ERASMUS UNIVERSITEIT ROTTERDAM

Deep Reinforcement Learning Approach for Portfolio Management Applications with a Multivariate Reward Function

Master Thesis Econometrics and Management Science - Quantitative Finance

Emile Eijgenraam (484250)

Supervisor: Prof. Dr. Dick van Dijk

Second Assessor: Dr. Rasmus Lönn

April 26, 2024

Abstract

In this research, we propose various deep reinforcement learning (DRL) models that can be used for investment strategies with multiple objectives, namely portfolio returns, volatility and ESG performance. We find that the model that is used for an investment strategy with multiple objectives, which is an ensemble of individual DRL models that each focus on a single objective, performs best overall in terms of reward during the test period. One of these individual DRL models that can be used for an investment strategy to minimise portfolio volatility performs significantly better than the majority of DRL models in minimising portfolio volatility. This is because of the transformation of the input data for the neural network from historical daily returns to EWMA-based volatility estimates per asset. This results in a more intuitive relationship between model input and output, which is a solution to a problem faced in literature. However, the GMV portfolio still yields a significantly lower portfolio volatility during the test period. Finally, we use a hedging strategy to complement the DRL models to significantly reduce the portfolio volatility of most of the models.

Contents

1 Introduction

Over the last few years, numerous events had a major impact on financial markets. The COVID-19 pandemic, the Ukraine war and highly increasing inflation rates were some of the largest drivers of volatility in the stock and bond markets. Next to the challenge of maintaining steady returns during recent times of high volatility, there is increasing pressure from society and governments to decrease the investments made in companies that do not have high sustainability ambitions. Together, it becomes increasingly challenging for institutional investors like pension funds to perform simultaneously well in terms of portfolio returns, risk and sustainability.

In this paper, we formulate several models that can be used for investment strategies with multiple objectives, namely portfolio returns, risk and (exposure to) ESG factors. For the models we formulate, an investor can state her priorities concerning each portfolio performance metric by explicitly assigning weights to each metric. Subsequently, the models aim to incorporate the weights in the training process by assigning the highest priority to the metrics with the highest weight. As such, our models have the potential to be integrated into the investment strategies of institutional investors. They can address challenges in defining strategies that yield high portfolio returns and low risk while improving ESG performance, in line with the global trend of reducing investments in sectors like oil and gas. Therefore our research is relevant for institutional investors who can use this paper as a foundation to be used for investment strategies with multiple objectives.

For one of the portfolio performance metrics, namely portfolio volatility, the Global Minimum Variance (GMV) portfolio can be used to form a portfolio of stocks that aims to minimise the portfolio volatility, as shown by Merton (1972). For another portfolio performance metric, namely portfolio returns, several models can be used for an investment strategy that aims to maximise portfolio returns. In recent research, the success of machine learning-based models is shown when it is used for investment strategies that aim to maximise portfolio returns, as shown by for instance Jiang et al. (2017) and Liang et al. (2018).

Jiang et al. (2017) and Liang et al. (2018) show promising results in terms of portfolio returns by using a deep reinforcement learning (DRL) model for an investment strategy that aims to maximise portfolio returns, where an agent (which can be seen as an investor) is rewarded if her actions (selection of portfolio weights) yield high returns. In their research, a neural network (the 'deep' component of DRL) is trained to obtain the output (portfolio weights) that yields the highest portfolio returns based on the input (historical returns data). Instead of solely focusing on returns, we formulate several DRL models that are used for investment strategies with multiple objectives, namely portfolio returns, volatility and ESG performance. Since this has never been done in any previous research, our paper is also relevant from an academic perspective to investigate if DRL models are also suitable to be used for investment strategies, with multiple objectives and objectives that are not related to portfolio returns.

In this research, we formulate DRL models with a multi-dimensional reward function. That is, we combine portfolio return, volatility and ESG performance into one reward function. Each metric is given a weight indicating the priority an investor assigns to each. We combine daily returns of S&P 500-listed companies with daily bond returns to form ten ETFs to form a balanced portfolio. To translate

the environment (historical daily stock returns of the portfolio) to an action (portfolio weights assigned to each asset) we use a neural network. We formulate three different models, where the first DRL model calculates the gradient during training over the entire reward function. The second DRL model calculates the gradient of the reward function during training with respect to one of the three metrics per training iteration with probability in line with the weight assigned by the agent, leaving out the gradient components of the other two metrics for a specific training iteration. The third model is an ensemble of individual DRL models where each model can be used for an investment strategy with a singular objective, namely portfolio returns, volatility and ESG performance. Based on the weights assigned by the agent, the output of the three DRL models is combined into a single output.

Filos (2019) describes how research in portfolio management has shifted from traditional methods, such as the Markowitz (1991) model for mean-variance portfolios, to more machine learning-based approaches such as DRL models that can be used for investment strategies that aim to maximise portfolio returns. Using a DRL model that is used for an investment strategy that aims to maximise portfolio returns, they outperform the Markowitz (1991) mean-variance portfolio benchmark in terms of Sharpe ratio and returns. Jiang et al. (2017) propose a DRL model to form a successful investment strategy in a high-frequency trading environment that aims to maximise portfolio returns. Using this approach, they outperform a range of machine learning-based models in terms of portfolio returns that are used for an investment strategy that aims to maximise portfolio returns.

We build further on this research by following the steps taken to form the DRL model and in terms of notation, but with a different (multivariate) reward function. Jiang et al. (2017) propose a so-called 'ensemble of identical independent estimators' (EIIE) neural network structure in their DRL model. This means that the historical data (input of the neural network) of each asset flows independently through the neural network. Next to this approach, we evaluate a different approach that allows for interaction between the assets in the neural network, by combining the historical data of the considered assets in a matrix instead of a vector as input to the neural network. This intuition is based on the GMV portfolio, through the off-diagonal elements in the inverse covariance matrix, meaning that the dependence between assets can be relevant to minimise the portfolio volatility.

Liang et al. (2018) also propose a range of DRL models that can be used for an investment strategy aiming to maximise portfolio returns. Instead of using high-frequency data, they use daily returns from the Chinese stock market. While they mainly focus on portfolio returns, they also mention a reward function that includes both the portfolio return and volatility. However, they conclude that their approach does not work for multivariate reward functions. Volatility is added as a penalty term to the reward function, and only the DRL model with a very small penalty term shows some form of improvement in terms of portfolio returns, when it is used for an investment strategy aiming to maximise portfolio returns.

Liang et al. (2018) conclude that the addition of the volatility term makes the gradient of the reward function too complex to yield high portfolio returns during the test period and reduce portfolio volatility. We believe this is not the main reason, but rather that finding the relationship between the input of the neural network (historical daily returns) and the output (the portfolio weights) that aim to minimise (or in the case of Liang et al. (2018) reduce) the portfolio volatility is too complex, in the case it is used

for an investment strategy that aims to maximise returns. To handle this issue, we formulate one model that is an ensemble of individual DRL models, where one of the individual DRL models is used for an investment strategy aiming to minimise the portfolio volatility, but where the neural network input is historical EWMA-based volatilities for all assets, instead of historical daily returns. We mainly build further on the neural network architecture of Liang et al. (2018), which contains a few features that aim to solve the vanishing gradient problem during the neural network training procedure.

For a variety of weight combinations assigned by the agent to portfolio returns, volatility, and ESG performance, we analyse the performance of each of the three DRL models that are used for investment strategies with both singular and multiple objectives. We conclude that the model that is used for an investment strategy with multiple objectives, which is an ensemble of individual DRL models performs best in general in terms of average reward during the test period. We conclude that this is mainly due to the high performance in the corresponding portfolio performance metric of each of the individual DRL models which are used for investment strategies with a single objective.

In particular, the DRL model which is part of the model that is an ensemble of DRL models, which can be used for an investment strategy aiming to minimise portfolio volatility performs better compared to other DRL models in minimising the portfolio volatility. We believe this is due to the transformation that we apply to the input of the neural network, which originally is a matrix of historical daily returns, but which we transform to historical EWMA-based volatility estimates for all assets for this model. This leads to a less complex task for the neural network to find a relationship between the input to the neural network and the portfolio weights (the output of the neural network) that aim to minimise portfolio volatility. However, in terms of minimising portfolio volatility, the GMV portfolio still significantly outperforms the DRL models that are used for investment strategies that aim to minimise portfolio volatility. Despite the improvement following the transformation of the input data, directly optimising through the GMV portfolio on a daily basis seems still more effective than finding the patterns in the data to assign the highest portfolio weights to the least volatile assets based on historical EWMA volatilities data.

We use a Bayesian hyperparameter optimisation procedure, instead of a traditional grid search to find the optimal set of hyperparameter values that aim to maximise the reward value for each DRL model. Using the Bayesian hyperparameter optimisation procedure as shown by Wu et al. (2019), we can efficiently optimise a wide range of hyperparameters, since a grid search, for example, would require evaluating all possible combinations of hyperparameter values we consider. Since our model contains seven hyperparameters, this is computationally not feasible. We make use of SURF Research Cloud[1] to make use of parallel training with a budget of 15,000 SBU (computing hours) to perform the Bayesian hyperparameter optimisation and the training procedure for all the considered DRL models.

Aiming to limit our lower tail risk during extreme downside market circumstances and to circumvent the challenge of training DRL models for such scenarios with limited available data, we propose a hedging strategy during extreme downside market circumstances. Further elaboration on this strategy is provided later in this research. We use copulas to estimate the lower tail dependencies of all the pairs of ETFs we use in our portfolio to form a hedging strategy. By complementing our DRL models, which are used for

[1] Available through SURF Research Cloud

investment strategies with singular and multiple objectives, with a hedging strategy, we significantly reduce the portfolio volatility for most weight combinations for the DRL models. However, this is relatively expensive in the sense that the portfolio returns and ESG performance suffer significantly to realise this reduction.

Our research contributes to the current literature by forming a multi-dimensional reward function for the proposed DRL models, instead of a 'singular' returns-driven reward function, by adding portfolio volatility and ESG performance as extra reward function components. Using our approach, an agent can explicitly assign weights to each of the three portfolio performance metrics that represent her preferences. The weights are used during the training process to train a DRL model that can be used for an investment strategy with singular or multiple objectives, that aim to reflect the agent's priorities in terms of performance for the corresponding metrics.

In addition, we use a hedging strategy to complement our DRL models that aim to limit the lower tail risk during extreme downside market circumstances. This approach significantly reduces the portfolio volatility during the testing period for most models that are used for investment strategies with singular and multiple objectives. Finally, we propose a solution to one of the issues faced by Liang et al. (2018) that is related to the complexity of the gradient when a volatility component is added to the reward function. To do so, we transform the input data of the neural network from historical daily returns to historical EWMA-based volatilities for all assets. This leads to a more stable training process due to the less complex relation that the neural network aims to find between the input data and the output of portfolio weights that aim to minimise the portfolio volatility, when the DRL model is used for an investment strategy that aims to minimise portfolio volatility.

This research is structured as follows: in Section 2 we discuss the data that is used for this research and in Section 3 we discuss the methods we use and the details of each of the proposed DRL models. In Section 4 we discuss the results of the Bayesian hyperparameter optimisation procedure, analyse the performances of all models when they are used for investment strategies with both singular and multiple objectives and show the impact of using a complementary hedging strategy to reduce the portfolio volatility. Finally, We conclude the research in Section 5.

2 Data

In this section, we discuss the data that is used to train and test our DRL models, which consists of historical daily stocks and bonds data that are available through Yahoo Finance[2]. Liang et al. (2018) show that daily stock returns data is a suitable sampling frequency to use for DRL research in the context of portfolio management applications, which we extend with the inclusion of daily bond returns. We choose to not go beyond ten assets in our portfolio, as this would be potentially problematic to optimise efficiently due to the curse of dimensionality. Still, ten is larger than the paper we use the most for this research, as Liang et al. (2018) consider only five stocks. Using the stocks and bonds data, we form ten ETFs, which we explain in more detail in the remainder of this section. Compared to institutional

[2]Available through Yahoo Finance

investors, a portfolio of ten assets is likely relatively small, but given that we explore a relatively new research field, further research can be conducted to identify the maximum number of assets for which the proposed DRL models can be trained and used for an investment strategy with portfolio returns, volatility and ESG performance as objectives. The data ranges from 01/01/1992 until 31/12/2021, which includes both crises and stock rallies. The data contains crises such as the dot-com bubble burst in the late 90s and early 2000s, 9/11, the global financial crisis of 2008 and the COVID-19 crisis.

Regarding the stocks data, we use the historical daily returns of 45 US-listed stocks from the S&P 500 exchange, across a range of industries to form a balanced list of assets that include more and less volatile stocks. The stocks are selected from nine different industries, where we include five stocks from each industry, creating a diversified portfolio of stocks. Since the stocks have existed for at least 30 years, there is some 'survivorship bias', which is a disadvantage of using daily returns data to train the DRL models, since our models require a significant amount of data to train properly, hence the requirement that a company has been listed for a longer period of time to yield sufficient training data.

We also include a range of US bond yields in our data set that are typically also used by institutional investors. This approach is similar to Zhang et al. (2020), who form a DRL model that is used for an investment strategy that aims to maximise returns. Therefore, we include a range of US bonds, namely the 5, 10 and 30-year Treasury Yield. Using the stocks from the nine industries and the bonds, we form ten 'ETFs' based on the ten equally weighted portfolios for each of the ten groups of stocks and bonds, which is the portfolio of assets we will consider for this research. Making use of ETFs instead of single stocks or bonds will reduce some idiosyncratic risk, making the portfolio less volatile from the start of the training process compared to single stocks and bonds.

In addition, the S&P 500 exchange provides environmental, social and governance (ESG) scores for each company used for this research[3]. Using these scores, we can train DRL models that can be used for investment strategies with multiple objectives, where one of the objectives is to maximise the portfolio ESG performance. The ESG scores range from 0 (ignoring ESG) to 100 (highest possible ESG performance) and are available as a single score using the 2023 ESG measurement methodology used by S&P Global. As the amount of ESG data does not match the stock returns data, we assume that the ESG scores for each company have been constant between 01/01/1992 and 31/12/2021. Since the portfolio returns and volatility components are time-varying in our reward function and the ESG scores are constant, we expect the ESG performance component of the reward function to be potentially quite dominant in the training procedure. This is due to the ESG component of the gradient that aims to push the neural network weights in the same direction every training iteration to assign higher weights (output of the neural network) to the assets with the highest ESG scores.

To match the dimensions, we assume that bonds have an ESG score of 50, making them 'average performers' in terms of ESG. This is quite a strong assumption, but due to the lack of ESG bond data, we assume average ESG performance for bonds. In case the actual ESG score for bonds is lower than 50, we overestimate the overall actual ESG score of the portfolio and vice versa. In case more ESG bond data becomes available, this assumption can be assessed and changed to actual data in future research. Then,

[3]Available through S&P 500 ESG scores 2023

we form 'ESG ETFs' by taking the equally weighted sum over the ESG scores for all nine industries and bonds. We form the ETFs (for the stocks, bonds and ESG scores) ourselves, as the more common indices and portfolios that are available through for example Kenneth French do not report ESG scores. For all the ESG ETFs, we standardise the data by subtracting the mean and dividing by the standard deviation, which makes the scale of the ESG data more useful to apply in machine learning models due to its scale.

Within the dataset of 30 years, the amount of trading days of bonds is 26 days less than stocks. To cope with this mismatch, we define the returns of bonds as 0% during the days in which there is no bond trading. In this way, every asset has the same number of trading days and makes combining stocks and bonds in a portfolio possible.

In this research, we transform our returns to log-returns, similar to Jiang et al. (2017) and Liang et al. (2018). We show summary statistics for the ten ETFs in Table 1. Due to some crises that occur within our time horizon, some extreme values can be found. Interestingly, the average log-returns of the three bonds show the highest kurtosis. The bonds in the bond ETF do not tend to show a high correlation during extreme events between each other, such as on 29/09/2008. During the start of the global financial crisis of 2008, the 30-year Treasury Yield went up around 34%, whereas the 5-year Treasury Yield went down around 5%. Similar examples occur throughout the time window. Also, the Energy ETF is not the worst-performing asset in terms of ESG performance, but the Financial ETF performs the worst. This might be explained by the fact that ESG stands for environmental, social and governance, meaning that the companies within the Financial ETF might perform worse in terms of social and governance aspects or invest heavily in unsustainable companies, related to the public criticism for example that ING (February 2024) faced.

Table 1: Summary statistics for the log-returns and ESG scores of the formed ETFs. 'E&T' stands for Entertainment and Telecom, 'An. mean (pp)' stands for annualised mean returns in percentage points and 'J.B. stat' stands for the Jarque-Bera test statistic to test normality. The data runs from 01/01/1992 until 31/12/2021

	Tech	Industrial	Financial	Energy	Health	Consumer	E&T	Materials	Retail	Bonds
An. mean (pp)	21.9	12.7	12.3	12.5	11.6	9.58	11.6	12.2	16.3	-4.05
Minimum	-0.136	-0.165	-0.194	-0.175	-0.102	-0.078	-0.108	-0.140	-0.160	-0.289
Maximum	0.149	0.143	0.176	0.217	0.082	0.087	0.138	0.122	0.128	0.344
ESG score	0.361	-0.214	-1.24	-0.630	1.76	1.42	-1.14	0.170	-0.630	0.137
Std. Dev.	0.019	0.015	0.018	0.016	0.012	0.010	0.013	0.014	0.014	0.022
Skewness	0.007	-0.367	0.131	-0.261	-0.114	0.123	-0.014	-0.191	-0.051	0.122
Kurtosis	4.413	10.614	16.835	20.497	5.092	6.381	7.770	8.105	7.22	28.370
J.B. stat.	*629	*18424	*60290	*96486	*1394	*3618	*7165	*8252	*5619	*202677

$^*p < 0.01$

Due to the high kurtosis of all ETFs, which indicate fat tails in the distributions of the log-returns, we can reject the Jarque-Bera test for normality at a 1% significance level. Also, we observe that overall during the considered period, the stocks within the Tech and Retail ETFs outperform the other industries in terms of average annualised returns as percentage points.

In addition, we show how the ETFs are related through Kendall's τ and Spearman correlation values in Table 2. As expected, since bonds are a different asset type and are often chosen to diversify the

portfolio to include assets that are not stocks, they are the least related to the other ETFs.

Table 2: Kendall's τ values (lower-left) and Spearman correlations (upper-right) for the log-returns of the ETFs. The data runs from 01/01/1992 until 31/12/2021

	Tech	Industrial	Financial	Energy	Health	Consumer	E&T	Materials	Retail	Bonds
Tech	1.000	0.532	0.499	0.411	0.419	0.342	0.537	0.460	0.509	0.206
Industrial	0.338	1.000	0.695	0.591	0.532	0.493	0.645	0.662	0.616	0.306
Financial	0.323	0.455	1.000	0.544	0.523	0.465	0.634	0.586	0.600	0.311
Energy	0.248	0.350	0.321	1.000	0.450	0.423	0.534	0.612	0.437	0.248
Health	0.275	0.332	0.355	0.257	1.000	0.560	0.551	0.452	0.582	0.183
Consumer	0.226	0.300	0.314	0.247	0.354	1.000	0.516	0.461	0.494	0.139
E&T	0.338	0.393	0.410	0.298	0.354	0.334	1.000	0.562	0.619	0.231
Materials	0.290	0.408	0.350	0.363	0.267	0.266	0.328	1.000	0.516	0.233
Retail	0.325	0.382	0.400	0.255	0.378	0.317	0.390	0.306	1.000	0.206
Bonds	0.122	0.148	0.151	0.114	0.086	0.046	0.103	0.115	0.106	1.000

As shown in Table 1, each industry showcases a high kurtosis. These fat tails result in extreme values, that are, generally speaking, hard to predict. We define extreme downside market events as the trading days that yield the 5% lowest portfolio returns during a specific time horizon, using a DRL model that is used for an investment strategy with singular or multiple objectives. To limit the downside risk associated with the extreme downside market events, we define a hedging strategy to complement our DRL models, which are used for an investment strategy with singular or multiple objectives, which we elaborate further on in Section 3.2. We train our DRL models without the 5% lowest daily returns data in the case that we complement the DRL models with a hedging strategy. We will assess both the performances of the DRL models that are used for investment strategies with singular and multiple objectives that use the entire training data set, as well as the ones that are trained without the 5% lowest returns and that are complemented by the hedging strategy.

3 Methodology

3.1 DRL Model

3.1.1 Model setup

In a DRL model, an agent is rewarded after her actions following a given state (environment). In this setting, the agent can be seen as a portfolio manager who invests in ETFs over a period of time. After each action (finding a set of portfolio weights), the agent is rewarded following a reward function with multiple metrics, which we elaborate on in the next section. Similar to Jiang et al. (2017), the environment (state) our agent operates in is driven by the returns of the ten assets, comprising our defined ETFs. In addition, we add the (constant) ESG scores of each asset to our environment, denoted by E.

Since our dataset of returns is vast and returns before a certain threshold are likely to contain little insights for today's actions of the agent, we follow Jiang et al. (2017) in defining a 'price tensor' that limits the number of trading days included in our model, through the input of the neural network. Our

price tensor is a rolling window of the past N^H historical trading days for all assets. We denote our price tensor as V_t, which includes all returns for each asset from $t - N^H$ until $t - 1$. For each time t, our agent's environment is given through the state $S_t = (V_t, E)$, where V_t is the input to the DRL model at time t and E is used to train and evaluate the ESG component of the reward function. Given the state, the agent chooses an action a_t at time t that aims to maximise the reward function during a certain test period, based on the output of the DRL model. In our case, the output of the DRL model is the output of the neural network at time t, which is equal to a_t, which are the portfolio weights that the agent chooses at time t.

Due to our neural network architecture, which we discuss further in Section 3.1.4, we do not allow for short-selling and the sum of the portfolio weights a_t at time t is equal to one (the entire budget is always used). To reduce the transaction costs that the agent faces due to the daily rebalancing, additional restrictions could be incorporated to add a restriction between the difference between a_{t-1} and a_t. For example, there could be a maximum on the absolute or relative difference between the portfolio weights in a_t compared to a_{t-1} for all assets. Since transaction costs are not part of the main goal of this research, we do not explore such restrictions, but could be analysed in future research.

3.1.2 Reward function

The reward function measures the average reward throughout the test period, from $t = TP_{begin}$ until $t = TP_{end}$, where the total number of test days is denoted as T_{TP}, a similar intuition as used by Jiang et al. (2017). Since we aim to incorporate multiple portfolio metrics in our reward function, we define our reward function as multivariate. First, the portfolio's return r_t at time t should be as high as possible and is calculated by $w_t R_t$, where R_t denotes the vector of returns for each asset at time t. Second, our portfolio volatility should be as low as possible, meaning we subtract the estimated volatility $\widehat{Vol}_t$ as input in our reward function. In Section 3.1.3, we define how $\widehat{Vol}_t$ is derived. Last, we include the weighted ESG score E_t of our portfolio at time t, calculated by $w_t E$, which should be as high as possible. During training, we standardise the input for each of the three gradients (the R_t, $\widehat{Vol}_t$ and E components) by subtracting the mean and dividing by the standard deviation in each training iteration, as it is not desired that one gives a disproportional high contribution to the training procedure of the DRL model, only because of a different measurement unit.

Since we are combining the three portfolio performance metrics, we assign weights to each metric by defining $0 < w_R < 1$ as the relative importance of high portfolio returns, $0 < w_V < 1$ as the relative importance of low portfolio volatility and finally $0 < w_E < 1$ as the relative importance of a high portfolio ESG performance. In addition, we set the constraint of $w_R + w_V + w_E = 1$. In this way, an agent can adjust the model to be completely designed to her preferences. In Section 3.1.8 we show the set of weights assigned per metric that we evaluate and how we evaluate the reward function for each combination of weights. Together, the reward over a given test period from $t = TP_{begin}$ until $t = TP_{end}$ is given by the reward function:

$$R = \frac{1}{T_{TP}} \left(\sum_{t=TP_{begin}}^{TP_{end}} (w_R r_t - w_V \widehat{Vol}_t + w_E E_t) \right) \tag{1}$$

In the remainder of this paper, if we refer to a certain 'DRL model', it refers to a DRL model that is used for an investment strategy with multiple or singular objectives corresponding to the weights that are assigned by the agent through the reward function. The output of the DRL model is a vector of portfolio weights that are invested in the corresponding ten ETFs to form a portfolio. In addition, if the agent sets for example $w_V > 0$, the objective of the agent is to minimise the portfolio volatility during the test period if the DRL model is used for an investment strategy that aims to minimise portfolio volatility (and potentially maximise portfolio returns and ESG performance). We denote this as "the agent aims to minimise portfolio volatility" or "the DRL model aims to minimise portfolio volatility" for convenience.

3.1.3 Reward function extension: volatility

One of the components of our reward function revolves around portfolio volatility. A possible way to derive this is to compute the sample covariance matrix and use this to calculate the portfolio volatility. However, this method assumes recent returns to be as equally important as more dated returns, which may not be realistic given the fact that the agent rebalances the portfolio daily.

To address this limitation, we incorporate the Exponentially Weighted Moving Average (EWMA) in our volatility model, similar to Bollen (2015). By assigning higher weights to recent data points, EWMA accounts more for recent changes in market conditions, making it a good fit for real-time volatility assessment.

Bollen (2015) show that the EWMA model can be written as follows, where N^H denotes the amount of past daily returns of stock j that are used to estimate the volatility, which is available in the price tensor as input to the model. The importance of each past daily return included decays by factor λ. We denote the squared returns of asset j at time $t-i$ as $R^2_{j,t-i}$. We set the value of λ to 0.94, the RiskMetrics decay factor as described by Mina et al. (2001), who show that this yields favourable results in terms of root mean squared error (RMSE) and mean absolute error (MAE). Since we do not take the sum until infinity, we acknowledge that this approach is slightly biased, but since N^H ranges between 45 and 55 (explained in Section 3.1.9) this bias is limited. Together, the volatility estimator of asset j for time t is:

$$\sigma^2_{j,t} = (1-\lambda) \sum_{i=1}^{N^H} \lambda^i R^2_{j,t-i} \tag{2}$$

This procedure is done for all ten assets in our portfolio. In addition, we calculate the sample covariances of all pairs of assets within our portfolio using the N^H past daily returns of each asset. Together with the estimated variances of all assets, we form the estimated covariance matrix at time t as $\widehat{\Sigma}_t$. Since the outcome of our proposed DRL model will be the new portfolio weights w_t, the estimated volatility of our portfolio at time t becomes:

$$\widehat{Vol}_t = \sqrt{w_t' \widehat{\Sigma}_t w_t} \tag{3}$$

Using the proposed volatility model, we can add volatility to our reward function, described in Section 3.1.2. regarding our reward function setup.

3.1.4 Policy function specifications

To move from state to action, we form a policy function π_Θ, which includes model parameters Θ (Jiang et al. (2017)). In the case of a DRL model, Θ consists of the neural network parameters. The policy function generates the action of the agent:

$$a_t = \pi_\Theta(S_t) \tag{4}$$

We aim to design a policy function that maximises the expected reward of the agent during the testing period, which is equivalent to:

$$\pi^*_\Theta = \underset{\pi_\Theta}{\operatorname{argmax}}\, E[R|\pi_\Theta] \tag{5}$$

In our research, we form a neural network to form our policy function, hence the distinction between RL and DRL. Instead of a neural network, there are numerous other options to define a policy function. For example, a very simple policy function would be to assign a portfolio weight at time t of one to the asset that yielded the highest return at time $t-1$. As described in the previous section, the input of our model is the price tensor V_t, the output should be a vector of positive weights summing up to unity. After training our neural network over the training data, we test the performance of our DRL model during the test period. By using a neural network in our DRL model, we can estimate the relationship between the price tensor and the portfolio weights that yield the highest reward.

A significant difference between our proposed neural network architecture and the architecture used by Jiang et al. (2017) and Liang et al. (2018) is that we take the entire price tensor V_t at day t as input to our neural network. For the sake of training efficiency, Jiang et al. (2017) and Liang et al. (2018) consider a so-called 'ensemble of identical independent estimators' (EIIE). This means that the historical data of each asset is passed independently of all other assets through a neural network, where all assets share the same model parameters.

For a DRL model using EIIE neural network structure, Jiang et al. (2017) show that a single neural network is trained, that has an output of a single 'voting score' (the higher the voting score, the higher the portfolio weight the agent should assign to a certain asset) based on the price tensor of a single asset. The price tensors of all assets are passed independently through the neural network, collecting all the voting scores for all assets. Then, all the voting scores are combined in a softmax output function to create an output that is a vector of weights that sum to unity, which the agent can use to form an investment strategy to maximise returns. To propose a DRL model that aims to maximise portfolio returns or ESG performance or both, we believe the EIIE approach is quite intuitive, based on the idea that the voting score of a single asset does not necessarily need the price tensors of other assets to optimise these two metrics in a DRL model.

If the data of all assets is passed through the neural network independently for each asset, we believe it might be difficult to find cross relations that can be used in case the agent aims to minimise the

portfolio volatility. In the GMV portfolio, the covariance matrix Σ_t at time t, which includes the off-diagonal covariance values, is crucial to derive portfolio weights that aim to minimise portfolio volatility. If we translate this intuition to our neural network architecture, we believe the neural network should be able to find relations between assets as well and use this to train a DRL model that aims to minimise portfolio volatility. Since we consider a portfolio of ten assets, we believe that this approach is feasible, however, if for example twenty assets are considered, the number of neural network parameters is likely to increase exponentially which can make the training of the neural network too complex. Jiang et al. (2017) conclude that the scalability of a DRL model with an EIIE neural network is another advantage, since all assets share the same neural network, meaning that it can be used by an agent that considers a much larger number of assets similar to the scale of investments of an institutional investor. Due to this reasoning, we consider both our proposed and the EIIE neural network approach.

Jiang et al. (2017) consider three different types of hidden layers for their neural network models within their policy functions: the convolutional neural network (CNN), the recurrent neural network (RNN) and the long short-term memory (LSTM). They conclude that CNN and RNN outperform LSTM, possibly because LSTM is less capable of recognising repetitive patterns. Instead of choosing one type of layer, we construct a neural network consisting of several different layers to capture as many patterns from the financial data as possible. We make use of both CNN and RNN layers in our neural network architecture, which we elaborate further on in the remainder of this section.

In the context of stock price prediction, Selvin et al. (2017) conclude that RNN, unlike CNN, has an internal memory feature that is suitable for capturing sequential patterns in the input data, which is historical stock prices in their case. In terms of similarities, Selvin et al. (2017) describe that both CNN and RNN are capable of finding patterns in time series data, like historical stock returns, but that CNN and RNN use different approaches to extract these patterns from the data. In CNN, a kernel is defined and used to slide over the input of the CNN layer, which in most literature is either of size 3x3, 5x5 or 7x7. The kernel slides through the input data to extract features for all the subgroups of data equal to the kernel size. If we use a kernel size of 3x3 for example, CNN analyses all possible 3x3 submatrices within the input data to extract features. In the case of the EIIE approach, a kernel of size 3 will not analyse all submatrices of size 3x3, but all subvectors of size 3 of the input data, since the input data is a vector instead of a matrix for this approach.

Both Jiang et al. (2017) and Liang et al. (2018) conclude that a DRL model that uses a neural network containing CNN layers outperforms DRL models using a neural network containing RNN or LSTM layers, in terms of maximising the returns during the test period, but they both do not explain explicitly why CNN performs better. Selvin et al. (2017) conclude that CNN outperforms RNN and LSTM in terms of predicting stock prices, due to the capability of CNN to capture patterns in stock data that do not necessarily follow consistent patterns, due to the noisy nature of stock data. Therefore, we first add a CNN-based component in our neural network architecture designed to find the most obvious patterns in the data for each asset, which are not necessarily (temporarily) visible. After that, we add a RNN-based component, to capture sequential (short-term) patterns in the data. Since a DRL model that uses an LSTM layer is outperformed by similar DRL models using RNN and CNN layers in terms of portfolio

returns during the test period, as shown by Jiang et al. (2017), we exclude this type of hidden layer in our neural network architecture.

For extensive neural networks, as described by both Jiang et al. (2017) and Liang et al. (2018), the vanishing gradient problem can be problematic for the training procedure of the neural network. That is, the gradient used for the gradient ascent algorithm converges to zero. When deriving the gradient using backpropagation, the product of a range of (small) partial derivatives can potentially converge to zero quickly. After trying a few neural network models without any modifications to solve or prevent the vanishing gradient problem, we experienced that our gradients converged to zero quickly, which made the training of the neural network not possible. Hence the reason we focus on the vanishing gradient problem in the remainder of this section. We propose three possible solutions to tackle the gradient vanishing problem: finding the activation function that limits the vanishing gradient problem, using batch normalisation within the neural network and adjusting our neural network to be a residual neural network. These three solutions are based on the approach by Jiang et al. (2017) and Liang et al. (2018).

First, similar to Jiang et al. (2017) and Liang et al. (2018), we use ReLU activation functions. Limited research is conducted regarding activation functions in DRL context, focusing on solving the gradient vanishing problem. However, since ReLU is defined as:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

The derivative is either 1 or 0, which is computationally efficient and will partially prevent vanishing gradients. In contrast, the sigmoid function has the property that its gradient becomes very small as the absolute input values become large ($\sigma'(x)$ approaches zero as $|x|$ increases). This property of the sigmoid function leads to the vanishing gradient problem, especially in relatively deep networks with many hidden layers. When gradients become extremely small, the network becomes challenging to train effectively.

Second, we perform batch normalisation within each hidden layer similar to Liang et al. (2018). Batch normalisation was first proposed by Ioffe and Szegedy (2015) and is used to standardise (by subtracting the mean and dividing by the standard deviation) the input of each activation function. By using batch normalisation, Ioffe and Szegedy (2015) show that the distribution of the activations (the input to the activation function) is stabilised across all layers of the neural network, which subsequently makes the gradient flow in the neural network more stable and makes it less likely that the gradients will converge to zero. This will help to partially solve the gradient vanishing problem. Using batch normalisation, we prevent values from becoming very small or large by rescaling through normalisation before the values are activated through the activation function, as well as stabilising the distribution of activations across hidden layers.

Third, we transform our neural network to a deep residual network by using residual blocks and residual connections to our neural network architecture, which is based on the research by He et al. (2016). We will elaborate on the details of residual blocks and connections later in this section. Based on this research, Liang et al. (2018) use a residual network as a solution to the vanishing gradient problem as they note that "Deep residual network solves this (vanishing gradient) problem by adding a shortcut

for layers to jump to the deeper layers directly, which could prevent the network from deteriorating as the depth adds."

Similar to Liang et al. (2018), we form a neural network that starts with a CNN layer and is followed by a CNN residual block. A residual block, consisting of two successive CNN layers, helps to partially solve the gradient vanishing problem by allowing residual (shortcut) connections. The term 'residual' refers to the input of the activation function, which is added to the output of the activation function as an additional (or 'residual') term. These shortcuts provide a direct path for gradients to propagate backwards through the network, thus mitigating the risk of vanishing gradients. The price tensor data is first processed by our initial CNN layer, which then passes its output through the residual block. The connections within the residual block offer a direct pathway for the original input data to bypass the CNN layers and be added directly to the output of the activation function. If we define the activation function $F(x)$ with input x, the output $H(x)$ within a residual connection can be defined as:

$$H(x) = F(x) + x \tag{6}$$

Within the residual block, each of the two CNN layers also includes batch normalisation, before the ReLU activation function. Similar to Liang et al. (2018), the process of the residual block is repeated five times. Due to the structure of the residual block, we can create a much deeper neural network opposed to a 'standard' neural network, by repeating the residual block multiple times. Without the residual structure, the gradients would converge to zero if a similar deep neural network was created with the same number of CNN layers. The residual block hence allows the extraction of more information from the data and to reduce the risk of a vanishing gradient.

In addition, since our reward function is multivariate and hence more complex than the reward function described by Liang et al. (2018), we include a residual RNN layer after the CNN blocks, to capture sequential patterns in the input data. This approach is similar to the CNN residual block and helps to improve the gradient flow during the process of backpropagation. The difference is that instead of two hidden layers within a block, the input of the activation function is added to the activation function of the RNN layer. CNN and RNN layers can not be combined within the same residual block due to the convolutional nature of CNN and the sequential nature of RNN. Similar to our CNN residual block, we use batch normalisation before the output function of the layer.

To prevent overfitting, we add a dropout component to our neural network, similar to Liang et al. (2018). Before the softmax activation function, a random selection of neurons is set to zero, making sure the model becomes more robust. We set a percentage δ of all neurons equal to zero. In addition, since our output should be a vector of size ten and our input is a matrix of historical daily returns, we add a flatten layer to reach the desired output dimension. An activation function for our output layer that is suitable for our research is softmax, first of all since the output sums to unity (the entire budget of the agent is used) and second since the output ranges between 0 and 1 which gives interpretable long positions for each asset in our portfolio. One limitation is that this does not allow the agent to apply short selling any asset, similar to Jiang et al. (2017) and Liang et al. (2018).

In Figure 1, we show a visualisation of our proposed neural network architecture, starting from the

state and ending at the action. The residual CNN block and residual RNN connection are shown through the flow of the input X, which is added to the output of the residual block or connection $F(X)$ before flowing through the ReLU activation function. Similar to Liang et al. (2018), the residual block is repeated five times.

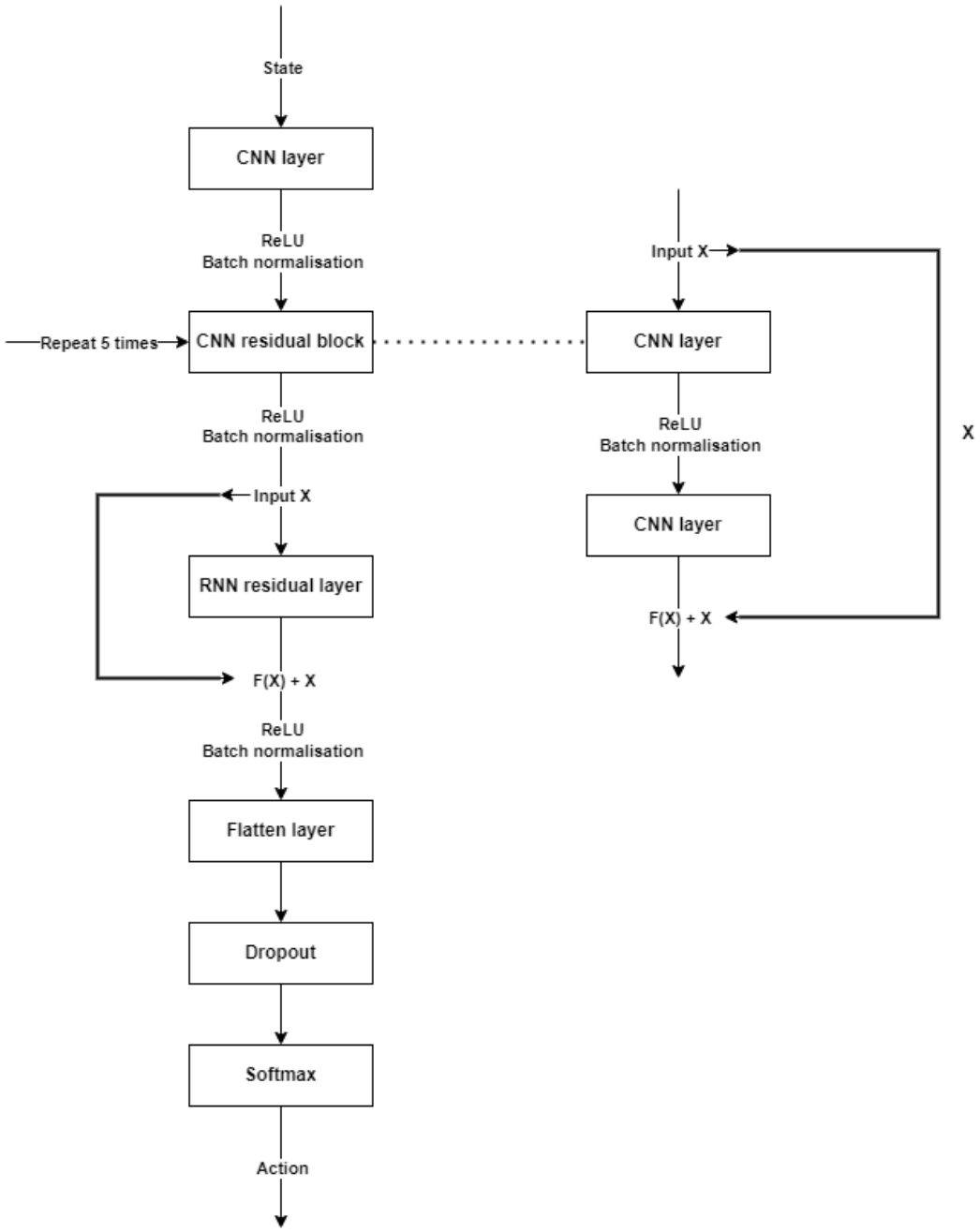


Figure 1: Overview of the proposed neural network architecture, including the residual CNN block and residual RNN layer

Using a Bayesian hyperparameter optimisation procedure, we aim to find the optimal number of nodes in each hidden layer N^{Nodes}, the value for the kernel used in the CNN layers N^{Kernel} and the appropriate

dropout rate δ, which is discussed in Section 3.1.9. To code the neural network, we make use of the Keras[4] and TensorFlow[5] packages within Python.

3.1.5 Gradient ascent algorithm

To train our DRL model, we update the neural network parameters Θ in the direction of maximising the reward function given the preferences (priority weights) of the agent. First, we initialise the model parameters randomly, after which we update the parameters using a mini-batch gradient ascent algorithm with learning rate λ. We make use of Adam to allow the learning rate to be adaptive as formulated by Kingma and Ba (2014), a method that is widely used in literature and shows successful results due to its ability to change learning rates per model parameter based on past gradients. Similar to Jiang et al. (2017), the training data is split into intervals, where one interval equals $[t_a, t_b]$, where t_a and t_b can be adjusted to change the batch size. We denote $R_{[t_a,t_b]}$ as the reward value within the interval $[t_a, t_b]$. Together, the mini-batch gradient ascent algorithm becomes:

$$\Theta \rightarrow \Theta + \lambda \nabla_{\Theta} R_{[t_a,t_b]}(\pi_{\Theta}) \tag{7}$$

To implement a form of randomness in the training procedure, we switch from mini-batch gradient descent to stochastic mini-batch gradient descent. By selecting batches randomly, we introduce a form of randomness inside our training procedure which is likely to help prevent the model from overfitting to the training data quickly.

If we define $B = \{B_1, \ldots, B_s\}$ as all s batches that can be formed given our batch size N^{Batch}, we pick a batch randomly from this set each training iteration following a uniform distribution, which is used in each step of the training procedure within our stochastic mini-batch gradient ascent algorithm. This means during each training iteration, we choose a batch $b \in B \sim U(B)$ randomly which we use to update our neural network parameters. For both the learning rate λ and the batch size N^{Batch}, we use a Bayesian hyperparameter optimisation procedure described in Section 3.1.9 to obtain the optimal hyperparameter values.

3.1.6 Model alternatives

To train our DRL model, we calculate the gradient of our reward function with respect to the model parameters Θ for each training iteration. Essentially, this means that the sum is taken from all individual gradients of returns, volatility and ESG scores within the reward function. We denote this original model that takes the 'combined' gradient as M_C. If successful, the training process results in a DRL model that performs well in terms of portfolio returns, volatility and ESG performance and the weights assigned to each metric in the reward function reflect the relative performance of each metric. However, a potential challenge is that the gradients of returns and volatility have different signs (returns positive and volatility negative in the reward function), which might cancel the effect of both gradients out if the sum is close to zero. In addition, the gradient might be quite complex since it is the sum of three gradient components,

[4] Available through Keras

[5] Available through TensorFlow

which might lead to difficulties during the training of the model. This leads to the choice of either training the DRL model based on the entire gradient (the sum of the gradients of all three components) or splitting the reward function into three components corresponding to the three metrics and focusing on each component separately rather than the combined reward. We present two alternative models that are based on the latter approach to analyse which method leads to the highest average reward value during the test period.

Pasunuru et al. (2020) define a method to train a DRL model with a multivariate reward function, where the gradient of the reward function is calculated with respect to a single component of the reward function each training iteration. Pasunuru et al. (2020) suggest first defining the priority assigned to each metric, which is a straightforward procedure in our case since we explicitly assign priority (weights) through w_R, w_V and w_E to each metric. Then, based on the given importance of each metric, they define a method that optimises the DRL model based on the metric that is expected to improve the reward value the most.

Pasunuru et al. (2020) use an approach where the priority is based on the metric that is most under-performing, where the metric that is most under-performing is selected as the metric that should be optimised in a given training iteration. Since we have explicitly assigned priorities through the weights in the reward function, we use a different approach. Despite the context of their research being centred around language generation, we make use of the intuition to calculate the gradient of the reward function with respect to a single component of the reward function, rather than considering the entire gradient, to define our first alternative model. Due to the lack of research conducted on DRL models with a multivariate reward function that is used for portfolio management applications, we believe that this intuition might be useful to achieve more favourable results in terms of average reward during the test period since this would make the gradient less complex for each training iteration.

We define $p_R = w_R$, $p_V = w_V$ and $p_E = w_E$ as the probabilities that we train the DRL model for a certain training iteration by calculating the gradient of the reward function with respect to the portfolio returns, volatility and ESG performance component respectively, in line with the weights assigned by the agent. This means that the other two metrics are left out of the gradient for a certain training iteration. We formulate two random variables: B_R and B_V, which are both boolean variables and defined using a uniformly drawn value $U(x) \sim U(0, 1)$. The returns boolean B_R is defined as:

$$B_R = \begin{cases} 1 & \text{if } U(x) < w_R \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

and the volatility boolean B_V is defined as:

$$B_V = \begin{cases} 1 & \text{if } w_R < U(x) < w_V + w_R \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Now we can define the first model based on separate gradients M_{S1} as follows, where the gradient that is used during gradient ascent over a given batch for a certain training iteration is denoted as

$\nabla_\Theta R_{[t_a,t_b]}(\pi_\Theta)_{S_1}$:

$$\nabla_\Theta R_{[t_a,t_b]}(\pi_\Theta)_{S_1} = \begin{cases} \nabla_\Theta(w_R r_t)_{[t_a,t_b]}(\pi_\Theta) & \text{if } B_R = 1 \text{ and } B_V = 0 \\ \nabla_\Theta(w_V \widehat{Vol}_t)_{[t_a,t_b]}(\pi_\Theta) & \text{if } B_R = 0 \text{ and } B_V = 1 \\ \nabla_\Theta(w_E E_t)_{[t_a,t_b]}(\pi_\Theta) & \text{otherwise} \end{cases} \tag{10}$$

$\nabla_\Theta(w_R r_t)_{[t_a,t_b]}(\pi_\Theta)$, $\nabla_\Theta(w_V \widehat{Vol}_t)_{[t_a,t_b]}(\pi_\Theta)$ and $\nabla_\Theta(w_E E_t)_{[t_a,t_b]}(\pi_\Theta)$ denote the separate gradients of the returns, volatility and ESG scores components respectively. The values of the parameters with a subscript t range from t_a until t_b, depending on the batch used for the training iteration.

The second alternative model based on separate gradients is a combination of three DRL models. The initial M_C DRL model has as input the price tensor V_t, consisting of a set of historical daily returns of the included assets. Based on this specific input, it is intuitively possible to find the assets that perform well returns-wise and adjust the neural network parameters in such a way that the output of the neural network yields higher portfolio weights for the best-performing assets in terms of returns. However, for portfolio volatility, the intuition is a bit more complex. Considering the EWMA formula for volatility, historical returns (the input for the neural network) need to be transformed to find the volatility per asset. The neural network is rather deep, which theoretically makes it possible to find this relationship and train the DRL model to minimise portfolio volatility. However, intuitively this might be too complex for the model to both use historical returns to improve portfolio returns as well as reduce portfolio volatility, since they both need to find different relationships from the same input data. Given these challenges, we develop a third model that is an ensemble of three individual DRL models that aim to optimise a single objective being the portfolio returns, volatility and ESG performance, which are then combined through a weighted sum based on w_R, w_V and w_E.

If we define the DRL models to optimise the portfolio returns, volatility and ESG scores as M_R, M_V and M_E respectively, the second alternative model M_{S2} is defined as:

$$M_{S2} = w_R M_R + w_V M_V + w_E M_E \tag{11}$$

Essentially, this means that the portfolio weights that follow from M_{S2} are a weighted sum, based on the weights assigned to returns, volatility and ESG scores, which are multiplied by the output (portfolio weights) of the three models M_R, M_V and M_E. This makes the M_{S2} model an 'ensemble' of individual DRL models to optimise portfolio returns, volatility and ESG, in line with the priority weights assigned by the agent. M_R, M_V and M_E use the same neural network architecture as described in Section 3.1.4 with the difference being the input and the values of each hyperparameter. M_R takes the original price tensor V_t as input consisting of historical returns, whereas M_V takes the EWMA volatility estimates as described in Section 3.1.3 as input. M_V has N^{Vol} number of historical EWMA-based volatility estimates for all assets as input. M_V uses the N^H figure from M_R to calculate the EWMA volatility estimates for N^{Vol} number of historical days for all assets.

M_R is chosen as the best-performing out of the M_C and M_{S1} models where $w_R = 1$ in terms of average daily returns during the validation period. Similarly, M_E is chosen as the best-performing model out of the M_C and M_{S1} models where $w_E = 1$ in terms of average daily ESG performance during the validation

period. M_V takes $w_V = 1$ to form a DRL model that focuses solely on minimising portfolio volatility.

3.1.7 Exploration and exploitation

Within reinforcement learning, the agent can either explore (try new actions) or exploit (evaluate existing actions). To train efficiently, a balance between exploration and exploitation can be beneficial. We follow Jiang et al. (2017) to form a portfolio vector memory (PVM) to store our actions during the training process. If we define T_{train} as the number of training days, the PVM is a tensor of size $N^{PVM} * T_{train} * 10$, because we have 10 assets in our portfolio. We set N^{PVM} equal to ten, given the memory-expensive storage of the past weights for all the train days that are considered. We initialise the PVM as a matrix containing only $1/N$ equally weighted portfolio weights, similar to Jiang et al. (2017) to consider portfolio weights that are relatively stable at the beginning of the training period in the sense that it assigns equal weights to all assets.

During each training day, we draw a value from a uniform distribution $U(x) \sim U(0, 1)$, if it is smaller than a specified value $0 \leq \epsilon \leq 1$, we use exploitation (use the corresponding weights from the PVM) and otherwise exploration (use the corresponding price tensor to derive the portfolio weights through the output of the neural network). If the agent uses exploration at day t, the corresponding column t of the PVM is updated with the new portfolio weights. After a number of training iterations, the majority of the $1/N$ weights will be overwritten by the actual output of the DRL model. Using PVM, we can improve training efficiency since we make less use of the computationally expensive neural network. In addition, we improve the stability of the training procedure mainly at the beginning of the training procedure, since $1/N$ weights are also considered, which is relatively less volatile compared to the output of the (limited trained) neural network model at the beginning of the training procedure. In Section 3.1.9, we describe how we aim to find the optimal value of ϵ through a Bayesian hyperparameter optimisation procedure.

3.1.8 Priority weights and reward function evaluation

We make the following selection of weights for w_R, w_V and w_E that each model will consider: $\{w_R, w_V, w_E\}$ = {(1/3, 1/3, 1/3), (1, 0, 0), (0, 1, 0), (0, 0, 1), (0.5, 0.5, 0), (0.5, 0, 0.5), (0, 0.5, 0.5), (0.7, 0.2, 0.1), (0.2, 0.7, 0.1), (0.45, 0.45, 0.1)}. Since the ESG data is constant, the last three weight combinations assume that a relatively low w_E still makes optimising the ESG performance possible, due to the relatively straightforward optimisation procedure for this metric. Using the selection of weights, we can evaluate the ability of each of the models to optimise the portfolio in line with the priority assigned to each metric.

The reward function we defined in Section 3.1.2 is convenient for the training of the DRL models since the gradient of the reward function can be split up into individual components per portfolio metric. However, in terms of evaluating the actual reward it is not. All three metrics have different scales, making the weighted sum of all metrics not informative or sparse. However, if we perform any kind of standardisation and take the average value or sum for each metric we are not able to observe if the training will improve the portfolio performance with respect to all metrics. If we take the average of a standardised variable (with standardisation using subtraction of mean and division of standard deviation) it will per definition be around zero. Therefore we make use of the reward function of 3.1.2 to train the

DRL models through the gradient ascent algorithm and the following reward functions to evaluate which model for a given combination of weights performs best.

This solution is not very intuitive, but we will focus mainly on the portfolio returns, volatility and ESG performance metrics in the analyses of the results. We believe that the tradeoff between having a reward function that is either convenient to use during training or one that is convenient for evaluation might be the most challenging aspect of a multivariate reward function in the context of portfolio management. The reward functions we define in the remainder of this section are convenient to use for evaluation, but due to the division and multiplication of metrics the reward functions are not very convenient for the training procedure due to the complex gradients.

For the models with weight $w_R = 1$, $w_V = 1$ and $w_E = 1$, the alternative reward function for evaluation is simply the average daily value during the test period for the corresponding portfolio performance metric. For the other weight combinations, we use the following reward functions to evaluate which model for a given weight combination performs best in terms of reward. Similar to the intuition behind the Sharpe ratio, we incorporate a term that is equal to the portfolio returns divided by the portfolio volatility. This term is then multiplied by the ESG performance to result in 'ESG and risk-adjusted returns'. The intuition behind the '(1-w_V)' part is that the reward increases when the denominator decreases, meaning that if we assign a high value to w_V the reward should increase relatively more when the portfolio volatility at time t decreases compared to a lower w_V value. The intuition behind the '(1+E_t)' part is that since we standardised the ESG data, this prevents the reward from becoming instantly negative when $E_t < 0$ (now only when E_t <1). This can be interpreted as a factor that either increases or decreases the risk-adjusted returns. Despite not being consistent, an increase in performance for each of the three metrics in the reward function as explained in Section 3.1.2 also leads to an increase in the reward functions stated below. For the different weights, the average daily reward functions are defined as:

$$R = \frac{1}{T_{TP}} \left(\sum_{t=TP_{begin}}^{TP_{end}} \frac{w_R r_t}{(1-w_V)\widehat{Vol}_t} w_E(1+E_t) \right) \quad \text{if } w_R > 0, w_V > 0, w_E > 0 \tag{12}$$

$$R = \frac{1}{T_{TP}} \left(\sum_{t=TP_{begin}}^{TP_{end}} \frac{w_R r_t}{(1-w_V)\widehat{Vol}_t} \right) \quad \text{if } w_R > 0, w_V > 0, w_E = 0 \tag{13}$$

$$R = \frac{1}{T_{TP}} \left(\sum_{t=TP_{begin}}^{TP_{end}} (w_R r_t) w_E(1+E_t) \right) \quad \text{if } w_R > 0, w_V = 0, w_E > 0 \tag{14}$$

$$R = \frac{1}{T_{TP}} \left(\sum_{t=TP_{begin}}^{TP_{end}} \frac{1}{(1-w_V)\widehat{Vol}_t} w_E(1+E_t) \right) \quad \text{if } w_R = 0, w_V > 0, w_E > 0 \tag{15}$$

For the above reward functions, we have also tried to use them in the training process, but the complexity of the division and multiplication, as well as the fact that the gradient is not a sum, made the gradients too complex to effectively train the DRL models to optimise the returns, volatility and ESG performance. Hence the distinction between the reward function used for training and the reward functions used for evaluation. In future research, a possible solution to this complex issue might be interesting to find.

3.1.9 Bayesian hyperparameter optimisation

To derive the hyperparameters that yield the highest reward values during the validation period for a certain DRL model, we perform a hyperparameter optimisation procedure on the hyperparameters of the model. There are several hyperparameters we can tune to optimise the reward that the policy function yields. The parameters that we aim to optimise are the learning rate λ, the exploration/exploitation distribution ϵ, the dropout rate ρ, the batch size N^{Batch}, the number of nodes per hidden layer N^{Nodes}, the kernel size N^{Kernel} for the CNN layers and the number of historical daily returns data included in the price tensor N^H.

A traditional grid search would be very computationally intensive and time-consuming since this would require considering a significant amount of different combinations of hyperparameters. For example, if we consider three values per hyperparameter for a single DRL model this would already lead to 3^7 = 2,187 different models to evaluate, which is not computationally feasible given the (already quite complex) training procedure. A different approach is Bayesian hyperparameter optimisation, designed to efficiently evaluate different hyperparameter values. We follow Wu et al. (2019) to define the Bayesian hyperparameter optimisation. First, we define the outcome space $\mathbf{X}$, containing the ranges of each of our hyperparameters in which the optimisation procedure aims to find the optimal value for each hyperparameter:

$$\mathbf{X} = \begin{bmatrix} \lambda \\ \epsilon \\ \rho \\ N^{Batch} \\ N^{Nodes} \\ N^{Kernel} \\ N^H \end{bmatrix} \in \begin{bmatrix} [0.00001, 0.01] \\ [0, 1] \\ [0, 0.5] \\ [25, 75] \\ [5, 20] \\ [3, 7] \\ [45, 55] \end{bmatrix} \tag{16}$$

Some hyperparameters have a relatively wider range compared to others, which allows the Bayesian hyperparameter optimisation procedure to find the optimal values, without the need for significant prior belief on each parameter. Some hyperparameters, namely λ, ϵ, ρ, N^{Kernel} and N^H are defined using some prior knowledge. Liang et al. (2018) set $\lambda = 10^{-3}$ and ϵ is per definition between 0 and 1, as we can either use full exploitation or exploration and anything in between. Also, ρ is per definition between 0 and 1, but we do not think it is realistic to drop more than 50% of all neurons in the neural network.

N^{Batch} and N^{Nodes} have a broader range than N^{Kernel} and N^H as we have less intuition for these values. N^{Kernel} is usually given an odd number of either 3, 5 or 7 which are also the three values we consider (we round any value between 3 and 7 to the nearest option of either 3, 5 or 7). Jiang et al. (2017) set $N^H = 50$, which we use to set a relatively narrow range of N^H between 45 and 55. For the M_{S2} models where $w_V = 1$, we set a range of N^{Vol} to be between 5 and 15, as we believe that the volatility of a certain asset changes relatively less from day to day compared to daily returns, making it redundant to have a high dimensional input of past EWMA-based volatility estimates.

As described by Wu et al. (2019), we choose an acquisition function to assess whether a new set of hyperparameters is expected to improve the performance of the DRL model. There are several acquisition functions, with probability of improvement (PI) and expected improvement (EI) being the most well-known. According to Wu et al. (2019), EI is more suitable for a balance of exploration and exploitation, whereas PI only considers exploration. In addition, they claim that EI is less prone to get stuck in a local optimum. Therefore, we choose EI as our acquisition function. The remainder of the Bayesian hyperparameter optimisation procedure, including more details regarding the acquisition function, is available in the Appendix.

3.1.10 Training procedure and parallel training

After finding the optimal set of hyperparameters for each DRL model, we train the model to find the optimal neural network weights that yield the highest reward during the test period. To train our DRL models, we set the maximum number of training iterations N^{train} equal to 200, since our model is quite complex and needs to find multiple patterns to optimise a range of metrics. In practice, we observed that after around 150 training iterations there was not much training improvement in terms of average daily reward during the validation period. For each day within a randomly drawn batch b, we either use exploitation or exploration. Then, for a given batch we update our model parameters Θ through gradient ascent. Since we train a substantial number of models, we stop the training procedure if the reward value of the considered model has not increased for five training iterations when it is tested on the validation set consecutively (based on the reward functions discussed in the previous section). This prevents the model from continuing learning when this does not yield any additional gains in terms of training and saves computational resources.

The training procedure of all DRL models uses a training set, which is a subset of our total data set. We define a train, validate and test ratio of 70% train data, 10% validation data and 20% test data. To prevent overfitting to some extent, we split the train and validate data into 8 folds, meaning that the total of 80% train and validate data is split into 8 different combinations of train and validate data. After completing the training of each of the 8 folds, we take the average of the model parameters Θ that result from each fold, also known as k-fold cross-validation. Together, the training procedure is defined as:

Algorithm 1 DRL training procedure

Randomly initialise Θ
for training iteration = 1, ..., T_{train} **do**
    Choose randomly batch $b \sim \mathcal{U}(B)$
    for $t = t_a, \ldots, t_b \in b$ **do**
        if $u(x) \leq \epsilon$ **then**
            Exploit: select weights PVM_t
        else
            Explore: calculate weights $w_t = \pi_\Theta(S_t)$
            Update PVM_t as w_t
        Update model weights through gradient ascent $\Theta \rightarrow \Theta + \lambda \nabla_\Theta R_{[t_a,t_b]}(\pi_\Theta)$
        Stop if: 5 training iterations consecutively no increase in reward during validation period

The decision regarding which part of the data belongs to the training/validation set and which part to the test set is arbitrary. In our research, we will analyse how the model performs on the last 20% of the data, which includes the COVID-19 crisis as well as a significant rally of stocks.

Since our research requires considering a range of computationally expensive models and different combinations (different weights for each metric and using all train data or not for the hedging strategy), we make use of SURF Research Cloud[6]. The software allows parallel training by making use of multiple GPUs at the same time, with a maximum of 16. The total budget which can be used is 15,000 SBU which means that 16 GPUs can be used for a total of 15,000/16 hours. The wallet and license are obtained through the EDSC Erasmus University Rotterdam Data team. Using the wallet of 15,000 SBU, we can conduct our computationally expensive Bayesian hyperparameter optimisation procedure as well as the training of the DRL models.

3.2 Extreme downside market risk hedging

Given the train, validate and test split, the train data set contains both crises and stock/bond rallies with extreme events. The volatility term in the reward function already aims to minimise the portfolio volatility for a given DRL model, but can still be quite prone to extreme market conditions, as they remain hard to predict. Per definition, the number of days within our training set that can be considered as extreme downside market events is limited, hence the reason why it can be potentially hard to train our DRL model to optimise portfolio returns, volatility and ESG performance during such events. Therefore, we also train the DRL models on the train data set where the days with the 5% lowest daily returns are excluded and complement these models with a hedging strategy to reduce the portfolio volatility. For all the pairs of assets, we fit copulas to derive the matrix of lower tail dependencies which we use to limit the lower tail risk during days we expect to classify as extreme downside market events.

In Section 2, we define extreme downside market events as the trading days that result in the 5% lowest daily returns during a certain period of time. In advance for a certain day t, the agent does not

[6] Available through SURF Research Cloud

know for certain whether or not the trading day can be classified as an extreme downside market event. Therefore we refer to trading days which the agent expects to be extreme downside market events as 'extreme downside market circumstances'. Consequently, we use a different threshold at day $t-1$ to trigger our hedging strategy that starts at day t. Taking the average of all ten assets over the entire train data set, the 10% percentile threshold value of the lowest daily returns is -0.01. We use this threshold as the trigger to switch to the hedging strategy the next trading day during the testing period. This means that we switch to the hedging strategy when we observe that our portfolio obtains a daily return that is below -0.01.

During our train data set, negative returns on average are observed during two consecutive days, which we use to formulate our hedging strategy, using the assumption that if we observe a daily portfolio return that is below -0.01 at day $t-1$, we expect that the return at day t is more negative and might be considered an extreme downside market event (extreme downside market circumstances). If this assumption is correct, the switch to a hedging strategy might prevent the exposure to a portfolio return that we would classify as an extreme downside market event otherwise. As switching to a hedging strategy frequently can be expensive due to transaction costs in a real-world application, we switch to a hedging strategy for five days, after which we switch back to the DRL model.

In Section 3.2.1 we elaborate on our choice of copula to estimate the lower tail dependencies and in Section 3.2.2 we describe our hedging strategy.

3.2.1 Copula selection

To model the dependencies of all assets, we make use of copulas. We follow Sklar (1973) to model the joint distribution of two random variables X_i and X_j for $i \neq j$, through a copula containing the marginal distributions:

$$F(x_i, x_j) = C(F_i(x_i), F_j(x_j)) \tag{17}$$

Here, $F(x_i, x_j)$ denotes the CDF of the joint distribution and $C(F_i(x_i), F_j(x_j))$ denotes the copula with marginal distributions $F_i(x_i)$ and $F_j(x_j)$. A crucial factor of copulas is that the marginal distributions are uniformly distributed. We follow Ko and Hjort (2019), to take the derivate on both sides to find an expression for the joint PDF $f(x_i, x_j)$ containing the marginal PDFs $f_i(x_i)$ and $f_j(x_j)$:

$$\begin{aligned} f(x_i, x_j) &= \frac{\partial^2}{\partial x_i \partial x_j} \left[C(F_i(x_i), F_j(x_j)) \right] \\ &= c(F_i(x_i), F_j(x_j)) \cdot \frac{\partial^2}{\partial x_i \partial x_j} \left[F_i(x_i) \right] \cdot \frac{\partial^2}{\partial x_i \partial x_j} \left[F_j(x_j) \right] \\ &= c(F_i(x_i), F_j(x_j)) \cdot f_i(x_i) \cdot f_j(x_j) \end{aligned} \tag{18}$$

Where $c(F_i(x_i), F_j(x_j)$ is the derivative of $C(F_i(x_i), F_j(x_j)$ with respect to x_i and x_j. Using this expression of $f(x_i, x_j)$, we can derive the log-likelihood of a time sequence of $t = 1$ until T as follows:

$$\begin{aligned}\log L &= \sum_{t=1}^{T} \left[\log\left(c(F_{it}(x_{it}), F_{jt}(x_{jt}))\right) + \log\left(f_{it}(x_{it})\right) + \log\left(f_{jt}(x_{jt})\right)\right] \\ &= \sum_{t=1}^{T} \log\left[c(F_{it}(x_{it}), F_{jt}(x_{jt}))\right] + \sum_{t=1}^{T} \log\left[f_{it}(x_{it})\right] + \sum_{t=1}^{T} \log\left[f_{jt}(x_{jt})\right] \end{aligned} \tag{19}$$

Using this expression, Ko and Hjort (2019) show that two-stage estimation can be used to find an expression for the joint distribution by first estimating the marginal distributions $f_i(x_i)$ and $f_j(x_j)$. The marginal distributions are then used to estimate the copula parameters using maximum likelihood. Before estimating the copula parameters for asset combinations, we need to choose a copula type to estimate the lower tail dependencies of the assets. Kole et al. (2005) show that the Student's t copula outperforms the Gumbel and Gaussian copulas in capturing the joint downside risk of a balanced portfolio containing stocks and bonds (and real estate in their research).

In addition, Lourme and Maurer (2017) show that the Student's t copula outperforms a range of other copulas (Frank, Joe, Clayton, and Gumbel) in terms of the BIC model comparison metric. Given that the Student's t copula is symmetric, it does not make a distinction between the left tail and the right tail of the joint distribution of returns. However, empirically, the dependence in the left tail is stronger than the right tail, as shown by Longin and Solnik (2001) for example. Consequently, the estimated lower tail dependence of the Student's t copula for a given pair of assets is possibly lower than the actual lower tail dependence. However, Kole et al. (2005) conclude that the Student's t copula performs relatively well in capturing the empirical tail behaviour for the considered copulas in their research.

The Student's t copula is a copula family used to model the joint distribution of random variables. It is characterised by two parameters, namely the correlation value denoted as ρ and the degrees of freedom denoted as ν, which controls the tail behaviour. We follow Ko and Hjort (2019) to denote the Student's t copula of two random variables as:

$$C_{Student-t}(u_i, u_j; \rho, \nu) = \Psi_2(\Psi^{-1}(u_i; \nu), \Psi^{-1}(u_j; \nu); \Omega, \nu)$$

Where Ω is the correlation matrix, Ψ_2 denotes the CDF of a bivariate Student's t distribution with correlation value ρ and degrees of freedom ν. In addition, Ψ^{-1} is the inverse of a univariate CDF of the Student's t distribution with ν degrees of freedom, mean zero and degrees of freedom of one. Also, u_i and u_j denote two uniformly distributed random variables.

3.2.2 Hedging procedure

We assume that the marginal distribution of each of the stocks and bonds follows a Student's t distribution, which we fit over the train data using maximum likelihood. For each pair of assets i and j for $i \neq j$, we start by estimating the marginal Student's t distribution t_i and t_j. Using the marginal t distributions, we estimate the bivariate t distribution for all asset pairs. Using this distribution, we simulate 10,000 pairs of data for each asset pair. Then, we transform the values we observe for each of the two assets into uniformly distributed values using the inverse CDF of t_i and t_j. Finally, after finding the pseudo

samples of uniformly distributed values, we estimate the copula parameters of copula C_{ij} using maximum likelihood. To do so, we make use of the 'Copula' package in MATLAB by Kopocinski (2024).

Since our goal is to limit our lower tail risk during extreme downside market circumstances, we derive the lower tail dependencies for all combinations of assets i and j. The lower tail dependence aims to measure the comovement of the two assets during extreme negative movements. The lower tail dependence is defined as:

$$\lambda_L(ij) = \lim_{q \to 0} \frac{C_{ij}(q, q)}{q} \tag{20}$$

Where $\lambda_L(ij)$ is the lower tail dependence of assets i and j. Caillault and Guegan (2005) show that the lower tail dependence of the Student's t copula can be written as:

$$\lambda_L(ij) = 2t_{\nu+1}\left(\sqrt{\frac{(\nu+1)(1-\rho_{ij})}{1+\rho_{ij}}}\right) \tag{21}$$

Where $t_{\nu+1}$ is a Student's t distribution with $\nu+1$ degrees of freedom and ρ_{ij} is the correlation parameter of the copula C_{ij}. After estimating the copula parameters of all possible copulas C_{ij} of assets i and j, we estimate all lower tail dependencies for all pairs of the ten assets. Also, since we use the train data to derive the lower tail dependencies, we assume that these values remain constant during the test period.

We define L as the matrix containing all lower tail dependencies of our selected assets within the portfolio, where the diagonal consists of ones. Using the portfolio weights w_t at time t, we calculate the lower tail exposures of all assets as the vector $w_t L$. We perform this multiplication to derive the sum of lower tail exposure per asset. After this, we rank the assets from low to high in terms of lower tail exposure. Then, the agent moves to a hedging strategy that is defined as going long in the five assets with the lowest lower tail exposure and going short in the five assets with the highest lower tail exposure. Both the long and short part of the portfolio has equal weights, meaning the weights are set at 0.2. If our agent encounters a daily return that is lower than -0.01, we trigger our hedging strategy for five days, starting the following day. Otherwise, the portfolio weights that follow from the DRL model are used. Together, the hedging strategy is:

1. Estimate all lower tail dependencies of the stocks and bonds in the portfolio and form the lower tail dependencies matrix L using the train data.

2. If at day $t-1$ the portfolio return $r_{t-1} < -0.01$, the hedging strategy is triggered at day t for five days. Otherwise, the DRL model weights are used.

3. In case the hedging strategy is triggered, we first calculate $w_t L$ and rank the assets from low to high in terms of lower tail exposure. Then, we invest long in the five assets with the lowest lower tail exposure and short in the five assets with the highest lower tail exposure with all assets assigned a weight of 0.2. During five days, the hedging strategy is used, after which we switch back to the DRL model.

In this way, we aim to deal with extreme market downside circumstances, without deciding to temporarily let the agent stop trading at all. Since the hedging strategy is focused on reducing the portfolio volatility by reducing the lower tail risk during extreme market downside circumstances, the portfolio return and

ESG performance objectives of the agent are temporarily neglected when she switches to the hedging strategy, which may have significant negative effects on the portfolio returns and ESG performance.

3.3 Practical issues

In portfolio management research, problems can arise when considering issues such as transaction costs and slippage (the difference in expected and realised price of a trade). For simplicity, we assume no transaction costs and zero slippage. Jiang et al. (2017) propose a solution for the transaction costs issue by incorporating a constant 'transaction costs factor' that extends the DRL models. However, since this is out of the scope of our research goal, we assume zero transaction costs. We can assume zero slippage in this research since we are not designing a model for a high-frequency trading environment, but rather one that is rebalanced daily. Therefore, it is likely that there is no significant difference between the expected and realised trading price when we execute a trade.

3.4 Performance evaluation

3.4.1 Benchmark models

To compare the performance of our proposed models, we compare the performance of the DRL models during the test period with a group of benchmark models. The benchmark models are both traditional portfolio management benchmarks, as well as machine learning-based models.

First, the most used benchmark model in portfolio management research is the equally weighted or $1/N$ portfolio. Since there is no input in this model - and therefore no prediction errors - it tends to be hard to outperform this seemingly simple model as Plyakha et al. (2012) show for example.

Second, we use the GMV portfolio as a benchmark, as shown by Merton (1972) to minimise the portfolio volatility. Since the size of N^H ranges between 45 and 55, as stated in Section 3.1.9, we take the average value of 50 as the number of past historical returns data used to estimate the covariance matrix, using the same approach as described in Section 3.1.3. For the GMV portfolio, we set the constraint that the portfolio weights should be positive, to be consistent with the DRL models that have the same constraint.

Also, we define a machine learning-based benchmark model. Our proposed neural network architecture is made in such a way that the entire price tensor is the input. However as mentioned, both Jiang et al. (2017) and Liang et al. (2018) consider the EIIE neural network architecture, where the price tensor is split per asset and passed independently through the neural network. To test if our DRL models outperform this approach, we also include the EIIE-based counterpart models for all DRL models as benchmarks.

3.4.2 Performance measures

Due to the inconsistent reward functions that are used for training and evaluation, we mainly focus our analysis of the results on the individual portfolio performance metrics included in the reward functions, namely the average daily returns, volatility and ESG performance. Next to these metrics, we analyse the

performance in terms of turnover and maximum drawdown. We include these metrics in case the agent has any interest in identifying the DRL model that yields the lowest transaction costs, but we will focus the majority of our analysis on the average daily returns, volatility and ESG performance. Turnover is defined as the sum of the differences between the portfolio weights during rebalancing, which we sum over the test period. Maximum drawdown (MDD) is defined as the largest drop in portfolio value expressed as a percentage over a given period. de Melo Mendes and Lavrado (2017) show that the formula for MDD is given as:

$$MDD = \max_{1 \leq k < j \leq T} \frac{P_k - P_j}{P_k} \tag{22}$$

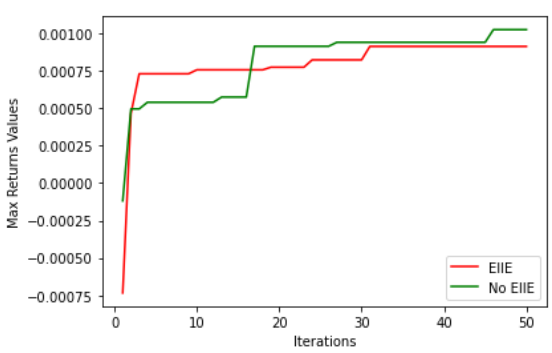
Where P_k and P_j are the portfolio values at time k and j within the test period that runs until $t = 1$ until $t = T$.

4 Results

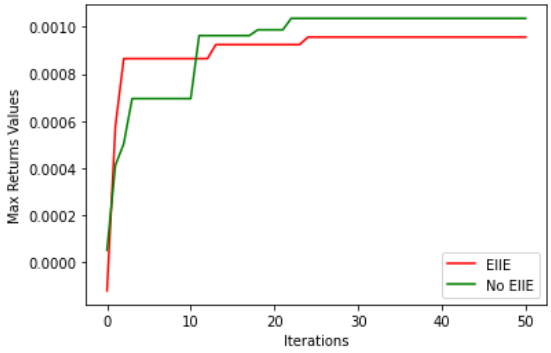
4.1 Bayesian hyperparameter optimisation procedure

For all the DRL models (both using the EIIE and proposed neural network architecture) and all weight combinations of w_R, w_V and w_E, we perform the Bayesian hyperparameter optimisation procedure to derive the hyperparameters that yield the highest reward during the validation period. We consider 50 sets of hyperparameters X that have an expected improvement $E\{I(X)\} \geq 0$ for each DRL model during the optimisation procedure.

In Figure 2, we illustrate the Bayesian hyperparameter optimisation procedure by showing the progress in average daily returns during the validation period for the M_C and M_{S1} models using the proposed and EIIE neural network structure, where we make use of all training data and set $w_R = 1$. We observe that the majority of the increase in daily average return is realised in the first number of iterations where $E\{I(X)\} \geq 0$ for a set of randomly chosen hyperparameters X. Especially for the M_{S1} models, the progress flattens after around 25 iterations, due to the difficulty of finding a new set of hyperparameters X that outperform any previous set. For the M_C model, some improvement is realised after 30 iterations for the model with an EIIE neural network structure and the model with the proposed neural network structure even improves after 45 iterations.



(a) M_C models using the proposed and EIIE neural network structure for $w_R = 1$



(b) M_{S1} models using the proposed and EIIE neural network structure for $w_R = 1$

Figure 2: Progression of average daily returns during the validation period resulting from the Bayesian hyperparameter optimisation procedure for the M_C and M_{S1} models where $w_R = 1$. For each iteration, for both the M_C and the M_{S1} models, we show the progression of average daily returns during the validation period after training the models for 50 training iterations using the entire train data set

Similar to 'standard' machine learning training, the rate of improvement for each iteration of the Bayesian hyperparameter optimisation procedure converges after a certain number of iterations. For our DRL models, this likely happens when the gradient converges to zero after getting stuck in a local optimum. For the Bayesian hyperparameter procedure, the expected improvement value decreases after each iteration where the average reward during the validation period for a certain set of hyperparameters X increases. Despite realising a reasonable amount of improvement for both the M_C and M_{S1} models using the proposed and EIIE neural network structure, the procedure of finding a relationship between the input (the set of hyperparameters X) and output (the average reward during the validation period) remains quite complex, potentially leading to sub-optimal results.

Since the reward function has no closed form of all the hyperparameters, we need to essentially model the average reward value during the validation period, given a certain set of hyperparameters X. One possible way to improve this would be to increase the number of randomly selected sets of hyperparameters that are used to form the initial Gaussian process or test several different kernel functions $k(X, X')$, however, this was not within the scope of our research, also due to the limit on our SURF Research Cloud budget for computational resources.

Compared to random search or grid search, the Bayesian hyperparameter optimisation procedure needs to consider fewer sets of hyperparameters since the only sets that are considered are the ones where $E\{I(X)\} \geq 0$. This makes it possible to consider more sets of X without the need to find the computationally expensive outcome. At some point, however, the Bayesian hyperparameter optimisation procedure seems to converge to a random search procedure, as $E\{I(X)\} \to 0$, which is visible in Figure 2 through the progress for instance for the M_{S1} models after around 25 iterations.

4.2 Analysis of model performances

After training all the DLR models using the hyperparameters that follow from the Bayesian hyperparameter optimisation procedure, we test the performance of the DRL models for both the proposed and EIIE neural network structure model during the test period. In Table 3 we show the performance of all DRL models for all ten weight combinations of w_R, w_V and w_E for the M_C, M_{S1} and M_{S2} models. Using a paired t test, we test the difference between the performance of the DRL models using the proposed and the EIIE neural network structure. Finally, we show the performance of the $1/N$ equally weighted portfolio and the GMV portfolio. We observe that in terms of average daily reward value, the M_{S2} outperforms the other DRL models.

Regarding the average daily returns, the M_{S1} model performs best for $w_R = 1$, which we expected at first to be equal to the M_C model, given that $w_R = 1$, making the models equivalent to each other. However, due to the random nature of the training of the neural network, the models do not result in the same average daily return during the test period. The M_{S1} model found a solution of investing almost all the budget in one asset, after which it got stuck in a local optimum and has not improved since. The M_C - EIIE and M_{S1} - EIIE models show similar performance in terms of average daily returns for $w_R = 1$, without getting stuck in a local optimum of investing the majority of the agents budget in one asset. Despite showing quite some differences in terms of average daily return values between the DRL models using the proposed and EIIE neural network, the differences are not significant based on the paired t test at a 5% significance level.

In terms of average daily portfolio volatility, the GMV portfolio performs best during the test period. For the other models, the M_{S2} model performs best for $w_V = 1$ using the proposed neural network architecture. We believe that this is due to the way that the historical returns data are transformed in the M_{S2} model for $w_V = 1$, where historical daily returns are used to derive the EWMA volatility estimates per asset for a number of historical days, which are then used as input to the neural network. The difference between this model and the GMV portfolio in terms of average daily volatility is significant using the paired t test at 5% significance.

A possible explanation of the outperformance of GMV compared to the M_{S2} model in terms of average daily volatility might be that the direct optimisation of portfolio weights is more effective than the M_{S2} model. The latter aims to find the patterns in historical EWMA volatilities for all assets and assign the highest portfolio weights to the least volatile assets. For the other models, the difference between the DRL models using the proposed and EIIE neural network structure regarding average daily volatility is often significant, but not consistently in favour of one of the two. In general, when w_R and w_E increase, the portfolio volatility increases for most of the models, especially when $w_R = 1$ and $w_E = 1$.

Regarding the average daily ESG scores, the DRL models using the proposed neural network structure generally perform significantly better than the DRL models using the EIIE neural network structure, except for some instances. Since the ESG values have been standardised, the $1/N$ model has an average daily ESG score of around zero. A possible explanation for the difference between the ESG performance of the DRL models using the proposed and EIIE neural network structure is the sensitivity of the gradient for the constant ESG data during training. More specifically, the proposed neural network contains a

relatively higher number of parameters compared to the EIIE neural network, because the output of the EIIE model is a single value and for the proposed models it is ten and because the input of the proposed neural network is a matrix of historical returns data and the input for the EIIE neural network is a vector. Since the ESG data is constant, the gradient of the ESG component in the reward function aims to push the output of the neural network constantly towards the assets with the highest ESG score. Hence, the gradient of the ESG component is generally less complex compared to the returns and volatility gradient component.

For the EIIE counterpart, the lower number of values in the gradient, due to the fewer number of neural network parameters, makes the ESG component likely less dominant within the total gradient. The returns and volatility gradient component likely contains more values that are relatively more informative compared to the proposed neural network structure since it contains fewer parameters. Therefore, the ESG component is likely to be less dominant in the DRL models with the EIIE neural network architecture.

The turnover value is mainly important for the transaction costs that are related to rebalancing the portfolio daily. Despite our assumption of zero transaction costs, in a real-world application, turnover might be relevant to the agent given the existence of transaction costs. Some models show a very small turnover value, such as the M_{S2} - EIIE model when $w_V = 1$, because the training procedure got stuck at a local optimum, making it not very representative to a real-world application. Intuitively, the higher the number of metrics that the agent aims to optimise (have a weight larger than zero in the reward function) the higher the turnover, since the model constantly aims to rebalance the portfolio each day to optimise the different metrics given the environment.

Finally, we show the maximum drawdown (MDD) for each model, which is a metric used for risk management purposes. Despite resulting in lower volatility, the M_{S2} model for $w_V = 1$ has a higher MDD value than the $1/N$ portfolio, which illustrates that there is no clear relationship between the average daily portfolio volatility and the corresponding MDD value.

Table 3: Overview of the performance of all DRL models using both the proposed and EIIE neural network structure as well as the $1/N$ and GMV portfolio. Using the entire training dataset, we show the average daily reward value, returns, volatility, ESG scores and the total turnover value. MDD stands for maximum drawdown and per weight combination the highest reward value is made bold, whereas for the other metrics, the bold figures represent the highest or lowest value across all weight combinations and models. Finally, the '$*$' indicates that $p<0.05$ for the paired t test between the DRL model using the proposed neural network structure and its EIIE counterpart

Metrics						
Model	**Reward**	**Returns**	**Volatility**	**ESG**	**Turnover**	**MDD**
Weight Combination: $w_R = 1/3, w_V = 1/3, w_E = 1/3$						
M_C	0.000149	0.000621	0.676*	0.0101*	867*	0.315
M_{S1}	0.000117	0.000513	0.675*	0.0181	879*	0.364
M_{S2}	**0.000261**	0.000598	0.699*	0.729*	17*	0.359
M_C-EIIE	0.000117	0.000464	0.693	0.00604	189	0.402
M_{S1}-EIIE	8.51e-05	0.000409	0.698	0.00802	800	0.360
M_{S2}-EIIE	0.000218	0.000583	0.679	0.623	850	0.432

Table 3 – continued from previous page

Model	Reward	Returns	Volatility	ESG	Turnover	MDD
Weight Combination: $w_R = 1, w_V = 0, w_E = 0$						
M_C	0.00102	0.00102	0.783*	-0.235*	881*	0.439
M_{S1}	**0.00115**	**0.00115**	0.971*	0.333*	8.61e-06*	0.291
M_{S2}	0.00115	0.00115	0.971*	0.333*	8.61e-06*	0.291
M_C-EIIE	0.00103	0.00103	0.739	0.0469	1281	0.501
M_{S1}-EIIE	0.00104	0.00104	0.796	0.129	2544	0.567
M_{S2}-EIIE	0.00104	0.00104	0.796	0.129	2544	0.567
Weight Combination: $w_R = 0, w_V = 1, w_E = 0$						
M_C	0.710*	0.000578	0.710*	-0.1975*	446*	0.368
M_{S1}	0.713*	0.000473	0.713*	-0.215*	791*	0.407
M_{S2}	**0.632***	0.000261	0.632*	0.0878*	43*	0.574
M_C-EIIE	0.689	0.000487	0.689	0.00045	290	0.381
M_{S1}-EIIE	0.687	0.000480	0.687	-0.000242	31	0.405
M_{S2}-EIIE	0.647	0.000376	0.647	-0.0369	**2.31e-06**	0.491
Weight Combination: $w_R = 0, w_V = 0, w_E = 1$						
M_C	**1.75**	0.000364	0.966	**1.75**	11	0.307
M_{S1}	1.71	0.000360	0.949	1.71	77*	0.300
M_{S2}	1.75	0.000364	0.966	1.75	11	0.307
M_C-EIIE	1.75	0.000365	0.965	1.75	12	0.304
M_{S1}-EIIE	1.70	0.000358	0.947	1.70	54	0.299
M_{S2}-EIIE	1.75	0.000365	0.965	1.75	12	0.304
Weight Combination: $w_R = 0.5, w_V = 0.5, w_E = 0$						
M_C	0.000927	0.000708	0.731*	0.180*	1135*	0.317
M_{S1}	0.000621	0.000457	0.718*	0.144*	714*	0.408
M_{S2}	**0.00106**	0.000706	0.702*	0.201*	22*	0.399
M_C-EIIE	0.000770	0.000574	0.693	0.00312	256	0.394
M_{S1}-EIIE	0.000670	0.000474	0.697	0.00354	176	0.376
M_{S2}-EIIE	0.00101	0.000709	0.670	0.0428	1272	0.507
Weight Combination: $w_R = 0.5, w_V = 0, w_E = 0.5$						
M_C	0.000223	0.000371	0.853*	1.38*	554*	0.310
M_{S1}	0.000183	0.000584	0.690	0.320*	826*	0.345
M_{S2}	**0.000396**	0.000766	0.664*	1.04	6*	**0.280**
M_C-EIIE	9.38e-05	0.000521	0.893	0.116	1580	0.442
M_{S1}-EIIE	0.000112	0.000442	0.685	0.000132	109	0.442
M_{S2}-EIIE	0.000314	0.000686	0.633	0.940	1275	0.415
Weight Combination: $w_R = 0, w_V = 0.5, w_E = 0.5$						
M_C	1.90*	0.000233	0.892*	0.663*	297*	0.306
M_{S1}	1.84*	0.000575	0.689*	0.236*	808*	0.524
M_{S2}	**2.84**	0.000321	0.699*	0.936*	26	0.413
M_C-EIIE	1.54	0.000164	0.718	0.067	398	0.688
M_{S1}-EIIE	1.50	0.000476	0.685	0.000364	107	0.401
M_{S2}-EIIE	2.72	0.000354	0.706	0.874	6	0.389
Weight Combination: $w_R = 0.7, w_V = 0.2, w_E = 0.1$						
M_C	0.000103	0.000917	0.765*	0.123	1048*	0.303
M_{S1}	0.000123	0.000957	0.940*	0.315*	226*	0.298
M_{S2}	**0.000142**	0.000896	0.786*	0.417*	9*	0.330
M_C-EIIE	7.61e-05	0.000716	0.847	0.125	802	0.326
M_{S1}-EIIE	5.72e-05	0.000451	0.693	0.0048	291	0.461
M_{S2}-EIIE	0.000112	0.000836	0.701	0.257	1781	0.509
Weight Combination: $w_R = 0.2, w_V = 0.7, w_E = 0.1$						
M_C	3.21e-05	0.000431	0.675*	0.275*	722*	0.348
M_{S1}	7.15e-05	0.000551	0.780*	0.392*	893*	0.485
M_{S2}	**7.30e-05**	0.000451	0.639	0.303*	31	0.450

Table 3 – continued from previous page

Model	Reward	Returns	Volatility	ESG	Turnover	MDD
M_C-EIIE	4.33e-05	0.000450	0.707	0.0311	545	0.362
M_{S1}-EIIE	4.41e-05	0.000470	0.711	0.0595	508	0.662
M_{S2}-EIIE	5.92e-05	0.000504	0.644	0.177	509	0.465
Weight Combination: $w_R = 0.45, w_V = 0.45, w_E = 0.1$						
M_C	9.65e-05	0.000571	0.710*	0.284*	539*	0.374
M_{S1}	7.43e-05	0.000638	0.742*	0.0268*	860*	0.460
M_{S2}	**0.000115**	0.000674	0.694	0.360*	20*	0.387
M_C-EIIE	7.69e-05	0.000706	0.734	0.0566	399	0.320
M_{S1}-EIIE	6.01e-05	0.000510	0.682	0.00638	204	0.367
M_{S2}-EIIE	9.04e-05	0.000670	0.664	0.217	1145	0.485
$1/N$ **portfolio**						
$1/N$	-	0.000439	0.684	6.66e-16*	11	0.410
GMV portfolio						
GMV	-	0.000288	**0.570**	0.299	584	0.491

Next to all performance metrics, an important part of this research is centred around whether a relatively high weight assignment to a certain metric in the multivariate reward function results in a relatively good performance for the given metric during the test period. In Table 3 we observe that for the DRL models, the highest or lowest values for average daily returns, volatility and ESG scores result from the models where $w_R = 1$, $w_V = 1$ and $w_E = 1$ respectively. For all models in between, we observe that for several models there is a positive relationship between the weight assigned to a metric and its performance, except for some models. The DRL models using the EIIE neural network architecture, however, can generally not translate a higher weight assignment w_E in a higher daily average portfolio ESG performance as observed by for instance the $w_V = 0.5$ and $w_E = 0.5$ model and the $w_R = 0.7$, $w_V = 0.2$ and $w_E = 0.1$ model, except for the M_{S2} model.

The relationship between the weights assignment in the reward function and the relative performance for each portfolio performance metric is not completely linear, as the training process remains quite complex given the difference in complexity to optimise the various metrics. For example, the training of the M_C model using the proposed neural network structure for $w_R = 0.5$ and $w_E = 0.5$ is dominated by the ESG gradient component, resulting in a lower average daily return than the M_C model for $w_R = 0.2$, $w_V = 0.7$ and $w_E = 0.1$ using the proposed neural network structure. Ideally, metrics have similar complexity in terms of gradients such that the relationship between the assigned weight by the agent in the reward function and the performance during the test period becomes more linear, hence the reason why we included three models where $w_E = 0.1$, since at least for the DRL models using the proposed neural network, this metric is quite dominant in the training procedure.

Between the M_C and M_{S1} models using the proposed and EIIE neural network architecture, there is no clear pattern in the differences in performance for the three performance metrics during the test period. For most models, the performance is quite similar in terms of average daily reward but due to different metrics that perform relatively well. For the models with $w_R = 0.5$ and $w_E = 0.5$ using the proposed neural network architecure for instance, M_C has a much higher average daily ESG score, but M_{S1} performs better in terms of average daily returns and volatility. Similar cases can be observed for the EIIE counterpart. Given the fact that the M_C and M_{S1} (when compared for a given neural network

structure) essentially use the same type (historical daily returns) and dimension (either a matrix using the proposed and a vector using the EIIE neural network structure) may clarify why there is no consistent difference in terms of performance.

The M_{S1} model was defined with the intuition that the model optimises one metric at a time, making the gradient less complex. In addition, we had the belief that the gradient was not 'disturbed' by the - sometimes more dominant - gradient of other metrics. Also, we expected that this would potentially solve the potential issue of the opposite direction of the return and volatility gradient component, which could potentially cancel out the total gradient. However, it seems that this method does not result in a significant or consistent difference in performance for certain metrics between the M_C model. After all, for the DRL models using the proposed neural network structure, if the ESG performance is optimised without the interference of the other two metrics using the M_{S1} model, the gradient is still considerably large. This might result in the ESG performance being optimised significantly faster than the other two metrics and dominate the training of the neural network parameters in that sense.

Regarding the output of the DRL models, which are the portfolio weights, we observe that for the DRL models with the highest w_R, the Technology ETF receives the highest weight on average during the test period, followed by the Retail ETF. This is in line with the observations in Section 2 regarding the data characteristics of the considered ETFs. For the M_{S2} models with $w_V = 1$ as well as the GMV portfolio, the Bonds ETF receives the highest weight on average. This is quite surprising given the relatively high standard deviation and kurtosis that can be observed in Section 2. However, generally, bonds are used in combination with stocks to diversify a portfolio by institutional investors, which may explain the high average weight during the test period. For the DRL models with a relatively high w_E, the Health and Consumer ETFs receive a relatively high average weight during the test period, which is in line with the data characteristics as described in Section 2.

For the model where $w_E = 1$, having a separate DRL model is not completely necessary given the constant data which makes the optimisation procedure relatively straightforward. Due to the limited data on ESG scores that are available through the S&P 500 exchange, this process was relatively simple. However, as Gyönyörová et al. (2023) conclude, the ESG data is likely to become more complex in the coming years, where not only the metrics that are used will be more different across industries but also the methods used to quantify certain ESG metrics are becoming increasingly complex to conduct and regulate. Therefore, a separate DRL model to optimise the ESG performance of a portfolio will become more relevant as the data availability increases.

4.3 Volatility optimisation analysis

An important result from the previous section is the performance of the M_{S2} models for $w_V = 1$ in terms of average daily portfolio volatility. For these models, the original environment (the price tensor containing historical daily returns data) is transformed to the EWMA-based volatility estimates for several historical days. For the other models, the neural network aims to find a relationship between the price tensor and the weights that aim to minimise the portfolio volatility. This is quite challenging given the relatively complex gradient of the volatility component of the reward function and the challenge

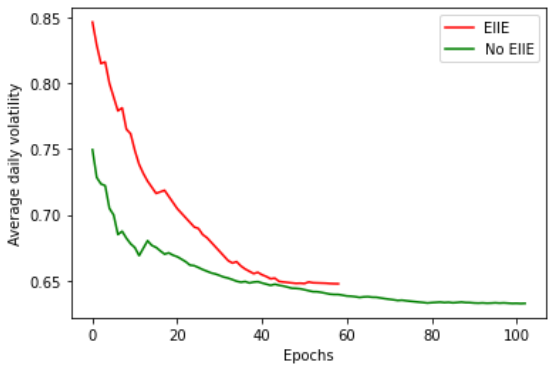
of finding a relationship between historical returns data and the portfolio weights that aim to minimise portfolio volatility.

Despite the relatively good performance of the M_{S2} models for $w_V = 1$ in terms of average daily portfolio volatility compared to the other DRL models, the GMV portfolio still performs best in terms of portfolio volatility. Despite this, we believe that the potential of using DRL models to minimise portfolio volatility can be as high as the usage of DRL models to maximise portfolio returns since this is a relatively unexplored field of research and because of the smooth training procedure described in the remainder of this section.

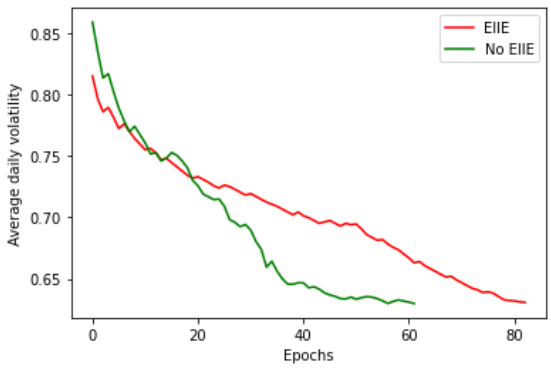
In Figure 3 we show the training process of each of the M_{S2} models for $w_V = 1$ using both the proposed and EIIE neural network architecture, based on the entire training data set and the set where the 5% lowest daily returns are excluded. The progress is shown through the average daily portfolio volatility during the validation period. We observe that for all models the training process results in a model that realises relatively low average daily portfolio volatility during the validation period. Due to the difference in learning rate, the training process either realises more or less reduction in average daily portfolio volatility in the first number of training iterations. As such, the EIIE M_{S2} model for $w_V = 1$ that is trained using the training data set where the lowest 5% of returns are excluded trains relatively at a slow rate compared to the other models. Due to the stopping criteria of our training procedure, not every model requires the same number of training iterations during the training procedure, which is highly related to the used learning rate that follows from the Bayesian hyperparameter optimisation procedure.

Liang et al. (2018) evaluate the performance of several DRL models to maximise portfolio returns during the testing period. Next to this, they also formulate a model that is designed to maximise the portfolio return during the test period while also reducing portfolio volatility. In essence, they penalise investing in assets that increase portfolio volatility, similar to our approach in the reward function. They conclude that the extra restriction makes the reward function - and subsequently, the gradient - too complex to realise substantial returns while also reducing the portfolio risk. However, observing our training procedure in Figure 3, the complex gradient of the volatility component seems to be no issue. Also, the combination of several metrics in the reward function, as seen through several examples in Table 3, is not an issue per se. In Table 3, the M_C and M_{S1} models for $w_V = 1$ perform significantly less than the M_{S2} models in terms of average daily portfolio volatility at a 5% significance level using the paired t test, even when no returns or ESG gradient component is used during training. The issues seem to be related to the relation that the DRL model aims to find between the historical daily returns (the input to the neural network) and the portfolio weights that aim to minimise portfolio volatility.

We believe that machine learning-based models, like our proposed DRL models, require additional support to improve performance as the objective (reward function) becomes more complex. Especially when portfolio volatility is included in the reward function, as this metric proves to be relatively challenging to optimise through a DRL model.



(a) M_{S2} models using the proposed and EIIE neural network structure for $w_V = 1$ using entire training set



(b) M_{S1} models using the proposed and EIIE neural network structure for $w_V = 1$ using adjusted train set

Figure 3: Progression of the training of the M_{S2} models through the average daily portfolio volatility during the validation set for $w_V = 1$, for the proposed and EIIE neural network architecture both using the entire training set and the training set where 5% of the lowest daily returns are excluded

4.4 Analysis of model performances using adjusted training set

We formulate a hedging strategy to limit our lower tail risk during extreme downside market circumstances. Before analysing the hedging results, we first analyse the performance of our DRL models using the adjusted train set. Instead of training our models on the entire data set, we remove 5% of the training days where the daily returns are the lowest. As such, we limit the training of our models during days of extreme downside market circumstances. In Table 4 we show the performance of the DRL models using the proposed and EIIE neural network structure during the test period using the adjusted training set. Compared to Table 3, the M_{S2} has the best performance in terms of average daily reward for fewer weight combinations, namely not for the models where $w_V = 0.5$ and $w_E = 0.5$ and finally for $w_R = 0.45$, $w_V = 0.45$ and $w_E = 0.1$. Generally, the average daily returns are most affected by the adjustment of the training set, which are generally lower compared to the figures of Table 3.

In terms of volatility, most DRL models yield lower average daily volatility figures compared to the corresponding values of Table 3. One possible explanation for this, which could also explain the difference in terms of average daily returns, is that the model is likely to adjust the portfolio weights more significantly when some historical data in the price tensor can be classified as extreme downside events when the model is trained on the entire data set. This might be because of the training exposure to such events and the major rebalancing of the portfolio weights that are required to maximise returns given the extreme downside market events, which are likely to increase the portfolio volatility. In case our models are trained using the adjusted training set, the model will be more flexible in the sense that it does not apply the same sense of urgency to adjust the portfolio weights during extreme downside market events and maximise portfolio returns, which makes the portfolio less volatility by reducing the portfolio exposure to major shifts in daily returns because of significant portfolio weights rebalancing.

Against our expectations, the models that are trained without the set of the 5% lowest daily returns

days perform better in terms of average daily volatility for the majority of the models. Despite this observation, we continue to use the hedging strategy complementary to the models in Table 4 aiming to reduce the average daily portfolio volatility even further, resulting in two types of model groups for each of the weight combination, where one is more relevant for maximising portfolio returns (the models from Table 3) and one that is more relevant to minimise the portfolio volatility, which eventually are the models in Table 6, that is discussed in Section 4.5.

In terms of average daily ESG scores and total turnover, there is no consistent trend visible when comparing the two tables. Finally, the MDD value for the models is generally lower for the models using the adjusted training set compared to the models of Table 3, in line with the decrease in average daily volatility for most of the models.

Table 4: Overview of the performance of all DRL models using the proposed and EIIE neural network architecture as well as the $1/N$ and GMV portfolio, using the training dataset where the train days within the 5% lowest daily returns group are excluded, by showing the average reward value, returns, volatility, ESG scores and the total turnover value. MDD stands for maximum drawdown and per weight combination the highest reward value is made bold, whereas for the other metrics, the bold figures represent the highest or lowest value across all weight combinations and models. Finally, the '$*$' indicates that p$<$0.05 for the paired t test between the proposed model and its EIIE counterpart

Metrics						
Model	**Reward**	**Returns**	**Volatility**	**ESG**	**Turnover**	**MDD**
Weight Combination: $w_R = 1/3, w_V = 1/3, w_E = 1/3$						
M_C	0.000116	0.000473	0.671*	0.0803*	427*	0.349
M_{S1}	0.000181	0.000399	0.766*	0.927*	1038*	0.396
M_{S2}	**0.000237**	0.000591	0.693*	0.612	319*	0.346
M_C-EIIE	0.000178	**0.00108**	0.909	0.144	528	0.354
M_{S1}-EIIE	0.000126	0.000535	0.689	0.00399	322	0.375
M_{S2}-EIIE	0.000235	0.000563	0.675	0.623	805	0.417
Weight Combination: $w_R = 1, w_V = 0, w_E = 0$						
M_C	0.000943	0.000943	0.891*	0.342*	937*	**0.283**
M_{S1}	**0.00108**	0.00108	0.790*	-0.0713*	949*	0.301
M_{S2}	0.00108	0.00108	0.790*	-0.0713*	949*	0.301
M_C-EIIE	0.00101	0.00101	0.870	0.110	2537	0.485
M_{S1}-EIIE	0.00103	0.00103	0.816	0.111	2409	0.566
M_{S2}-EIIE	0.00103	0.00103	0.816	0.111	2409	0.566
Weight Combination: $w_R = 0, w_V = 1, w_E = 0$						
M_C	0.686*	0.000207	0.686*	0.0131	783*	0.622
M_{S1}	0.680	0.000388	0.680	0.0406*	512*	0.443
M_{S2}	**0.629**	0.000324	0.629	0.131*	44*	0.465
M_C-EIIE	0.696	0.000430	0.696	0.0210	604	0.398
M_{S1}-EIIE	0.688	0.000671	0.688	0.0169	326	0.383
M_{S2}-EIIE	0.631	0.000278	0.631	-0.0208	**4.56e-06**	0.527
Weight Combination: $w_R = 0, w_V = 0, w_E = 1$						
M_C	**1.75**	0.000381	0.963	**1.75**	19	0.304
M_{S1}	1.71	0.000348	0.946	1.71	73	0.306
M_{S2}	1.75	0.000381	0.963	1.75	19	0.304
M_C-EIIE	1.75	0.000371	0.959	1.75	17	0.299
M_{S1}-EIIE	1.70	0.000342	0.909	1.70	66	0.310
M_{S2}-EIIE	1.75	0.000371	0.959	1.75	17	0.299

Table 4 – continued from previous page

Model	Reward	Returns	Volatility	ESG	Turnover	MDD
Weight Combination: $w_R = 0.5, w_V = 0.5, w_E = 0$						
M_C	0.000789	0.000616	0.723*	0.0450*	538*	0.410
M_{S1}	0.000674	0.000519	0.725*	0.0740*	567	0.378
M_{S2}	**0.00106**	0.000701	0.667	0.0259*	477*	0.365
M_C-EIIE	0.000656	0.000469	0.685	0.00316	103	0.398
M_{S1}-EIIE	0.000598	0.000308	0.692	0.0116	642	0.560
M_{S2}-EIIE	0.000990	0.000654	0.672	0.0421	1205	0.504
Weight Combination: $w_R = 0.5, w_V = 0, w_E = 0.5$						
M_C	0.000255	0.000370	0.965*	1.75*	3.07*	0.302
M_{S1}	0.000164	0.000536	0.710*	0.248*	821*	0.377
M_{S2}	0.000343	0.000724	0.663*	0.84*	477*	0.297
M_C-EIIE	0.000226	0.000791	0.716	0.0487	658	0.425
M_{S1}-EIIE	0.000178	0.000625	0.692	0.0111	385	0.355
M_{S2}-EIIE	**0.000357**	0.000706	0.628	0.931	1207	0.408
Weight Combination: $w_R = 0, w_V = 0.5, w_E = 0.5$						
M_C	1.92*	0.000217	0.945*	0.774*	66*	0.292
M_{S1}	**2.92***	0.000379	0.943*	1.70*	121*	0.328
M_{S2}	2.77	0.000347	0.719*	0.958*	27	0.381
M_C-EIIE	1.53	0.000519	0.743	0.110	1231	0.479
M_{S1}-EIIE	1.48	0.000529	0.696	0.00776	339	0.409
M_{S2}-EIIE	2.83	0.00329	0.696	0.882	5.62	0.393
Weight Combination: $w_R = 0.7, w_V = 0.2, w_E = 0.1$						
M_C	5.79e-05	0.000664	0.814*	-0.130*	1199*	0.423
M_{S1}	5.46e-05	0.000580	0.748*	-0.193*	1193*	0.382
M_{S2}	0.000128	0.000857	0.705*	0.149*	665*	0.324
M_C-EIIE	5.59e-05	0.000520	0.832	0.0944	2570	0.359
M_{S1}-EIIE	5.94e-05	0.000458	0.688	0.000322	93	0.440
M_{S2}-EIIE	**0.000135**	0.000815	0.713	0.248	1687	0.511
Weight Combination: $w_R = 0.2, w_V = 0.7, w_E = 0.1$						
M_C	3.59e-05	0.000295	0.771*	0.339*	623*	0.408
M_{S1}	3.00e-05	0.000500	0.706*	-0.271*	856*	0.375
M_{S2}	**6.18e-05**	0.000479	0.643	0.254*	199*	0.401
M_C-EIIE	5.27e-05	0.000544	0.693	0.00533	242	0.444
M_{S1}-EIIE	5.15e-05	0.000503	0.680	0.0199	288	0.512
M_{S2}-EIIE	5.74e-05	0.000439	0.633	0.185	482	0.465
Weight Combination: $w_R = 0.45, w_V = 0.45, w_E = 0.1$						
M_C	**0.000117**	0.000653	0.760*	0.618*	1118*	0.362
M_{S1}	8.72e-05	0.000564	0.676*	0.173*	546*	0.354
M_{S2}	0.000102	0.000668	0.667	0.202	429*	0.359
M_C-EIIE	5.58e-05	0.000478	0.694	0.00388	238	0.393
M_{S1}-EIIE	0.000104	0.00104	0.806	0.133	348	0.317
M_{S2}-EIIE	9.94e-05	0.000627	0.663	0.216	1085	0.479
$1/N$ **portfolio**						
$1/N$	-	0.000439	0.684	6.66e-16*	11	0.410
GMV portfolio						
GMV	-	0.000288	**0.570**	0.299	584	0.491

4.5 Analysis of hedging perfomances

In Section 4.4, we observed that training the model on the adjusted training set reduces the average daily portfolio volatility for the majority of the models. To further reduce the average daily portfolio

volatility, we switch to the proposed lower tail dependence-based hedging strategy as soon as the strategy is triggered. In short, as soon as our proposed model encounters a daily return that is below -0.01, we switch to our hedging model the following day for a duration of 5 days, as explained in more detail in Section 3.2.2.

In Table 5 we show the lower tail dependencies of all pairs of the assets we use in our portfolio. The Financial and Bonds ETFs show the least association between extremely negative movements with the other ETFs, but do so together, which can possibly be explained by the high dependence on daily returns with the interest rates. Based on the Student's t copula, quite some pairs of assets show no lower tail dependence, which in theory can be used to limit the lower tail risk. A strong decline in returns of one asset for such a pair during extreme downside market events is generally not associated with a similar observation for the other asset. Therefore, we believe that temporarily changing to a portfolio where we go long in the assets that show the least lower tail exposure and short in the assets that show the highest lower tail exposure will reduce the average portfolio volatility during the test period.

Table 5: Lower tail dependencies of all combinations of all ETFs, based on the 70% train data using Student's t copula to model the dependencies. E&T stands for entertainment and telecom

	Tech	Industrial	Financial	Energy	Health	Consumer	E&T	Materials	Retail	Bonds
Tech	1.000	-	-	-	-	-	-	-	-	-
Industrial	0.656	1.000	-	-	-	-	-	-	-	-
Financial	0.000	0.000	1.000	-	-	-	-	-	-	-
Energy	0.686	1.30e-05	0.000	1.000	-	-	-	-	-	-
Health	0.000	0.456	0.000	0.000	1.000	-	-	-	-	-
Consumer	0.392	0.321	0.000	0.000	0.633	1.000	-	-	-	-
E&T	0.693	0.726	8.82e-06	0.661	0.548	0.378	1.000	-	-	-
Materials	0.627	0.000	0.000	0.000	0.668	0.526	0.574	1.000	-	-
Retail	0.415	0.733	0.000	0.730	0.543	0.414	9.69e-06	0.000	1.000	-
Bonds	0.186	0.201	0.609	0.074	0.000	0.000	0.000	0.000	0.000	1.000

Using the hedging strategy and the obtained lower tail dependence matrix in Table 5, we show the performance of the DRL models that are complemented by the hedging strategy in Table 6. For each model, we show the results similarly as Table 3 and Table 4, but we test whether there is a significant difference with the corresponding model in Table 3 using the paired t test at a 5% significance level, rather than testing the difference between the models using the proposed and EIIE neural network architecture. For the majority of the models, we realise a significant decrease in average daily portfolio volatility compared to the models in Table 3 that are trained on the entire training data set.

Similar to the models in Table 3 and Table 4, the M_{S2} models (either using the proposed or EIIE neural network architecture) realise the highest daily average rewards, except for the DRL models where $w_E = 1$ as well as for $w_R = 0.2$, $w_V = 0.7$ and $w_E = 0.1$. In terms of average daily returns, the performance is worse for most models, where the majority of models for $w_V = 1$ and $w_E = 1$ realise negative daily average returns, making a strategy of doing nothing a more profitable strategy. The hedging strategy does not take into account the weights that are assigned by the agent per metric, resulting in a major decrease in average daily portfolio returns in particular for the models with the highest weight assigned to returns through w_R. If we, for example, look at the M_{S1} model for $w_R = 1$ in Table 3, we observe a daily average portfolio return of 0.00115 and volatility of 0.971. In Table 6, the corresponding model

yields a daily average portfolio return of 0.000251 (around 78% reduction) and volatility of 0.672 (around 31% reduction), making the reduction in average daily portfolio volatility relatively expensive.

Since the hedging strategy is designed to reduce the average daily portfolio volatility by limiting the lower tail risk, the portfolio returns and ESG score metrics are more or less neglected in the optimisation decision-making during a switch to the hedging strategy. In terms of average ESG scores, this also leads to a significant reduction for the majority of models. For some models, the hedging strategy significantly improves the average ESG scores, which is more a coincidence due to a relatively large position in a well-performing ESG asset due to a relatively low lower tail exposure, which leads to an unintended improvement in the average daily portfolio ESG score.

Due to the major rebalancing that is required to switch to the hedging strategy as soon as the model is triggered, the total turnover is significantly higher for most models. In case this research would be extended to analyse the impact of transaction costs, this would give an additional negative impact on the total returns of the agent. In terms of maximum drawdown (MDD), the models of Table 6 do not strictly outperform the corresponding models of Table 3. In terms of the range where the MDDs fall within for Table 6 (between 0.175 and 0.543) and Table 3 (between 0.280 and 0.688), the models from Table 6 fall within a more favourable range. Also, in terms of average MDD values, the models in Table 6 (0.370) outperform the models in Table 3 (0.393), which is however not significant at a 5% significance level using the paired t test.

Table 6: Overview of the performance of all DRL models using the proposed and EIIE neural network architecture as well as the $1/N$ and GMV portfolio, which are complemented using the hedging strategy to switch from the models that are trained on the adjusted train set to limit the lower tail risk, by showing the average reward value, returns, volatility, ESG scores and the total turnover value. MDD stands for maximum drawdown and per weight combination the highest reward value is made bold, whereas for the other metrics, the bold figures represent the highest or lowest value across all weight combinations and models. Finally, the '$*$' indicates that p$<$0.05 for the paired t test between the value in this table compared to the equivalent value in Table 3

Metrics						
Model	**Reward**	**Returns**	**Volatility**	**ESG**	**Turnover**	**MDD**
Weight Combination: $w_R = 1/3, w_V = 1/3, w_E = 1/3$						
M_C	5.20e-05	0.000261	0.602*	-0.111*	675*	0.346
M_{S1}	0.000121	0.000244	0.671	0.357*	1165*	0.324
M_{S2}	0.000122	0.000244	0.625*	0.290*	321*	0.341
M_C-EIIE	-1.47e-05	0.000431	0.747*	0.0935*	607*	0.245
M_{S1}-EIIE	8.44e-05	-3.28e-05	0.622*	-0.0312*	570*	0.489
M_{S2}-EIIE	**0.000151**	0.000358	0.579*	0.203*	1012*	0.259
Weight Combination: $w_R = 1, w_V = 0, w_E = 0$						
M_C	0.000251	0.000251	0.737*	0.476*	1170*	0.425
M_{S1}	**0.000446**	**0.000446**	0.672*	0.275*	1075*	0.216
M_{S2}	0.000446	0.000446	0.672*	0.275	1075*	0.216
M_C-EIIE	-6.84e-05	-6.84e-05	0.681*	0.0558*	1930*	0.532
M_{S1}-EIIE	0.000129	0.000129	0.663*	0.0536*	1852*	0.426
M_{S2}-EIIE	0.000129	0.000129	0.663*	0.0536*	1852*	0.426
Weight Combination: $w_R = 0, w_V = 1, w_E = 0$						
M_C	0.593*	8.81e-05	0.593*	-0.00618*	928*	0.321
M_{S1}	0.606*	1.97e-05	0.606*	-0.00867*	660*	0.392

Table 6 – continued from previous page

Model	Reward	Returns	Volatility	ESG	Turnover	MDD
M_{S2}	0.542	-3.96e-05	0.542	0.0676*	516*	0.297
M_C-EIIE	0.622*	-1.39e-05	0.622*	-0.0352*	801*	0.362
M_{S1}-EIIE	0.607*	-1.48e-05	0.607*	-0.0164*	593*	0.422
M_{S2}-EIIE	**0.527***	-0.000192	0.527*	0.0447*	632*	0.496
Weight Combination: $w_R = 0, w_V = 0, w_E = 1$						
M_C	0.781*	-0.000262	0.779*	0.781*	618*	0.505
M_{S1}	**1.05***	-0.000195	0.794*	**1.05***	**11***	0.531
M_{S2}	0.781*	-0.000262	0.779*	0.781*	618*	0.505
M_C-EIIE	0.780*	-0.000274	0.779*	0.780*	615*	0.504
M_{S1}-EIIE	1.03*	-0.000204	0.792*	1.03*	41*	0.526
M_{S2}-EIIE	0.780*	-0.000274	0.779*	0.780*	615*	0.504
Weight Combination: $w_R = 0.5, w_V = 0.5, w_E = 0$						
M_C	-0.000378*	-0.000171*	0.633*	0.0423*	743*	0.495
M_{S1}	0.000140	6.63e-05	0.645*	0.0260*	787*	0.321
M_{S2}	0.000499*	0.000421	0.575*	0.0497*	722*	0.202
M_C-EIIE	0.000362	0.000242	0.622*	0.00711	390*	0.194
M_{S1}-EIIE	0.000486	6.29e-05	0.610*	-0.0554*	761*	0.507
M_{S2}-EIIE	**0.000674**	0.000358	0.559*	0.0339*	1245	0.282
Weight Combination: $w_R = 0.5, w_V = 0, w_E = 0.5$						
M_C	8.07e-05	-1.18e-05	0.781*	1.03*	610*	0.537
M_{S1}	8.22e-05	0.000234	0.642*	0.0144*	946*	0.338
M_{S2}	8.66e-05	9.31e-05	0.629*	0.644*	763*	0.312
M_C-EIIE	2.77e-05	3.30e-05	0.616*	-0.00522*	888*	0.510
M_{S1}-EIIE	7.76e-05	0.000386	0.630*	-0.00916	587*	0.236
M_{S2}-EIIE	**0.000175**	0.000333	0.597*	0.478*	1282*	0.325
Weight Combination: $w_R = 0, w_V = 0.5, w_E = 0.5$						
M_C	1.30*	6.30e-05	0.781*	0.165*	655*	0.281
M_{S1}	1.92	0.000230	0.767*	0.762*	664*	0.234
M_{S2}	**2.27***	0.000587	0.630*	0.487*	478*	0.190
M_C-EIIE	1.73*	**0.000592**	0.647*	0.0411*	1210*	0.281
M_{S1}-EIIE	1.71*	-6.26e-06	0.628*	-0.0171*	569*	0.475
M_{S2}-EIIE	2.24	0.000170	0.608*	0.410*	508*	0.343
Weight Combination: $w_R = 0.7, w_V = 0.2, w_E = 0.1$						
M_C	4.77e-05	0.000283	0.697*	-0.103*	1267*	0.378
M_{S1}	-2.02e-05	0.000178	0.654*	-0.111*	1197*	0.225
M_{S2}	**5.79e-05**	0.000315	0.624*	0.0279*	833*	0.275
M_C-EIIE	2.46e-05	-0.000150	0.719*	0.0486*	226*	0.251
M_{S1}-EIIE	-1.69e-05	3.03e-06	0.630*	-0.0115*	388*	0.430
M_{S2}-EIIE	4.97e-05	0.000307	0.585*	-0.0846*	1489*	0.295
Weight Combination: $w_R = 0.2, w_V = 0.7, w_E = 0.1$						
M_C	**5.62e-05**	0.000201	0.682	0.281*	888*	0.306
M_{S1}	-9.93e-05	0.000250	0.629*	-0.0790*	919*	0.300
M_{S2}	4.32e-05*	0.000425	0.578*	-0.0540*	557*	0.252
M_C-EIIE	1.12e-06	1.44e-05	0.634*	0.000435*	242*	0.336
M_{S1}-EIIE	-1.80e-05	-0.000229	0.588*	-0.00388*	604*	0.543
M_{S2}-EIIE	2.04e-05	-2.91e-05	0.547*	-0.124*	830*	0.344
Weight Combination: $w_R = 0.45, w_V = 0.45, w_E = 0.1$						
M_C	4.17e-05	0.000160	0.656*	0.249*	1176*	0.278
M_{S1}	5.44e-05	0.000372	0.609*	-0.0571*	732	**0.175**
M_{S2}	6.01e-05	0.000346	0.599*	-0.0369*	679*	0.356
M_C-EIIE	5.11e-06	0.000165	0.637*	-0.0277*	468*	0.353
M_{S1}-EIIE	-8.59e-05	1.60e-05	0.678	-0.0297*	828*	0.439

Table 6 – continued from previous page

Model	Reward	Returns	Volatility	ESG	Turnover	MDD
M_{S2}-EIIE	**0.000117**	0.000447	0.563*	-0.0823*	1167	0.284
$1/N$ portfolio						
$1/N$	-	0.000205	0.645*	-0.0545*	324*	0.405
GMV portfolio						
GMV	-	-0.000106	**0.516***	0.0171	726	0.421

In general, the models that show the highest average daily volatility in Table 3 achieve the highest reduction in average daily portfolio volatility through the hedging strategy. Using the hedging strategy to complement our models, we realise the lowest average daily volatility across all the discussed models for the GMV model, followed by the M_{S2} model using the EIIE neural network architecture.

In conclusion, by making use of a hedging strategy to complement the DRL models that are trained without the training days that fall within the group of the lowest 5% of returns, we obtain for the majority of models a significant reduction in average daily portfolio volatility during the test period. However, due to the focus on reducing portfolio volatility during the switch to the hedging strategy, the other metrics perform worse since there is no focus on these metrics during the hedging strategy. The ability to focus on each metric in line with the weight that is assigned to the metric by the agent is therefore less successful, leading to an overall worse performance if we take into account the other metrics. The agent should decide what the reduction in average daily portfolio volatility is worth in terms of the reduction in performance of the other metrics and whether this is worth it or not.

5 Conclusion and discussion

In this research, we analyse the performances of several deep reinforcement learning (DRL) models that can be used for an investment strategy with multiple objectives, namely portfolio returns, volatility and ESG performance. The considered assets consist of nine ETFs based on S&P 500 stocks and one bond ETF. A neural network is trained to derive the portfolio weights (output of neural network) to invest in the ETFs, that aim to maximise the reward of the agent. Based on the weights that an agent assigns to each portfolio performance metric through the reward function, the training of the DRL models is more focused on the metrics with the higher weights.

We formulate three different DRL models, where the first calculates the gradient during the training process over the entire reward function. The second model trains the model using the gradient of the reward function with respect to one of the metrics, with a probability per metric in line with the weights assigned by the agent. The other two metrics are subsequently neglected for the given training iteration. The third model is an 'ensemble' of three different DRL models, where each model can be used for an investment strategy with a single objective, focusing on optimising either the portfolio returns, volatility or ESG performance. Given the weights assigned by the agent, the output of the individual DRL models is combined into a single model. For the individual DRL model of the last model focusing on volatility, we transform the original input data (a matrix of historical daily returns data) to historical EWMA-based volatility estimates per asset.

In general, we observe that there is a positive relationship between the weight assigned by the agent to a certain metric and the actual performance for the corresponding metric if the given DRL model is used for an investment strategy with multiple objectives. Also, we conclude that there is no clear difference in terms of performance between the first two described DRL models in terms of reward in case they are used for an investment strategy with multiple objectives.

For each of the three models, we form two types of neural network architectures. The first is the so-called 'ensemble of identical independent evaluators' (EIIE), which lets the historical data of every asset flow independently of each other through the neural network. For the second neural network architecture, we considered a new approach that allows interaction between all assets inside the neural network since the input data is a matrix of historical data of all assets. This idea is based on the GMV portfolio, which relies on the covariance matrix. Since the covariance matrix includes the relationships among all assets, we tested if translating this intuition would be beneficial in terms of minimising portfolio volatility. When applied to an investment strategy with multiple objectives, we conclude that the DRL models using the proposed neural network architecture do not strictly outperform the EIIE counterparts. This could partly be explained by the fact that both neural network architectures aim to find the same patterns from the same data (through a matrix or split up into vectors).

In terms of model performances, the model that uses an ensemble of individual DRL models performs best in general if it is used for an investment strategy with multiple objectives. We believe that this is mainly due to two reasons, the first being that it uses the ensemble of DRL models that perform best for each of the individual metrics if they are used for an investment strategy with a singular objective. The second reason is that the individual DRL model that can be used for an investment strategy that aims to minimise portfolio volatility uses a transformation of the input data.

Liang et al. (2018) conclude that the addition of a volatility component makes the gradient of the reward function too complex, which is the main reason for the suboptimal results if the DRL model is used for an investment strategy that aims to maximise portfolio returns and reduce portfolio volatility. After transforming the input data from the environment to EWMA-based historical volatility estimates for each asset, the DRL model establishes a more intuitive relationship between the input data of the neural network and the portfolio weights that aim to minimise portfolio volatility. This is beneficial for the minimisation of portfolio volatility when the DRL model is used for an investment strategy that aims to minimise portfolio volatility.

The available and relevant literature on this topic is relatively limited, due to the recent innovations and interest in the field of DRL applications for portfolio management. As such, the majority of previous research was centred on portfolio returns, rather than also focusing on portfolio volatility and other factors such as ESG performance. However, the proposed DRL models that are used for investment strategies with singular or multiple objectives are outperformed by the GMV portfolio in terms of minimising portfolio volatility.

The DRL model which is used for an investment strategy that aims to minimise portfolio volatility by transforming the input data of the neural network to EWMA historical volatility estimates, aims to assign the highest portfolio weights to the assets with the lowest volatility. The GMV portfolio, on the

other hand, directly optimises the portfolio weights to minimise portfolio volatility through the inverse covariance matrix, which includes the relations between assets through the off-diagonal elements. Given the relatively smooth training process of the DRL model that is used for an investment strategy aiming to minimise portfolio volatility, that uses a transformation of the input data to historical EWMA estimates, we believe that future research can be conducted to form DRL models or other machine learning-based models that are used for an investment strategy aiming to minimise portfolio volatility that outperforms the GMV portfolio in terms of minimising portfolio volatility.

In addition, we train the DRL models using both the proposed and EIIE neural network architecture on an adjusted training data set, where the days with the lowest 5% of daily returns are excluded (extreme market downside events), subsequently removing the exposure that each DRL model has during training with extreme market downside events. We use a hedging strategy to complement the DRL models to limit the lower tail risk. Without the use of the hedging strategy, we already observe a decrease in the average daily portfolio for the majority of the models that are used for an investment strategy with singular or multiple objectives, in the case we train our model on the adjusted train data set. In case we complement the DRL models with a hedging strategy, we realise a significant reduction in average daily portfolio volatility for the majority of DRL models that are used for an investment strategy with singular or multiple objectives. The highest improvement in portfolio volatility is obtained for the DRL models that had the highest average daily portfolio volatility without the use of a hedging strategy.

To tackle some limitations of our research and to explore the capabilities of the DRL models further, there are some topics to conduct future research. First, future research can be conducted to include transaction costs, as this has a major impact in real-world applications on the profitability of a trading strategy. Second, in the case more ESG data becomes available, the ESG component of the multivariate rewarded function becomes more relevant and the bonds ETF might be assigned an actual ESG score. Third, future research can be conducted to further explore the capabilities of DRL models that can be used for an investment strategy that aims to minimise portfolio volatility. Supported by the smooth training procedure of the DRL model that aims to minimise portfolio volatility using the adjusted input data for the neural network, through the EWMA historical volatility estimates, one can aim to significantly outperform the GMV portfolio in minimising the portfolio volatility. Finally, future research can be conducted to find a solution to the complex challenge of having either a multivariate reward function that is convenient for training the DRL models or for evaluating the DRL models.

References

Bollen, B. (2015). What should the value of lambda be in the exponentially weighted moving average volatility model? *Applied Economics*, 47(8):853–860.

Caillault, C. and Guegan, D. (2005). Empirical estimation of tail dependence using copulas: application to asian markets. *Quantitative finance*, 5(5):489–501.

de Melo Mendes, B. V. and Lavrado, R. C. (2017). Implementing and testing the maximum drawdown at risk. *Finance Research Letters*, 22:95–100.

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

Gyönyörová, L., Stachoň, M., and Stašek, D. (2023). Esg ratings: relevant information or misleading clue? evidence from the s&p global 1200. *Journal of Sustainable Finance & Investment*, 13(2):1075–1109.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

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

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

Ko, V. and Hjort, N. L. (2019). Model robust inference with two-stage maximum likelihood estimation for copulas. *Journal of Multivariate Analysis*, 171:362–381.

Kole, E., Koedijk, K., and Verbeek, M. (2005). Testing copulas to model financial dependence. *Department of Financial Management, RSM Erasmus University, Rotterdam, The Netherlands*.

Kopocinski (2024). Copula generation and estimation, matlab central. *MATLAB Central*.

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

Longin, F. and Solnik, B. (2001). Extreme correlation of international equity markets. *The journal of finance*, 56(2):649–676.

Lourme, A. and Maurer, F. (2017). Testing the gaussian and student's t copulas in a risk management framework. *Economic Modelling*, 67:203–214.

Markowitz, H. M. (1991). Foundations of portfolio theory. *The journal of finance*, 46(2):469–477.

Merton, R. C. (1972). An analytic derivation of the efficient portfolio frontier. *Journal of financial and quantitative analysis*, 7(4):1851–1872.

Mina, J., Xiao, J. Y., et al. (2001). Return to riskmetrics: the evolution of a standard. *RiskMetrics Group*, 1:1–11.

Pasunuru, R., Guo, H., and Bansal, M. (2020). Dorb: Dynamically optimizing multiple rewards with bandits. *arXiv preprint arXiv:2011.07635*.

Plyakha, Y., Uppal, R., and Vilkov, G. (2012). Why does an equal-weighted portfolio outperform value- and price-weighted portfolios? *Available at SSRN 2724535*.

Selvin, S., Vinayakumar, R., Gopalakrishnan, E., Menon, V. K., and Soman, K. (2017). Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE.

Sklar, A. (1973). Random variables, joint distribution functions, and copulas. *Kybernetika*, 9(6):449–460.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40.

Zhang, Z., Zohren, S., and Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40.

6 Appendix

6.1 Data

Full list of stocks used per industry: Tech: Apple, Autodesk, Microsoft, Oracle and Adobe. Heavy Industry: Boeing, Ford, General Electric, Lockheed Martin, Deere & Company. Financial: JPMorgan, Wells Fargo, Marsh & McLennan, Globe Life and Bank of America. Energy: Chevron, Baker Hughes, ConocoPhillips, Williams Companies and NextEra Energy. Health: Abbott, CVS Health, Bristol-Myers Squibb, Johnson & Johnson and Pfizer. Consumer: Hershey's, Coca-Cola, Kellogg's, Molson Coors and General Mills. Entertainment/Communication: Disney, Verizon, AT&T, Comcast and Hasbro. Materials: Nucor, Ball Corporation, PPG Industries, International Paper and Newmont. Retail: Home Depot, Walmart, Lowe's, Walgreens Boots Alliance and TJX Companies. Bonds: 5 Year Treasury, 10 Year Treasury and 30 Year Treasury.

6.2 Bayesian hyperparameter optimisation procedure

We define our reward function as the objective function in our optimisation procedure, which we aim to maximise through the Bayesian optimisation procedure. We denote the reward function as $R(X)$, where we aim to maximise the reward as a function of our hyperparameters $X \in \mathbf{X}$ to find the optimal hyperparameters X^*:

$$X^* = \arg\max_{X \in \mathbf{X}} R(X) \tag{23}$$

Next, we follow Wu et al. (2019) to define a Gaussian process around our objective function. We have defined our reward function R, but this is not of a closed form of parameters like λ, ρ and ϵ for example. Therefore, we aim to approach the relation between X and R using $f(X)$, which follows a Gaussian process. We denote the Gaussian process as $\mathcal{GP}$ and we assume a diffuse prior for the mean $m(X)$. In addition, we use the exponential square function as covariance function $k(X, X')$. Together the Gaussian process is denoted as:

$$f(X) \sim \mathcal{GP}(m(X), k(X, X')) \tag{24}$$

$$m(X) = 0 \tag{25}$$

$$k(X, X') = \exp\left(-\frac{1}{2}(X - X')'(X - X')\right) \tag{26}$$

After defining the Gaussian process, we use a random set of $X = [X_1, \ldots, X_s]$ to form the initial sample set of size s as $S = [(X_1, R(X_1), \ldots, (X_s, R(X_s)]$. The values $R(X_1), \ldots, R(X_s)$ result from the original objective function, which is the reward value after 50 training iterations. We assume that the reward values follow a multivariate normal distribution $f_s(X) \sim N(0, K)$, where $K = k(X, X')$. Each of the randomly selected sets of hyperparameters of X should be within our outcome space $\mathbf{X}$.

We continue to follow Wu et al. (2019), to determine the updated distribution after sampling a new value $s + 1$. We set $k = [k(X_{s+1}, X_1), \ldots, k(X_{s+1}, X_s)]$ as the vector of covariances. We define $S' = [R(X_1), \ldots, R(X_s)]$, which is used to update the mean of the distribution. Since we assume that the objective function follows a Gaussian process, we can update the distribution belief as follows:

$$\mu_{s+1}(X_{s+1}) = k' K^{-1} S \tag{27}$$

$$\sigma^2_{s+1}(X_{s+1}) = -k' K^{-1} k + k(X_{s+1}, X_{s+1}) \tag{28}$$

The new value $s+1$ follows $f_{s+1}(X_{s+1}) \sim N(\mu_{s+1}(X_{s+1}), \sigma^2_{s+1}(X_{s+1}))$ as distribution.

After defining the procedure of finding the posterior distribution after a new considered set of hyperparameters, we choose an acquisition function to assess whether a new set of hyperparameters is expected to improve the performance of the DRL model when it is applied to an investment strategy. There are several acquisition functions, with probability of improvement (PI) and expected improvement (EI) being the most well-known. According to Wu et al. (2019), EI is more suitable for a balance of exploration and exploitation, whereas PI only considers exploration. In addition, they claim that EI is less prone to get stuck in a local optimum. Therefore, we choose EI as our acquisition function.

We define $X^* = \arg\max_{X_i \in S} R(X_i)$ as the set of hyperparameters that yields the highest reward from our initial sampled data in S, where we denote the highest reward as $R(X^*)$. After considering a random new set of hyperparameters $X_{s+1} \in \mathbf{X}$, we can aim to improve our hyperparameter distribution. Using EI, we aim to maximise the degree of improvement $I(x)$, which can be defined as:

$$X = \arg\max E(max\{0, R(X_{s+1}) - R(X^*)\}) \tag{29}$$

When considering $X_{s+1} \in \mathbf{X}$, we are interested if $E(\{0, R(X_{s+1}) - R(X^*)\}) \geq 0$. Wu et al. (2019) show that this expression is equivalent to:

$$\sigma_{s+1}(X_{s+1})(Z_{s+1}\Phi(Z_{s+1}) + \phi(Z_{s+1})) \geq 0 \tag{30}$$

Where Φ and ϕ are the CDF and PDF of a standard normal distribution respectively. Using the mean and variance from $f_{s+1}(X_{s+1})$, Z_{s+1} is equal to:

$$Z_{s+1} = \frac{\mu_{s+1}(X_{s+1}) - R(X^*)}{\sigma_{s+1}(X_{s+1})} \tag{31}$$

This effectively means that if we expect the new set of hyperparameters to increase our reward, we update the distribution using the updating procedure we defined earlier using the Gaussian process in the direction of the considered hyperparameters. If X_{s+1} yields a positive expected improvement, we calculate the reward value $R(X_{s+1})$ after 50 training iterations and add this to our set of hyperparameter values. We follow this process for N_{HP} times when the expected improvement is larger than zero, which we set equal to 50 to allow for sufficient exploration. Together, the algorithm is defined as:

Algorithm 2 Bayesian hyperparameter optimisation process

Sample s sets of hyperparameters $X_1, \ldots, X_s \in \mathbf{X}$
$N_{HP} = 50$
while $N_{HP} > 0$ **do**
    Consider $X_{s+z} \in \mathbf{X}$ new set of hyperparameters
    if $Z_{s+z}(\Phi(Z_{s+z}) + \phi(Z_{s+z})) \geq 0$ **then**
        Update distribution: $f_{s+z}(X_{s+z}) \sim N(\mu_{s+z}(X_{s+z}), \sigma^2_{s+z}(X_{s+z}))$
        $N_{HP} = N_{HP}$ - 1
    else
        Do not update distribution: $f_{s+z}(X_{s+z}) = f_{s+z-1}(X_{s+z})$

Using this procedure, we can optimise our hyperparameters without using the inefficient grid search method, where each combination of hyperparameters would need to be considered. Also, compared to random search, we can evaluate more sets of hyperparameter values without evaluating all of them, but rather only considering the sets of hyperparameter values that have an expected improvement that is larger than zero.

6.3 Hyperparameters

6.3.1 M_C model without EIIE

Table 7: Hyperparameters derived using Bayesian hyperparameter procedure for the M_C model using the proposed neural network architecture, where 5% of extreme downside returns are left out to test the hedging procedure. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0025	34	0.15	0.24	5	54	5
1, 0, 0	0.0088	64	0.70	0.43	5	47	5
0, 1, 0	0.0022	30	0.45	0.034	9	50	5
0, 0, 1	0.0025	36	0.95	0.11	6	52	3
0.5, 0.5, 0	0.0064	35	0.63	0.29	10	55	7
0.5, 0, 0.5	0.0041	69	0.088	0.24	12	52	7
0, 0.5, 0.5	0.0039	64	0.78	0.49	17	48	5
0.7, 0.2, 0.1	0.0026	34	0.60	0.0027	19	45	3
0.2, 0.7, 0.1	0.0081	34	0.67	0.019	12	51	7
0.45, 0.45, 0.1	0.010	65	0.65	0.38	7	49	3

Table 8: Hyperparameters derived using Bayesian hyperparameter procedure for the M_C model using the proposed neural network architecture. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0010	64	0.62	0.48	17	54	5
1, 0, 0	0.0042	30	0.70	0.29	17	52	7
0, 1, 0	0.00092	65	0.23	0.024	7	50	3
0, 0, 1	0.00075	32	0.86	0.0011	7	54	5
0.5, 0.5, 0	0.0093	30	0.89	0.43	6	55	5
0.5, 0, 0.5	0.0037	29	0.80	0.22	18	52	7
0, 0.5, 0.5	0.0055	56	0.75	0.14	16	55	5
0.7, 0.2, 0.1	0.0018	63	0.78	0.24	8	50	3
0.2, 0.7, 0.1	0.0090	44	0.40	0.20	20	51	5
0.45, 0.45, 0.1	0.0080	72	0.47	0.40	7	51	5

6.3.2 M_{S1} model without EIIE

Table 9: Hyperparameters derived using Bayesian hyperparameter procedure for the M_{S1} model using the proposed neural network architecture, where 5% of extreme downside returns are left out to test the hedging procedure. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0095	29	0.56	0.48	8	49	3
1, 0, 0	0.0094	71	0.65	0.39	7	51	5
0, 1, 0	0.0034	37	0.38	0.15	8	45	5
0, 0, 1	0.0035	43	0.72	0.24	8	45	5
0.5, 0.5, 0	0.0074	45	0.45	0.41	8	46	5
0.5, 0, 0.5	0.0053	56	0.14	0.34	9	48	7
0, 0.5, 0.5	0.00053	55	0.55	0.26	14	49	7
0.7, 0.2, 0.1	0.0041	42	0.44	0.05	15	51	3
0.2, 0.7, 0.1	0.0064	44	0.72	0.073	14	48	7
0.45, 0.45, 0.1	0.0076	52	0.44	0.52	10	53	3

Table 10: Hyperparameters derived using Bayesian hyperparameter procedure for the M_{S1} model using the proposed neural network architecture. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.00063	32	0.55	0.32	13	46	5
1, 0, 0	0.00053	47	0.56	0.33	15	55	7
0, 1, 0	0.00076	49	0.32	0.094	9	53	5
0, 0, 1	0.0014	44	0.28	0.064	9	47	5
0.5, 0.5, 0	0.0021	46	0.48	0.66	7	47	5
0.5, 0, 0.5	0.0011	35	0.65	0.092	14	47	7
0, 0.5, 0.5	0.00041	42	0.39	0.073	17	53	5
0.7, 0.2, 0.1	0.0034	54	0.81	0.32	11	48	3
0.2, 0.7, 0.1	0.0055	39	0.56	0.33	14	47	3
0.45, 0.45, 0.1	0.0033	66	0.69	0.27	9	46	5

6.3.3 M_C model with EIIE

Table 11: Hyperparameters derived using Bayesian hyperparameter procedure for the M_C model using the EIIE neural network architecture, where 5% of extreme downside returns are left out to test the hedging procedure. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0076	40	0.40	0.41	10	49	5
1, 0, 0	0.0051	62	0.86	0.21	9	47	5
0, 1, 0	0.0059	48	0.28	0.35	6	50	5
0, 0, 1	0.0094	67	0.59	0.46	7	49	5
0.5, 0.5, 0	0.0022	72	0.39	0.13	3	46	5
0.5, 0, 0.5	0.0089	54	0.17	0.07	8	51	5
0, 0.5, 0.5	0.0087	48	0.78	0.38	6	49	5
0.7, 0.2, 0.1	0.0091	69	0.45	0.50	6	47	3
0.2, 0.7, 0.1	0.0068	47	0.78	0.22	5	54	5
0.45, 0.45, 0.1	0.0092	44	0.63	0.049	8	52	5

Table 12: Hyperparameters derived using Bayesian hyperparameter procedure for the M_C model using the EIIE neural network architecture. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0072	47	0.13	0.44	4	51	5
1, 0, 0	0.0092	57	0.31	0.075	7	46	5
0, 1, 0	0.0084	69	0.82	0.33	5	46	3
0, 0, 1	0.0058	47	0.90	0.16	8	46	3
0.5, 0.5, 0	0.00060	35	0.16	0.030	4	50	3
0.5, 0, 0.5	0.0096	42	0.94	0.19	5	46	5
0, 0.5, 0.5	0.0061	57	0.143	0.0091	9	53	3
0.7, 0.2, 0.1	0.0086	66	0.96	0.44	7	46	5
0.2, 0.7, 0.1	0.0059	26	0.37	0.35	10	48	3
0.45, 0.45, 0.1	0.0099	42	0.94	0.37	7	46	5

6.3.4 M_{S1} model with EIIE

Table 13: Hyperparameters derived using Bayesian hyperparameter procedure for the M_{S1} model using the EIIE neural network architecture, where 5% of extreme downside returns are left out to test the hedging procedure. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0019	39	0.88	0.0086	7	52	5
1, 0, 0	0.0080	51	0.059	0.25	9	52	3
0, 1, 0	0.0082	27	0.27	0.14	8	46	3
0, 0, 1	0.0097	36	0.46	0.070	8	51	3
0.5, 0.5, 0	0.00034	56	0.62	0.38	7	48	3
0.5, 0, 0.5	0.0090	67	0.52	0.14	7	47	5
0, 0.5, 0.5	0.0092	51	0.70	0.24	4	55	5
0.7, 0.2, 0.1	0.0089	61	0.24	0.27	8	51	5
0.2, 0.7, 0.1	0.00019	53	0.98	0.19	7	49	5
0.45, 0.45, 0.1	0.0088	70	0.14	0.45	9	45	5

Table 14: Hyperparameters derived using Bayesian hyperparameter procedure for the first separate model using the EIIE neural network architecture. Hist. input stands for historical input of the model (N^H)

(w_R, w_V, w_E)	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
1/3, 1/3, 1/3	0.0028	39	0.80	0.45	6	47	5
1, 0, 0	0.0090	53	0.35	0.24	3	46	5
0, 1, 0	0.0092	56	0.43	0.18	6	51	3
0, 0, 1	0.0075	37	0.93	0.42	8	55	5
0.5, 0.5, 0	0.0037	59	0.17	0.098	9	46	5
0.5, 0, 0.5	0.0094	45	0.87	0.41	6	48	3
0, 0.5, 0.5	0.0072	57	0.93	0.032	3	47	3
0.7, 0.2, 0.1	0.0029	26	0.32	0.24	9	49	3
0.2, 0.7, 0.1	0.0090	51	0.098	0.49	7	47	5
0.45, 0.45, 0.1	0.0038	49	0.95	0.43	7	47	5

6.3.5 M_{S2} volatility models

Table 15: Hyperparameters derived using Bayesian hyperparameter procedure for the M_{S2} model without EIIE, focusing on volatility. Hist. input stands for historical input of the model

Hedge/ no Hedge	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
Hedge	0.0082	46	0.024	0.45	12	10	5
No hedge	0.0039	55	0.41	0.17	5	6	5

Table 16: Hyperparameters derived using Bayesian hyperparameter procedure for the M_{S2} model using EIIE, focusing on volatility. Hist. input stands for historical input of the model

Hedge/ no Hedge	Learning rate	Batch size	Explore/exploit	Dropout	Nodes/layer	Hist. input	Kernel
Hedge	0.00034	59	0.64	0.14	6	9	5
No hedge	0.0087	71	0.60	0.29	9	5	3