-only reward function

Metrics	Model-Free	AR(1)-H	AR(1)-P	AR(3)-P	VAR(1)-H	VAR(3)-P	N-G-DyBM
GSR	81.9	85.6	85.0	**86.2**	84.7	86.1	85.6
Expected wealth	724.5	747.5	739.6	785.5	758.7	**863.6**	773.2
Median wealth	670.4	684.5	688.8	735.0	697.5	**756.0**	719.8
$\text{VaR}_{0.95}$	221.6	267.4	239.9	242.7	237.3	**264.5**	259.1
$\text{CVaR}_{0.95}$	163.7	194.2	187.7	172.4	170.1	**188.4**	186.4
Expected returns	5.75	5.77	5.80	6.00	5.79	**6.11**	5.84
$D\bar{D}R$	0.927	0.994	0.984	1.000	1.006	1.012	**1.030**

Note: All the model-based agents are referred to by the prediction model that they use, where N-G-DyBM refers to the nonlinear model used in Yu et al. (2019). "-P" and "-H" denote if the linear model is trained using the pooled training data or the historical data, respectively. The GSR refers to the Goal Success Rate in %. The expected and median wealth and the VaR and $CVaR$ are all estimated at the final state and given in \$ [x1000]. Finally, the expected returns and Downside Deviation Ratio (DDR) are averaged over time, with the returns shown in %.

We further compare the model-based and model-free methods in the remaining sections, where we use the model-based agent that is trained using the VAR3-P prediction model to generate the results for the model-based strategy, which we refer to as the model-based agent moving forward.

Table 5: Performances of the model-based and model-free methods with Behaviour Cloning (BC)

Metrics	Model-Free	Model-Free+BC	Model-Based	Model-Based+BC
GSR	**88.6**	86.8	**92.1**	88.9
Expected wealth	913.6	**954.8**	949.2	**1110.0**
Median wealth	**864.1**	788.2	866.0	**962.9**
$\text{VaR}_{0.95}$	208.7	**284.3**	**338.2**	279.7
$\text{CVaR}_{0.90}$	122.5	**211.5**	**245.1**	198.0
Expected returns	6.23	**6.53**	6.75	**7.09**
DDR	**1.006**	0.982	**1.113**	1.015

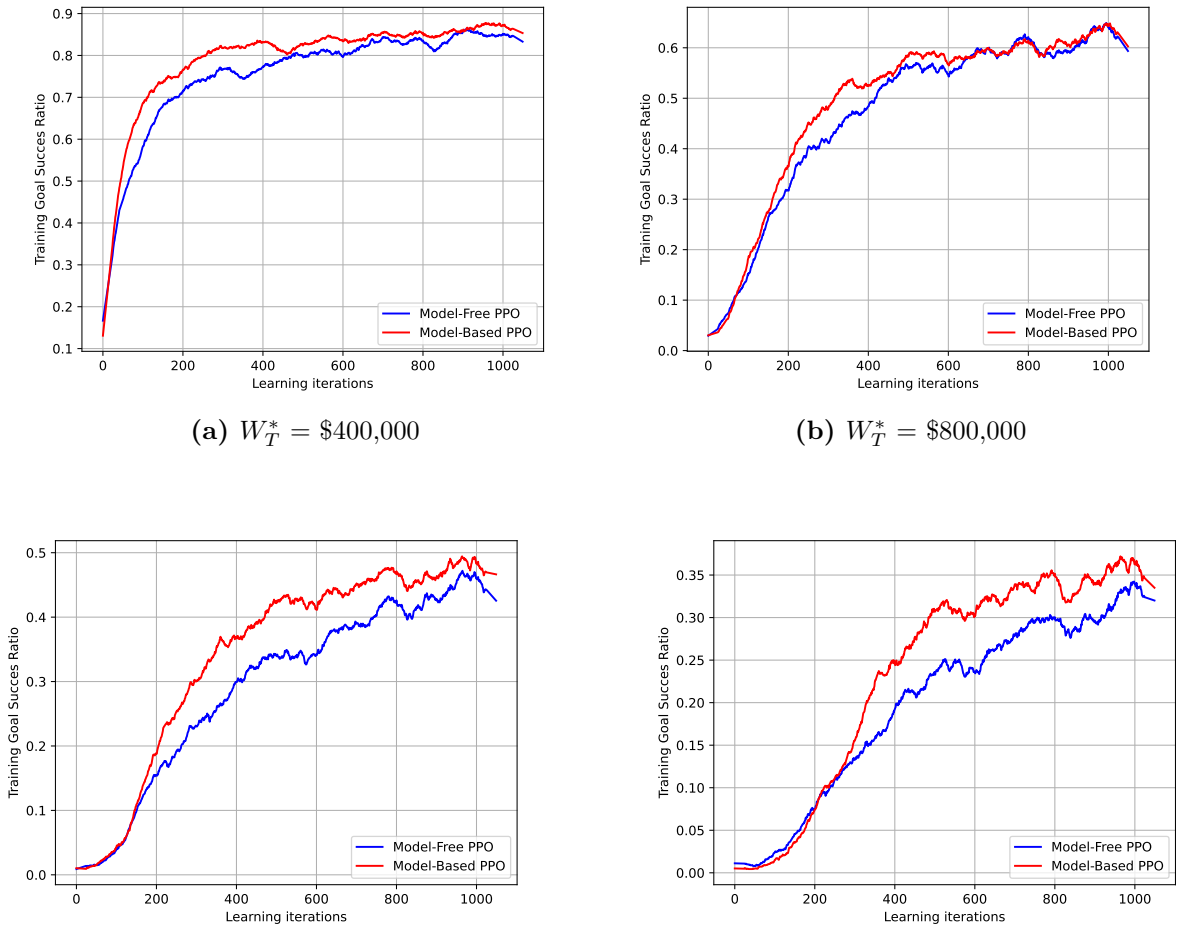
Note: All methods are trained using the shaped reward function and the model-based method uses the pooled VAR(3) model. Furthermore, the GSR refers to the Goal Success Rate in %. The expected and median wealth and the VaR and $CVaR$ are all estimated at the final state and given in \$ [x1000]. Finally, the expected returns and Downside Deviation Ratio (DDR) are averaged over time, with the returns shown in %.

6.2 Behaviour cloning

Moving on, we implement behaviour cloning in both our model-free and model-based algorithm and evaluate the performance of the new learned strategies on the test set. The results are shown in Table 5, where we also again include the results without behaviour cloning for ease of comparison. We find that in both the model-free and model-based case, behaviour cloning leads to an increase in average returns and terminal wealth but a decrease in the GSR. Furthermore, we note that although the average returns increase, the DDR in fact decreases, showing that behaviour cloning actually stimulates learning more risky strategies. This result is in contrast to the findings of Yu et al. (2019), where the incorporation of behaviour cloning lead to a reduction in risk. A possible explanation for this is that the goal in their problem setting is to find an allocation strategy that maximizes returns. This means that the expert actions are all greedy actions in the sense that they maximize the rewards received by the agent at each timestep. However, although we discussed that the expert is "pseudo-optimal" in our setting, maximizing the returns is not exactly in line with achieving the goal of the agent. An example of this is when the wealth goal of the investor has already been reached before the end of the investment period. In such a case, we can imagine that it would be beneficial for the agent to switch to a more risk-averse strategy and focus on wealth preservation instead of aggressive growth. Hence, the learning signals from behaviour cloning and from the goal-focused rewards can become conflicting, which in turn deteriorates the performance of the agent. Furthermore, the addition of behaviour cloning also leads to a significant increase in the computational cost of the learning algorithm, due to the addition of needing to solve the optimization problem of (5.26) at every timestep in order to find the expert actions.

6.3 Sample efficiency

We study the sample efficiency of the model-free and model-based agents by comparing their respective learning curves, which are shown in Figure 4. The curves show the achieved GSR of the agent during training against the number of completed scenario batches. Moreover, both agents are trained using the goal-only reward function and for increasingly larger final wealth goals of the investor in order to study the effect on the learning process when rewards become more sparse. The curves are smoothed using a Savitzky-Golay filter for clarity.



(a) $W_T^* = \$400{,}000$ **(b)** $W_T^* = \$800{,}000$

(c) $W_T^* = \$1{,}200{,}000$ **(d)** $W_T^* = \$1{,}600{,}000$

Figure 4: Comparison of the sample efficiency of the model-free and model-based algorithms. The model-based algorithm here uses the VAR(3) prediction model estimated on the pooled scenario training data. The learning curves are produced using the goal-only reward function for various terminal wealth goals.

We observe that the model-based agent tends to converge at a slightly faster rate than the model-free agent, as seen from the earlier flattening of the learning curve. As expected, both algorithms

take longer to converge when the goal becomes more difficult and rewards more sparse. We also see that the difference in sample efficiency becomes more apparent in this increased sparse reward setting. Thus, despite the fact that the state of the model-based agent has a higher dimensionality than that of the model-free agent, it is still able to converge to its (local) optimal policy faster. Although this increased sample efficiency is not orders of magnitude higher than that of the model-free method as is usually the case for model-based algorithms, such algorithms tend to use the model predictions to directly create a future plan of actions (Moerland et al., 2020). We thus find that the implicit use of the predictions through state augmentation is also beneficial to the sample efficiency of the algorithm.

6.4 Backtesting

After comparing the model-free and model-based agents with the test scenarios, we perform a backtest using the historical data in order to observe how the learned strategies perform when applied on non-synthetic data. The results are shown in Figure 5, where we plot the changes in wealth over time using the different allocation strategies. Note that the investment period here is only 28 years, taking into account the 3 required lags for the prediction model, instead of the usual 40-year period that the agents is trained on. We take this into account by adjusting the scaling factors of the state accordingly. Furthermore, we include the performance of a Constantly Rebalanced Portfolio (CRP) strategy, which distributes the wealth equally over all assets at each timestep (Cover, 1991). This strategy is a commonly used benchmark in dynamic asset allocation research and can also be interpreted as an initialized RL agent without any learning experience (Hoi et al., 2021).

Similar to the results for the test scenario set, the model-based strategy appears to outperform the model-free strategy, reaching a higher wealth at the end of the investment period. Certain major events such as the 2008 financial crisis can be observed in the graph, where we see the model-free agent losing a larger portion of wealth than the model-based agent, which is perhaps another indication of the model-free agent taking more risk. Furthermore, we find that both the RL strategies have a higher final wealth than the benchmark, although this difference is considerably larger for the model-based agent.

Finally, we must note that this is not a "formal" backtest, in the sense that we know that the scenarios on which the agents are trained include economic insight that is derived from the historical data, which introduces look-ahead bias. A more sophisticated form of backtesting is required in

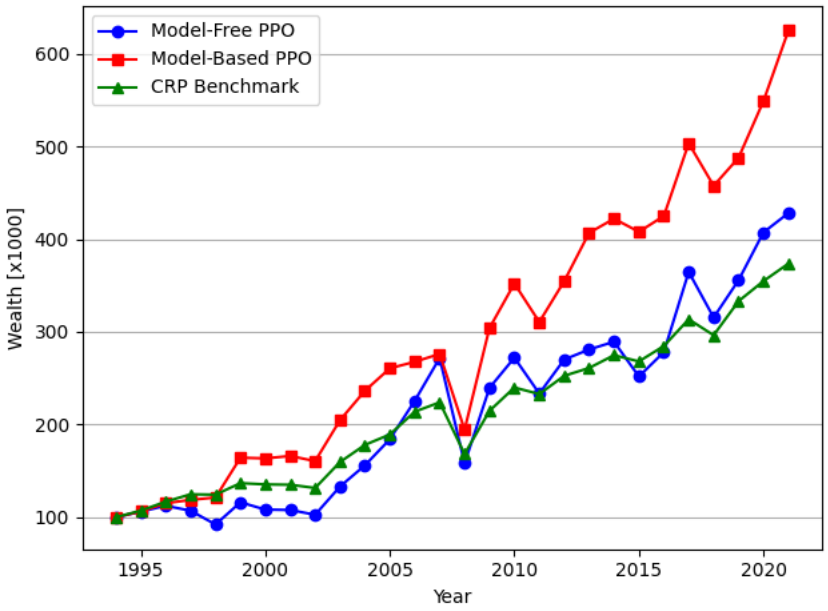


Figure 5: A comparison of the different allocation strategies applied on historical data. We use the pooled VAR(3) model for model-based PPO and include a Constantly Rebalanced Portfolio (CRP) benchmark that reconstructs an equal weights portfolio at each timestep

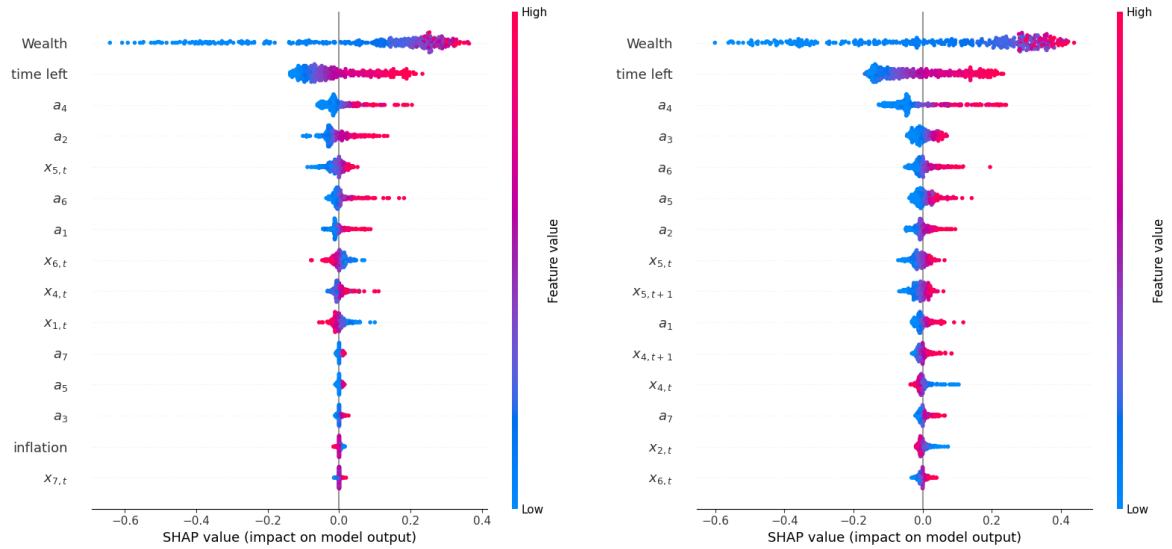
order to omit this bias.

6.5 Interpretability

6.5.1 SHAP

The SHAP values for the critics of the trained model-based and model-free agents are displayed in Figure 6. Both these agents are trained using the goal-only reward function with no discounting, i.e $\gamma = 1$, such that the value of a state can be interpreted as the expected probability of achieving the goal wealth. The derivation for this property can be found in Appendix B. We further incorporate this property by training these agents with a sigmoid output activation function in the critic network. These minor changes have little effect on the final performance and are purely made to aid the interpretability of the SHAP values.

We see that the current wealth appears to be by far the most important feature for both critics when it comes to estimating the state value or the probability of success. The SHAP values indicate that larger values for wealth are associated with higher state values, which make sense since the goal of the agent is to reach a certain amount of wealth. Note that some wealth values that are deemed "low" also have a positive impact on the critic output. This is likely due to the fact that SHAP



(a) Model-free critic

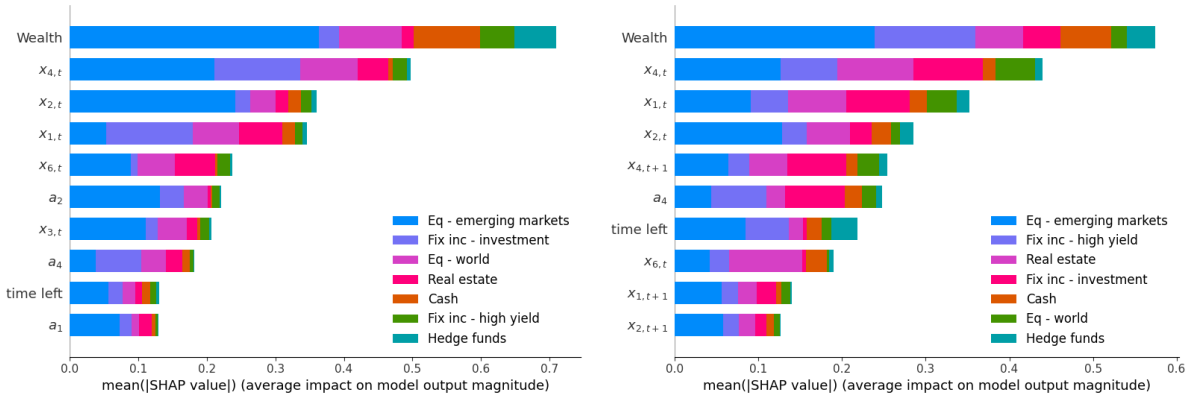
(b) Model-based critic

Figure 6: Estimated SHapley Additive exPlanations (SHAP) values for the critic networks of the model-free and model-based agents.

samples the feature values from a given set of feature data, which in our case consists of a number of observed states during testing. As seen in Table 3, it is not uncommon for the agent to achieve a final wealth that is around a million dollars, hence why some wealth values that are above or close to the goal wealth can still be deemed low. Also, the value contribution of the wealth is likely closely linked to the time left, where a "low" wealth that is relatively large for the current time left can still indicate a larger probability of success. As discussed earlier, this is a known limitation of SHAP. Furthermore, we see that for both critics, the next most important feature is the time left, where large values for time left have a positive impact on the estimated state value. Since more time left also means more time to increase the wealth and reach the goal of the investor, this result is also rather intuitive. We also find that the model-based critic seems to place more importance into the current portfolio weights than the model-free critic, although this difference seems insignificant compared to the feature importance of the current wealth and time left.

Moreover, Figure 7 displays the relative importance of the state elements for the model-free and model-based actor networks, obtained through SHAP. Focusing on the overall feature importance in this analysis, we see that also in the actor networks, the current wealth appears to be the most important feature. Furthermore, we find little difference between the relative feature importance of the model-free and model-based methods. Both actors appear to not be very much influenced by the current portfolio weights or the time left when deciding what action to take, and find the

previous returns of certain assets, such as fixed income investment grade and equity, more important. We do observe that some of the forecasted returns have relatively high feature importance for the model-based actor.



(a) Model-free actor **(b)** Model-based actor

Figure 7: Estimated absolute SHapley Additive exPlanations (SHAP) values for the actor networks of the model-free and model-based agents.

6.5.2 Perfect predictions

Finally, Table 6 displays the performance of the model-based agents whose respective model predictions are replaced with the true returns of the next timestep during testing. These results are obtained using the shaped reward function. We observe that all agents appear to have an increase in performance when given perfect predictions. However, the VAR(3)-P and the N-G-DyBM model see by far the largest increase, achieving a perfect GSR and nearly quadrupling their average final wealth.

Table 6: Model-based PPO performances using different models given perfect predictions

Metrics	AR(1)-H	AR(1)-P	AR(3)-P	VAR(1)-H	VAR(3)-P	N-G-DyBM
GSR	94.0	95.7	95.5	93.1	**100**	**100**
Expected wealth	1600.7	1562.3	1598.8	1294.4	**5158.4**	4766.6
Median wealth	1228.7	1239.9	1244.9	1112.4	**4678.3**	4255.6
$VaR_{0.95}$	373.1	422.8	411.9	351.0	**2038.9**	1914.2
$CVaR_{0.95}$	267.7	306.7	298.0	258.7	**1665.5**	1571.0
Expected returns	7.68	7.61	7.65	7.31	**10.95**	10.74
DDR	1.267	1.362	1.299	1.168	**4.976**	4.674

Note: All the model-based agents are referred to by the prediction model that they use, where N-G-DyBM refers to the nonlinear model used in Yu et al. (2019). "-P" and "-H" denote if the linear model is trained using the pooled training data or the historical data, respectively. The GSR refers to the Goal Success Rate in %. The expected and median wealth and the VaR and $CVaR$ are all estimated at the final state and given in $ [x1000]. Finally, the expected returns and Downside Deviation Ratio (DDR) are averaged over time, with the returns shown in %,

To see if these discrepancies in performance increases are related to the accuracy of the forecasting models, we calculate the RMSEs of the one-step-ahead predictions of the different models in the training set, which are shown in Table 7. We find that all prediction models have a lower RMSE than a simple random walk prediction, with the VAR(3)-P and the N-G-DyBM model achieving the lowest RMSEs of all the models. This result can indicate that the agent places more value in its received forecasts if these forecasts are more accurate.

Table 7: RMSE of different prediction models in the training set

	RW	AR(1)-H	AR(1)-P	AR(3)-P	VAR(1)-H	VAR(3)-P	N-G-DyBM
RMSE	0.206	0.149	0.141	0.141	0.186	**0.135**	0.138

Note: N-G-DyBM refers to the nonlinear model used in the work Yu et al. (2019) and RW denotes a random walk procedure. "-P" and "-H" denote if the linear model is trained using the pooled training data or the historical data, respectively.

7 Conclusion

In this research, we investigate the efficacy of model-based RL in learning how to solve GBWM problems and compare its performance to that of a model-free RL algorithm, for which we use standard PPO. In the model-based framework, we augment the state vector with one-step-ahead forecasts of the asset returns made by autoregressive prediction models. We train and test our RL agents on a set of 15,000 generated economic scenarios, each resembling a possible future investment period. Furthermore, we investigate the influence of the sparsity of rewards on this comparison by specifying both a sparse, goal-only reward function and a dense, shaped reward function.

Moreover, we estimate linear, i.e AR and VAR models, and a nonlinear model used in prior literature, namely the N-G-DyBM, and compare their respective performances. A number of the linear models are estimated using only historical data instead of the scenario training data, such that these models cannot simply approximate the DGP of the scenario set.

Finally, we implement a form of behaviour cloning where expert actions that maximize the returns at each timestep are used to update the actor network of the agent. We apply this in both the model-free and model-based methods and investigate its performance in our GBWM problem setting.

Our main findings are as follows. First, we find that the model-based RL approach consistently achieves a higher goal success rate and lower risk than the model-free approach. This result remains true in both a sparse and a dense reward feedback environment, as well as when additional asset

constraints are enforced. Additionally, we find some evidence that the model-based approach is more sample efficient, especially as the sparsity of the rewards increases. The increased performance of the model-based approach is consistent with Yu et al. (2019) and can likely be attributed to the improved Markov property of the state due to the addition of the forecasts of the asset returns.

Second, we find that the simple linear autoregressive models and the nonlinear model accomplish similar performance gains over the model-free baseline when incorporated into the model-based method. This also holds for linear models fitted only on the historical data, indicating that this model-based approach can already be effective with just the predictions of simple regression models.

Finally, we find that the addition of behaviour cloning in the learning process has a negative effect on the goal success rate in both the model-free and model-based setting. This is contrary to the result of Yu et al. (2019), where behaviour cloning was found to have a positive impact on performance. An explanation for this is the fact that the expert actions in our case are not greedy in the sense that they do not directly maximize the rewards of the agent. This can lead to the agent receiving conflicting feedback signals during the training process, which in turn leads to a decrease in performance.

When it comes to generalizing these results to real-world applications, we highlight the limitations of using synthetic training data. Although our use of simulated scenarios allows us to train and test our agent on a much larger amount of data than what would otherwise be possible, it also brings the possibility of the agent exploiting any deficiencies or false assumptions that might be present in the scenario set. This can in turn inflate the performance of our RL methods during testing. To ensure that the learned strategies also perform well in a real-world setting, a more comprehensive form of backtesting is required. For example, the training and testing of the agents could be performed on two separate scenario sets, where each set is generated using different market assumptions.

Furthermore, an explanation for the fact that the N-G-DyBM is not necessarily the best model in our problem environment, is that it might not translate well to a panel data setting. One of the strengths of this model lies in its ability to capture long-term dependencies in a time series through the eligibility traces and the RNN layer (Dasgupta and Osogami, 2017). It is thus likely better suited for long, non-cross-sectional time series data, instead of the scenario data that we use in our research. Therefore, we recommend that further research within this model-based approach also includes more traditional nonlinear models, such as a Random Forest or an ANN. We also propose incorporating a combination of forecasts instead of only individual model forecasts, as the former

has often been found empirically to result in better forecasts than the latter (Timmermann, 2006).

Finally, as discussed in Henderson et al. (2018), the large amount of randomness during the training process in RL makes it difficult to conclude if differences in performance between algorithms are truly significant. Ideally, all results are averaged over multiple random seeds in order to gain insight into the distribution of the algorithm performances. However, this is computationally expensive, as a single training and testing run can already take up to multiple hours.

References

Aas, K., Jullum, M., & Løland, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, *298*, 103502.

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the twenty-first international conference on Machine learning*, 1.

Aboussalah, A. M., Xu, Z., & Lee, C.-G. (2022). What is the value of the cross-sectional approach to deep reinforcement learning? *Quantitative Finance*, *22*(6), 1091–1111.

Achiam, J. (2018). Spinning up in deep reinforcement learning. *GitHub repository*.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight experience replay. *Advances in Neural Information Processing Systems*, *30*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Blanchett, D. (2015). The value of goals-based financial planning. *Journal of Financial Planning*, *28*(6), 42–50.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., & Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in Neural Information Processing Systems*, *31*.

Cover, T. M. (1991). Universal portfolios. *Mathematical Finance*, *1*(1), 1–29.

Dasa, S. R., & Varmaa, S. (2020). Dynamic goals-based wealth management using reinforcement learning. *Journal Of Investment Management*, *18*(2), 1–20.

Dasgupta, S., & Osogami, T. (2017). Nonlinear dynamic boltzmann machines for time-series prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1).

Dom, S., Huang, D., Nagelkerken, R., & Sassenus, P. (2022). Towards efficient reinforcement learning for goal-based wealth management [unpublished seminar report].

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., & Clune, J. (2019). Go-explore: A new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Feng, Y., et al. (2016). A signal processing perspective on financial engineering. *Foundations and Trends® in Signal Processing*, *9*(1–2), 1–231.

Filos, A. (2019). Reinforcement learning for portfolio management. *arXiv preprint arXiv:1909.09571*.

Fischer, T. G. (2018). *Reinforcement learning in financial markets-a survey* (tech. rep.). FAU Discussion Papers in Economics.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International conference on machine learning*, 1587–1596.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning*, 1861–1870.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. *Proceedings of the AAAI conference on artificial intelligence*, *32*(1).

Henrique, B. M., Sobreiro, V. A., & Kimura, H. (2018). Stock price prediction using support vector regression on daily and up to the minute prices. *The Journal of Finance and Data Science*, *4*(3), 183–201.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade* (pp. 599–619). Springer.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hoi, S. C., Sahoo, D., Lu, J., & Zhao, P. (2021). Online learning: A comprehensive survey. *Neurocomputing*, *459*, 249–289.

Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, *12*.

Liang, Z., Chen, H., Zhu, J., Jiang, K., & Li, Y. (2018). Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, *30*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, 1928–1937.

Moerland, T. M., Broekens, J., & Jonker, C. M. (2020). Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, *12*(4), 875–889.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. *2018 IEEE international conference on robotics and automation (ICRA)*, 6292–6299.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Icml*, *99*, 278–287.

Osogami, T., & Otsuka, M. (2015). Learning dynamic boltzmann machines with spike-timing dependent plasticity. *arXiv preprint arXiv:1509.08634*.

Osogami & Young. (2016). Dynamic boltzmann machine library. *GitHub Repository*. https://github.com/ibm-research-tokyo/dybm.

Ota, K., Oiki, T., Jha, D., Mariyama, T., & Nikovski, D. (2020). Can increasing input dimensionality improve deep reinforcement learning? *International Conference on Machine Learning*, 7424–7433.

Pardo, F., Tavakoli, A., Levdik, V., & Kormushev, P. (2018). Time limits in reinforcement learning. *International Conference on Machine Learning*, 4045–4054.

Qi, X., Luo, Y., Wu, G., Boriboonsomsin, K., & Barth, M. (2019). Deep reinforcement learning enabled self-learning control for energy efficient driving. *Transportation Research Part C: Emerging Technologies*, *99*, 67–81.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1–8.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shapley, L. (1953). A value for n-person games. *Contributions to the Theory of Games*, *28*, 307–317.

Silver, D. (2015). Lectures on reinforcement learning.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Steehouwer, H. (2016). Ortec finance scenario approach. *ORTeC Finance Scenario Department*, 54.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Timmermann, A. (2006). Forecast combinations. *Handbook of Economic Forecasting*, *1*, 135–196.

Tsay, R. S. (2013). *Multivariate time series analysis: With r and financial applications*. John Wiley & Sons.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., & Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Yu, P., Lee, J. S., Kulyatin, I., Shi, Z., & Dasgupta, S. (2019). Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M., & Levine, S. (2019). Solar: Deep structured representations for model-based reinforcement learning. *International Conference on Machine Learning*, 7444–7453.

Acronyms

A2C Asynchronous Advantage Actor-Critic

ANN Artificial Neural Network

BC Behaviour Cloning

CRP Constantly Rebalanced Portfolio

CVaR Conditional Value-at-Risk

DDPG Deep Deterministic Policy Gradient

DDR Downside Deviation Ratio

DSG Dynamic Scenario Generator

GAE General Advantage Estimation

GBWM Goal-Based Wealth Management

GSR Goal Succes Rate

LGF Loss Given Failure

N-G-DyBM Nonlinear Gaussian Dynamic Boltzmann Machine

PPO Proximal Policy Optimization

RL Reinforcement Learning

RMSE Root Mean Squared Error

RNN Recurrent Neural Network

SAC Soft Actor-Critic

SHAP SHapley Additive exPlanations

TD3 Twin-Delayed Deep Deterministic Policy Gradient

TRPO Trust Region Policy Optimization

VaR Value-at-Risk

Appendix A Model-Based PPO pseudo code

Algorithm 1: Model-Based PPO

input: Initialize actor and critic networks μ and V with parameters θ_0 and ϕ_0, respectively. Furthermore, let M denote the number of loops we perform over the entire training set.

for *m = 1, .., M* **do**

Split the training set into batches of size N

for *i = 1, .., N* **do**

Initialize the state with $s_{i,0}$

for *t = 1, .., T* **do**

Forecast the next asset returns $x_{i,t+1}$ with the prediction model and create the augmented state $\tilde{s}_{i,t}$

Use the actor network to obtain the action mean $\mu_\theta(\tilde{s}_{i,t}) \in \mathcal{R}$ and execute $\mathbf{a_t} \sim \mathcal{N}\left(\mu_\theta(\tilde{s}_{i,t}), \sigma^2\right)$

Transition to s_{t+1} and store a_t, R_t, $\tilde{s}_{i,t}$, the log probability of the action and the value estimate $V_\phi(\tilde{s}_t)$ in the memory

end

Estimate the advantages $\hat{A}_{i,t}$ with (5.12) and (5.13) and calculate the returns $G_{i,t}$

Sample mini-batches of size K from the memory without replacement

for *All mini-batches* **do**

Update the actor network by maximizing the PPO objective with respect to θ such that

$$\theta_{new} = \arg\max_\theta \frac{1}{KT} \sum_{j=1}^{K} \sum_{t=1}^{T} \min\left(r_{j,t}(\theta)\hat{A}_{j,t}, \text{clip}\left(r_{j,t}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{j,t}\right),$$

$$\text{with } r_{j,t}(\theta) = \frac{\pi_\theta\left(a_{i,t} \mid \tilde{s}_{j,t}\right)}{\pi_{\theta_{\text{old}}}\left(a_{j,t} \mid \tilde{s}_{j,t}\right)}.$$

Update the critic parameter ϕ by minimizing the least squares loss of the value estimates

$$\phi_{new} = \arg\min_\phi \frac{1}{KT} \sum_{j=1}^{K} \sum_{t=1}^{T} (V(\tilde{s}_{j,t}) - G_{j,t})^2.$$

$\theta \leftarrow \theta_{new}$

$\phi \leftarrow \phi_{new}$

end

clear the memory of the agent

end

end

Appendix B Goal-only value interpretation derivation

$$\begin{aligned} V(s_t) &= \underset{\tau\sim\pi}{E}[G(\tau) \mid s_t] \\ &= E_\pi[\sum_{\tau=t}^{T} \gamma R_\tau \mid s_t], \end{aligned}$$

where setting $\gamma = 1$ and using (5.4) for R_t yields

$$\begin{aligned} V(s_t) &= E_\pi[R_T] = E[0 \cdot \mathbb{1}_{W_T<W_T^*} + \mathbb{1}_{W_T\leq W_T^*} \mid s_t] \\ \therefore V(s_t) &= E_\pi[W_T \leq W_T^* \mid s_t]. \end{aligned} \tag{B.1}$$

Appendix C Hyperparameter tuning

The search spaces and optimal values for the hyperparameters are shown in Table E1. The behaviour cloning hyperparameter is tuned after the optimal values for all other hyperparameters are already found. Note that the tuning is done through a random search, thus not guaranteeing that this specific combination is optimal. Aside from the behaviour cloning hyperparameter, we try a total of 192 different combinations, where we select the combination of hyperparameters that achieves the highest GSR.

Table C1: Hyperparameter search spaces and tuning results

Hyperparameter	Search space	Result
Actor layer width	[64, 128]	128
Critic layer width	[128, 256]	256
Actor learning rate	$[10^{-3}, 10^{-4}, 10^{-5}]$	10^{-4}
Critic learning rate	$[10^{-3}, 10^{-4}, 10^{-5}]$	10^{-4}
Discount factor	[0.99, 1.0]	0.99
scenario batch size	[25, 50, 75, 100]	50
PPO clipping factor	[0.05, 0.1, 0.15, 0.2]	0.1
Adam mini batch size	[32, 64]	32
α_1 (scaling DDR in shaped reward)	[0.01, 0.05, 0.1, 0.5]	0.1
α_2 (scaling LGF in shaped reward)	[0.1, 0.5, 1]	0.5
Behaviour cloning gradient discount	[0.001, 0.0025, 0.01, 0.05, 0.1, 0.2]	0.05

Appendix D VAR lag selection robustness check

To study how the choice of the lag size p in the pooled VAR model affects the performance of the model-based method, we perform a robustness check. Table D1 shows the performance of the model-based method with the pooled VAR model using different lag sizes. We find that the different models lead to relatively similar performance, indicating that this choice of lag size does not greatly affect the outcome of the results.

Table D1: Performance of the model-based method for various lag lengths p used in the VAR(p)-P model

Metrics	VAR(3)-P	VAR(4)-P	VAR(5)-P	VAR(6)-P
GSR	**93.2**	92.3	92.0	92.9
Expected wealth	**1456.3**	1304.2	1289.8	1414.6
Median wealth	**1196.0**	1095.8	1051.5	1116.1
$\text{VaR}_{0.95}$	**348.9**	327.06	326.6	341.8
$\text{CVaR}_{0.90}$	250.3	236.7	239.3	**252.7**
Expected returns	**7.47**	7.21	7.04	7.28
DDR	1.339	1.334	1.357	**1.363**

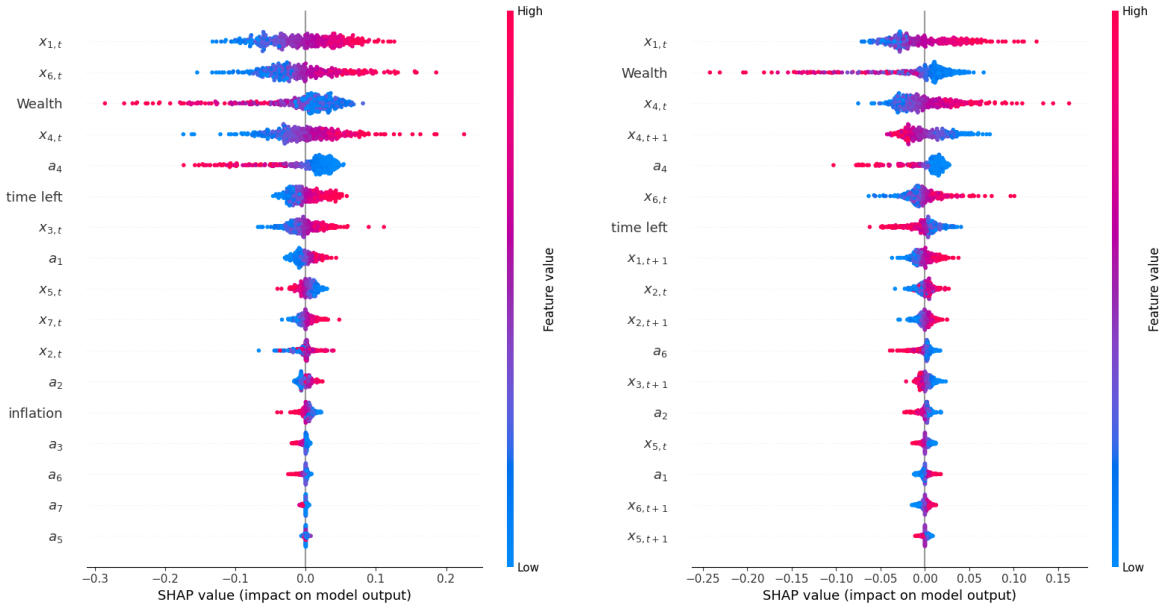
Note: "-P" denotes that the model is estimated on the pooled scenario training data. The GSR refers to the Goal Success Rate in %. The expected and median wealth and the VaR and $CVaR$ are all estimated at the final state and given in $ [x1000]. Finally, the expected returns and Downside Deviation Ratio (DDR) are averaged over time, with the returns shown in %. The results are generated using the shaped reward function.

Appendix E Nonlinear model settings

Table E1: Parameters used in the nonlinear model

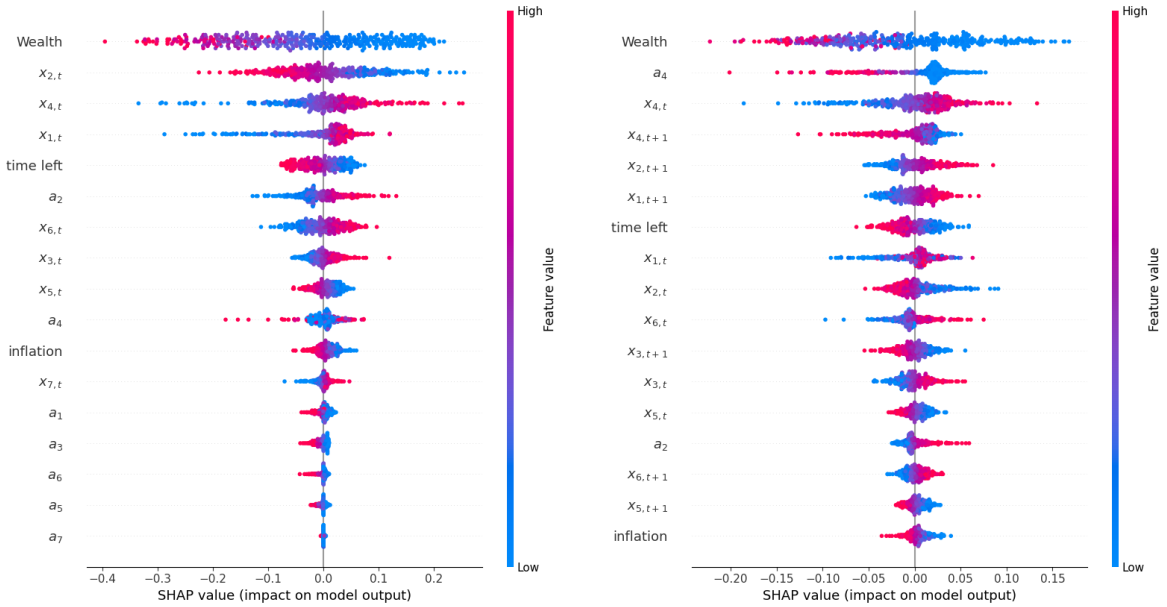
Parameter	Value
Delay	4
Decay rates	[0.1, 0.2, 0.5, 0.8]
RNN sparsity	0.9
RNN spectral radius	0.95
Leak	0.5
Learning rate	10^{-3}
RNN dimension	50

Appendix F Shapley values actor



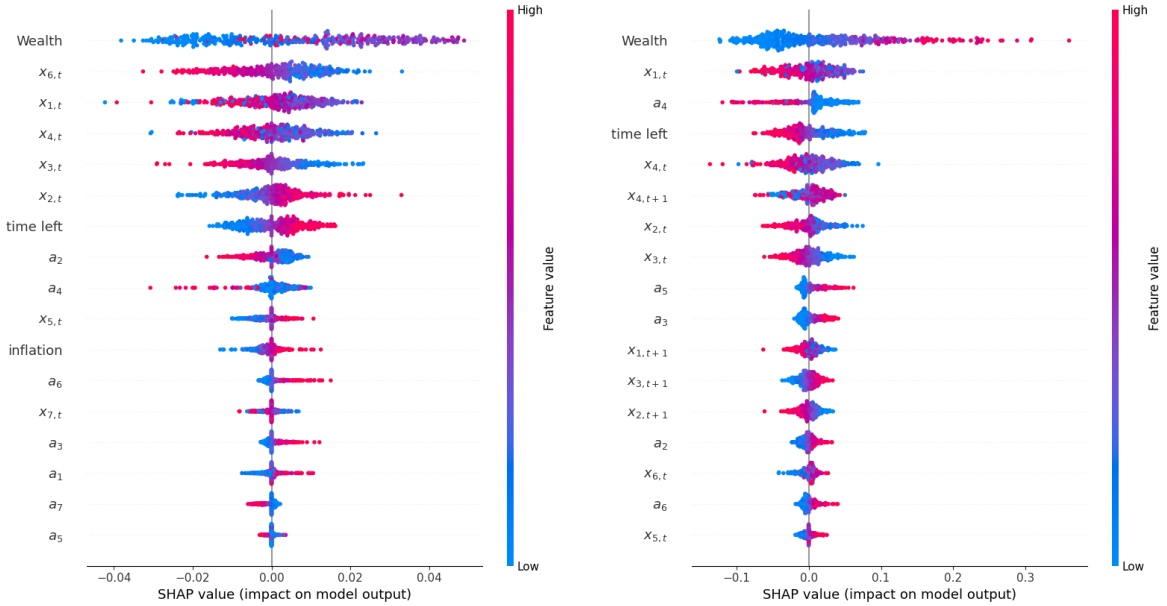
(a) Model-free actor **(b)** Model-based actor

Figure 8: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in equity (world) by the model-free and model-based actors.



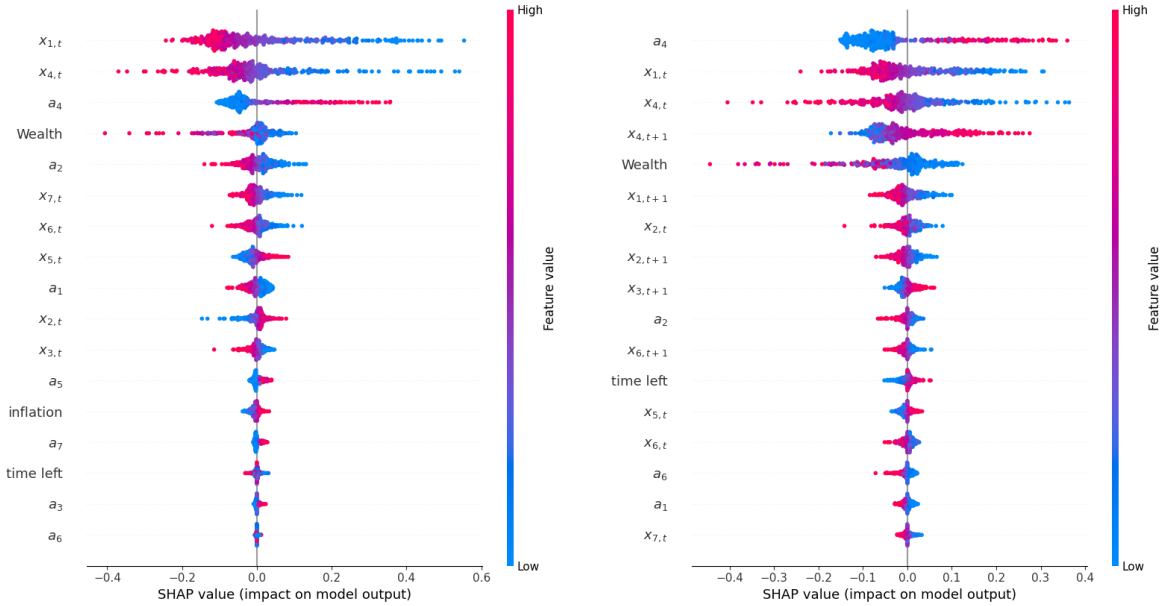
(a) Model-free actor **(b)** Model-based actor

Figure 9: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in equity (emerging markets) by the model-free and model-based actors.



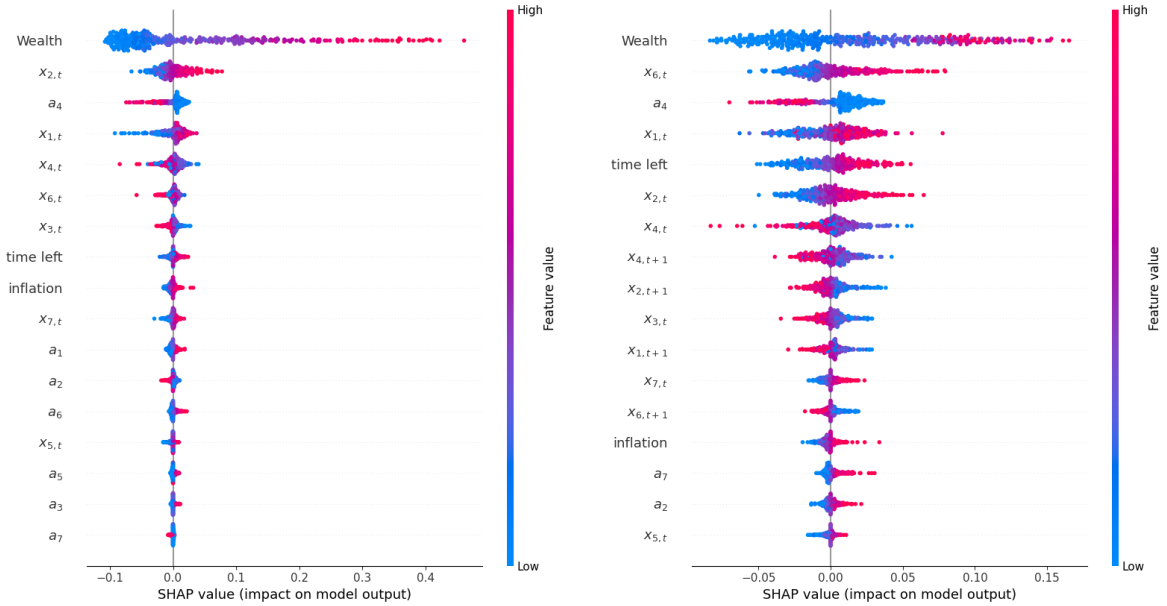
(a) Model-free actor **(b)** Model-based actor

Figure 10: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in fixed income (high yield) by the model-free and model-based actors.



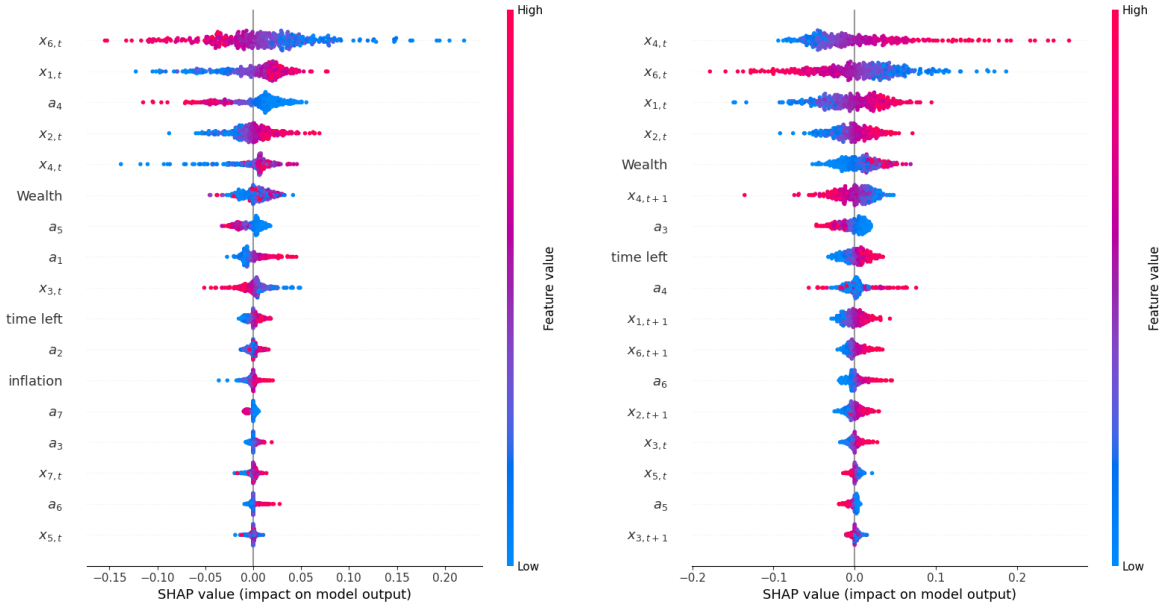
(a) Model-free actor **(b)** Model-based actor

Figure 11: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in fixed income (investment grade) by the model-free and model-based actors.



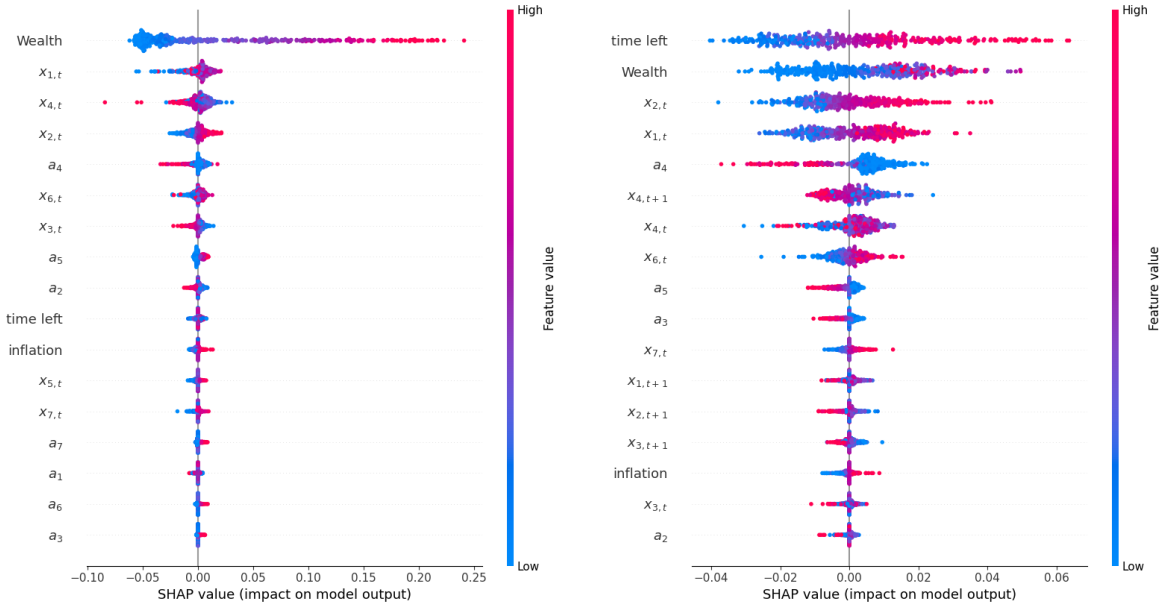
(a) Model-free actor **(b)** Model-based actor

Figure 12: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in cash by the model-free and model-based actors.



(a) Model-free actor **(b)** Model-based actor

Figure 13: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in real estate (indirect) by the model-free and model-based actors.



(a) Model-free actor **(b)** Model-based actor

Figure 14: Estimated SHapley Additive exPlanations (SHAP) values for the action weight in hedge funds by the model-free and model-based actors.

Appendix G Backtest results

Table G1: Performances of the different trading strategies when tested on the historical data

Metrics	CRP	Model-free	Model-based
Final wealth	373.6	428.1	**624.8**
Average returns	5.25	6.95	**7.81**
DDR	1.072	0.723	**1.235**
volatility	**9.20**	17.9	15.3

Note: The reinforcement learning methods are trained using the shaped reward function and the model-based method uses the pooled VAR(3) model. The Constantly Rebalanced Portfolio (CRP) strategy is shown as a benchmark. We backtest the different strategies in the investment period spanning from 1994 to 2021, allowing for an annual reconstruction of the portfolio. Each strategy starts with $100,000 and the final wealth is given in $ [x1000]. Furthermore, the average returns and volatility are shown in %.