Ca' Foscari
University
of Venice

Master's Degree
in Economics, Finance and Sustainability

Final Thesis

Can Reinforcement Learning Agents Generate Reliable Market Views?

Some evidence from dynamic portfolio management through a RL-BL hybrid approach.

Supervisors
Ch. Prof. Marco Corazza
Ch. Prof. Stefano Federico Tonellato

Graduand
Edoardo Andretta
Matriculation Number 888603

Academic Year
2025 / 2026

Acknowledgments

This work would not have been possible without the guidance and supervision of Prof. Marco Corazza and Prof. Stefano F. Tonellato. Prof. Corazza contributed massively to the Reinforcement Learning part of this thesis, while the statistical testing and analysis parts were conducted under the supervision of Prof. Tonellato. Unfortunately, Ca'Foscari does not allow for two first supervisors. Since both Stefano's and Marco's contributions held equal importance for this study, the role of first supervisor was assigned to Prof. Corazza following the toss of a fair coin[1].

I wish to thank the IT department at Ca'Foscari for allowing—and swiftly set up—the use of the computational cluster to train our agents, simulate their runs, and conduct the statistical tests used in this study.

This thesis allowed me to narrow my personal research interests down to the most modern aspects of computational and quantitative finance. This study represents for me the first stone of my research, which I will carry on as a PhD student in Finance at the David Eccles School of Business of the University of Utah. To this extent I would like to sincerely thank my referees, Prof. Silvia Faggian, which also supervised my activity as academic tutor of Mathematics, and Prof. Andrea Teglio, which supervised my Bachelor's thesis. With respect to this instance, I wish to thank once again Prof. Tonellato, both for being my third referee and for supervising my activity as academic tutor of Statistical Models and Methods for Finance.

Outside thesis specific acknowledgments, I wish to thank first and foremost Charis, who oftentimes spent time listening to the developments of this study[2], for her patience, support, and for always being next to me whenever I needed it. A special thank goes also to all the members of my family, to my parents for having always supported my choices and my growth, and to my brothers for having always been amazing friends to me.

My friends also deserve to be mentioned in this section. I wish to thank all of my closest friends for the laughs, the good moments, the company, and the time spent together in the past years, perfectly knowing how difficult it will be to leave them for the new chapter of my life awaiting me in Salt Lake City.

Lastly, a special thank goes to all the faculty members at Ca'Foscari which I've met during my undergraduate and graduate years spent there, and which I have not expressly mentioned yet. My goal in life is to pursue a successful academic career, and the faculty at Ca'Foscari serves as a powerful inspiration to me.

[1] It shall be reported that, after being informed of the outcome, Prof. Tonellato commented: "Still better than losing at penalties against Bosnia".

[2] Despite not being particularly engaged with the topic, I must say.

Disclosure of use of Artificial Intelligence Tools

As required by Ca'Foscari guidelines for degree theses, Section 1.3, I, Edoardo Andretta, disclose of having used generative AI tools—Google Gemini—to revise and refine text in order to improve clarity and coherence, and to debug and optimize the code I wrote while conducting the experiment. The generated text has been verified manually for accuracy, quality, and reliability.

Abstract

We assess the ability of Reinforcement Learning (RL) agents to form reliable, consistent market views by constructing four families of agents tasked with generating views to dynamically manage a diversified equity portfolio via the Black-Litterman (BL) model. Using narrow and wide datasets, we explore the impact of information density on View Generation (VG). We evaluate how Artificial Neural Network (ANN) architecture and temporal awareness (look-back windows) affect VG, and compare performance across on-policy (PPO) and off-policy (SAC) algorithms.

We find that RL agents are able to conduct VG reliably and consistently provided they possess an information-dense observation space, an optimal ANN architecture, and sufficient temporal awareness. Both algorithm classes train functional policies provided adequate reward functions are used in each case; specifically, on-policy PPO requires a highly specific reward function, called IRR, whereas off-policy SAC performs best with a generic one. We conclude that RL agents are able to dynamically manage portfolios through the BL model which are able to outperform benchmarks reliably. However, they seem to suffer from shifts in market regime as posited by the Adaptive Market Hypothesis (AMH). Furthermore, we establish that our framework is valid to conduct empirical tests of both the AMH and the Efficient Market Hypothesis (EMH).

Contents

Chapter 1

Introduction

Harry Markowitz gave birth to Modern Portfolio Theory with the paper *Portfolio Selection*, published in, arguably, the most important journal in financial economics—The Journal of Finance—in March 1952. By proposing a simple portfolio selection model, relying on variance minimization, Markowitz started a secular revolution within the field of finance. What was a field originally dominated by rules of thumb, biases, and heuristics, transitioned to a mathematically rigorous and academically noteworthy field from March 1952 onward.

As of today, more than seventy-four years have passed since the foundational study published by Markowitz, and finance still is one of the fastest evolving disciplines under the economics umbrella. Especially within the last ten years, finance has witnessed the rise of applied artificial intelligence techniques, which seek to offer a model-free alternative solution to problems of portfolio allocation, asset pricing, time series forecasting, and many more.

The benefit of using model-free methods, such as the ones coming from artificial intelligence, and mainly from its machine learning sub-field, is evident: the number of assumptions needed for the solution tools to work drops significantly. For example, using a function approximator, such as an Artificial Neural Network, to solve portfolio optimization problems only assumes that a measurable *optimization function* exists to be learned. This approach effectively bypasses traditional, rigid assumptions like investor rationality, relying instead on empirical data to output optimal solutions.

As a consequence of this emerging academic interest in blending advancements in computer science with finance, the financial industry is also rapidly implementing AI tools and techniques in various fields, from asset management and derivatives pricing, to quantitative and algorithmic trading.

In this study, we supply evidence in favor of this novel academic tendency of leaving financial models for data-driven approaches to finance. Specifically, we aim at assessing the ability of Reinforcement Learning agents to reliably form views about equity returns by sensing the state of the market with multiple look-back windows. We then measure the portfolio management performance of these agents by feeding their generated views into the Black-Litterman model to manage dynamic portfolios.

1.1 Goal of the Study

As previously mentioned, the prime goal of our study is to assess the viability of Reinforcement Learning (RL) agents to be used as views generating computational entities, their views being used to dynamically manage a Black-Litterman optimized diversified equity portfolio.

The Black-Litterman (BL) dynamic allocation problem serves as a rigorous testing ground to evaluate our core objective, that is, rephrased, to understand whether or not machines, in this case RL agents, can form reliable expectation about the future state of the market—hence views on asset returns—by only being fed data. Essentially, we are leaving the deeply-explored path of model-based financial forecasting in order to assess whether a model-free, computational, and data-driven alternative is feasible and applicable.

Throughout this thesis, we aim at studying how the type of data, architecture, reward function, and training algorithm influence the ability of RL agents to generate views reliably and consistently. Then, we aim at assessing whether view-forming RL agents are able to outperform buy-and-hold benchmarks by dynamically managing a diversified portfolio of stocks. This objective aligns closely with contemporary trends in computational finance that seek to develop autonomous, adaptive tools for asset management.

Conducting our study will also provide us with the opportunity of observing the manifestations—and the limitations too—of two foundational hypotheses of finance: the Efficient Market Hypothesis, proposed in Fama (1970), and the Adaptive Market Hypothesis, proposed in Lo (2004).

Data-driven computational approaches to finance, like our instance of views formation through RL agents, unavoidably have to deal with the information density of the observation space, the so-called *signal*. In every dataset the amount of signal competes with the remaining *noise*, which does not carry any information. The weak form of the EMH posits that past returns contain little signal. By comparing the performance of agents trained, and acting, on datasets with different dimensionalities—including a configuration containing only past returns—we will observe a manifestation of the weak form of market efficiency.

The Adaptive Market Hypothesis instead posits that heuristics, which contribute to view formation in our study, are based on the current market regime, and that shifts in it can hamper old heuristics' functioning. By tracking the performance of our agents' portfolios throughout distinct market regimes we aim to observe and evaluate the empirical validity of Lo's conjecture.

The goals of our study extend beyond simply drawing conclusions on the viability of automated portfolio management. We aim at assessing whether a new paradigm of finance is needed by studying how machines interact with the market, and the systemic consequence the evidence drawn in our experimental framework will have on finance as a discipline.

A thorough description of our five main research goals, alongside the methodology applied, can be retrieved in Section 3.1.

1.2 Literature Review

Our study fits within the recent research trend in finance aimed at automating financial decision making, according to the classification provided by Aritonang et al. (2024).

Furthermore, we are strongly motivated by recent advancements in Artificial Intelligence Pricing Theory postulated in Didisheim et al. (2024), which proposes that returns are driven by a large number of factors, which can be handled only through Machine Learning techniques.

Aritonang et al. (2024) notice that the interest of the financial academic community for the use of Machine Learning (ML) in asset valuation and portfolio management has started around 2015, with the number of publications on the topic increasing massively over the last eleven years. While the first two years were concerned with building the foundations for modern computational finance, it was from 2017 onward that the potential of ML applied to portfolio optimization models was acknowledged, with Ha et al. (2017) proposing a portfolio optimization model which minimizes a soft margin-based generalization bound.

Further developments to the application of ML methods to portfolio selection problems came alongside the application of deep learning and big data analytics in 2019 and 2020. In those years, the research trend involving RL methods in portfolio optimization began.

Wang et al. (2025) presents a survey of notable instances of the application of RL methods to, among others, portfolio selection problems. Wang et al. (2019) proposed one of the first interpretable investment strategy using DRL models. In their work, a LSTM network is trained through a Policy Gradient (PG) algorithm to output asset weights based on trading and fundamental features of the considered stocks.

The use of PG algorithms to train various kinds of ANNs in choosing asset weights has been common practice within the first phase of portfolio selection oriented (PSO) RL research. Other notable examples can be found in Wang et al. (2021), where macro market conditions are embedded in a DRL portfolio selection model, and in Xu et al. (2021), where a Relation-aware Transformer (RAT) is trained through PG to manage cryptocurrency portfolios.

Concurrently to the aforementioned use policy-based RL methods, the early years of PS oriented RL research brought development also on the use of value-based RL methods, mainly employed through Deep Q-Network (DQN) algorithms used to train Convoluted Neural Networks (CNNs) and Multi-Layer Perceptrons (MLPs). Notable examples of this specific research trends can be found in Gao et al. (2020), in which a CNN is trained through DQN to output asset weights, in Gao et al. (2021), which improves on the precedent paper by introducing agents capable of handling transaction costs, and in Lee et al. (2020), in which Multi-Agent reinforcement learning-based Portfolio management System (MAPS) are proposed, where each agent is composed of a MLP network trained through DQN.

The application of PPO—the on-policy algorithm we will use throughout our study—to PSO RL research is more recent. While Yang et al. (2020) could be considered as an early proponent of the use of PPO in portfolio optimization, PPO's widespread adoption by the academic community began around 2023. Examples of applications of PPO to portfolio optimization can be found in Kumlungmak and Vateekul (2023), where PPO is used to train RL agents in manage cryptocurrency portfolios, and in Sun et al. (2024a), where PPO is used to train a GraphSAGE feature extractor in generating asset weights.

SAC—the off-policy algorithm we will use throughout our study—has been only recently applied to portfolio optimization. One of the first studies involving SAC in portfolio optimization was Aboussalah et al. (2022), where SAC was initially considered unsuitable for portfolio management applications, due to supposed difficulties in converging. However, successive research, such as the study from Choi and Kim (2024) revisited the

concept of using SAC in PS, this time in conjunction with other models, finding it able to converge properly, providing satisfactory results.

So far we presented the literature behind the plain application of RL to portfolio selection, which involved autonomous agents generating asset weights directly. However, this study takes the concept one step further. Our agents are not simply tasked to observe the state of the market and directly output weights. Instead, they are tasked to output absolute views about future assets' performances—essentially they are trying to predict future returns—and their associated confidence. Those measures are then used as input for the BL model, which then augments a market cap weighted prior portfolio to generate asset weights.

Originally proposed in Black and Litterman (1990), and later improved in Black and Litterman (1992), the Black-Litterman model was the first of its kind to incorporate investor's views into portfolio optimization, by using them to suitably modify the Bayesian prior portfolio. Specifically, the mathematical interpretation of the BL model we use in our thesis is the one from He and Litterman (2002). Furthermore, in our study we will follow the view incorporation procedure set out in Idzorek (2007).

The idea of using ML methods to generate views for the Black-Litterman model is not revolutionary by itself. Starting from 2021 onward, different research groups have published about the view generating abilities of ML models. Kolm et al. (2021) discusses the Bayesian interpretation of the BL model and the potential of using machine learning methods to generate objective investment views. In Min et al. (2021) Random Forests were used to generate investor's opinions which proved able to significantly improve returns. In later papers more complex ML techniques are used to generate views. In Punyaleadtip et al. (2024), recurrent neural networks are used to generate views, while in Zhu and Yen (2024) Generative Adversarial Networks (GANs) are used to the same extent.

Aside from the specific ML techniques used to generate views, the kind of data used to do so has evolved too. From simple datasets containing only technical indicators used in early research, to datasets including also macroeconomic indicators—such as the one we will use in our study—and to more exotic data sources. Barua and Sharma (2023), for example, includes also fear and greed indicators, alongside the use of the XGBoost ensemble learning algorithm, to generate BL views. Frontier papers, such as Ko and Lee (2025), go one step further by augmenting the BL model itself to seamlessly integrate ML generated views.

The use of specifically reinforcement learning agents to generate BL views is more niche. An example of this instance can be found in Sun et al. (2024b), where a DRL agent was trained using various algorithms, including PPO and SAC, to apply the BL model in its entirety—so the agents were not tasked only with generating views. In Mikriukov et al. (2025) a DRL agent defined through a CNN was trained using TD3 in generating views, obtaining highly satisfactory results. Another instance of the application of RL to portfolio optimization through the BL model can be found in Vasilevich (2025), where a dataset containing multiple technical indicators is fed to the view-generating agents.

To our understanding, as of today no published study focused solely on the view generating ability of RL agents, investigating to which extent it is influenced by the dimensionality of the datasets used. Furthermore, it seems that no direct comparison between on-policy and off-policy RL methods applied to BL portfolio selection has been made so far. Our study is novel with respect of the two aforementioned characteristics. We aim at, first, investigating to which extent the view generating ability of RL agents

is dependent on the data they are fed and, secondly, whether on-policy methods offer an edge over off-policy algorithms, or vice versa. Furthermore, our study is innovative with respect to a third, collateral, implication: by analyzing the performance of agents operating on different dimensionality datasets, we aim at observing manifestations of the Efficient Markets Hypothesis, as postulated in Fama (1970), and of the Adaptive Market Hypothesis, presented in Lo (2004). To our knowledge, no study so far has used RL agents to find empirical evidence of the two aforementioned conjectures.

1.3 Navigating the Study

This study is, unavoidably, dense. We divided our thesis in three main chapters. Chapter 2 is dedicated to presenting thoroughly the theoretical background needed to fully grasp the content of our study. Chapter 3 presents the experimental framework we employ to achieve the goals we set out in Section 1.1. Finally, in Chapter 4 we present thoroughly the results we obtained through our experiment. In addition to the three main chapters, the reader might need to refer to Appendices A to D for commented tables of results, plots, detailed tables containing the hyperparameters used for training our RL agents, and exhaustive lists of acronyms and symbols.

With respect to Chapter 2, it is divided in four main sections. Section 2.1 presents the Black-Litterman model for asset allocation we will use in our experiment. Section 2.2 presents the concept of Artificial Neural Networks (ANNs) as they will be the function approximators of choice for our RL agents. Section 2.3 is focused on providing the reader with the basis of Reinforcement Learning (RL). Finally, Section 2.4 formalizes the two algorithms which will define our agents: PPO and SAC.

Chapter 3 is organized in five main sections. In Section 3.1 we present a brief overview of the five main experimental goals of our study, alongside providing with some useful terminology. In Section 3.2 we present in detail each one of the five experimental goals laid out in the preceding section, providing the reader with the rationale being them and with the methodology we will use to achieve them in our experiment. Section 3.3 provides a detailed description of the two datasets we will use as observation spaces for our agents: the *narrow* dataset, which contains only past stock prices per each stock, and the *wide* dataset, which instead contains multiple technical, financial, momentum, and macroeconomic factors. Furthermore, we also explain how the dataset will be divided between training-validation of our agents and their successive backtesting windows. In Section 3.4 we will introduce formally the main characters in our study: the RL agents. Specifically, we will discuss the differences between the two classes of agents, *narrow* and *wide*, and their network architectures. Finally, in Section 3.5 we introduce the two reward functions which will be used to reward our agents and provide the reader with some technical notes on the implementation of our experiment for reproducibility.

Chapter 4 is where we tie all our knots. The conclusive chapter is organized in four main sections. In Section 4.1 we discuss thoroughly agents' training, by first providing an overview of our computational setup, to then present the process of hyperparameter optimization. We conclude the section with some insights gained from agents' validation logs. Section 4.2 is perhaps the most important of our study. In it, we first describe our general backtesting setup, to then move on presenting deterministic backtesting, initially from a theoretical standpoint, just to then discuss our agents' performances. In Subsection 4.2.1, we show that our *wide* agents can manage portfolio able to outper-

form benchmarks reliably across multiple backtesting windows. Once we have discussed deterministic runs, we will move to stochastic simulations. In Subsection 4.2.2 we first present the concept of stochastic simulations, the two tests we will use to assess FSD and stochastic greatness of agents' returns over a random baseline—the Kolmogorov-Smirnov (KS) and the Mann-Whitney U test respectively—to then discuss the results obtained. Once again, we find that certain wide agents' returns often test positive for stochastic greatness, occasionally validated by attaining also FSD over the random baseline. In Section 4.3 we evaluate our five research goals set out in Section 3.2 in light of the evidence collected through our backtests. Finally, in Section 4.4 we make some final remarks on the impact of our study, address future research, and lay out five hypotheses empirically supported by our results which we believe might be worth exploring in future research. Furthermore, we evaluate if we are in front of a paradigm shift in finance, as mentioned in Section 1.1, answering in the affirmative.

Chapter 2

Theoretical Foundations

Our study is theoretically dense. Recall that the main objective of our study is to assess the ability of Reinforcement Learning trained agents to manage dynamic equity portfolios by means of an underlying asset allocation model, alongside considering the impact this could have on the theoretical foundations upon which modern finance rests—namely, market efficiency and theoretical asset pricing.

There are four theoretical pillars needed by the reader to fully grasp the content of this thesis. The first pillar is the portfolio optimization model our agents will use to manage their portfolios, the Black-Litterman model. The second pillar is represented by function approximators, which constitute the *digital brain* of our agents, that in this study take the form of Multilayer Perceptrons. The third pillar is the machine learning technique that will train our agents on how to act: Reinforcement Learning. The fourth, and final, pillar is constituted by the specific algorithms we will use to apply Reinforcement Learning: PPO and SAC.

The goal of this chapter is to provide the reader with an in-depth theoretical presentation of the aforementioned pillars, which will be needed to fully grasp not only the experimental design we propose in Chapter 3, but also the conclusions we draw in Chapter 4.

Section 2.1 provides a deep presentation of the Black-Litterman model for asset allocation, specifically focusing on the contemporary mathematical interpretation proposed in He and Litterman (2002). Section 2.2 aims at presenting our second pillar: function approximators. In particular, the reader will be presented with the evolution of the first artificial neuron to the neural network architecture we will use throughout our study, the Multilayer Perceptron, alongside brief presentations of other ANNs architectures of modern financial relevance. In Section 2.3 we present thoroughly Reinforcement Learning, from its definition and elements to the main class of RL algorithms. Finally, in Section 2.4 we present in depth PPO and SAC.

2.1 The Black-Litterman Model

Developed in the early 1990s by Fischer Black, also known for his work on the Black-Scholes-Merton model, and Robert Litterman, the Black-Litterman (hereinafter referred as BL) asset allocation model uses the Bayesian approach to infer the assets' expected returns. Under this approach, the expected returns are non-observable random variables themselves. One can only infer their probability distribution starting with a prior belief, in the BL model case represented by the CAPM equilibrium distribution, alongside additional information, the investor's views, used to infer the posterior distribution. The purpose of this section is to briefly present the intuitions behind the BL model for asset allocation from He and Litterman (2002), which follows. For a thorough and detailed discussion of the model, the reader shall refer to Black and Litterman (1990), and to Black and Litterman (1992).

Consider a market in which N assets exist, which may include equities, bonds, currencies, and other assets. Now assume that the simple returns of these assets have a normal distribution with μ being the expected return and Σ the covariance matrix. That is,

$$r \sim N(\mu, \Sigma) \tag{2.1}$$

where r is the vector of asset returns [1]. In equilibrium, all investors hold the market portfolio w_{eq}. The equilibrium risk premiums, Π, are such that if all investors hold the same view, the demand for these assets exactly equals the outstanding supply. Assuming the average risk tolerance of the world is represented by the risk parameter δ, the equilibrium risk premiums are given by

$$\Pi = \delta \Sigma w_{eq} \tag{2.2}$$

The Bayesian prior is that the expected returns are centered at the equilibrium values, that is they are normally distributed with the mean of Π,

$$\mu = \Pi + \epsilon^{(e)} \tag{2.3}$$

where $\epsilon^{(e)}$ is a normally distributed random vector with zero mean and covariance matrix $\tau\Sigma$, where τ is a scalar indicating the uncertainty of the CAPM prior [2].

In addition to the CAPM prior, the investor also has a number of views on the market returns. Each view is expressed as a statement that the expected return of a portfolio p has a normal distribution with mean equal to q and a standard deviation given by ω[3]. Let K be the total number of the views, P be a $K \times N$ matrix whose rows are the portfolios weights and Q be a K-vector of the expected returns on these portfolios. That is,

$$P' = \begin{pmatrix} p_1 & p_2 & \cdots & p_K \end{pmatrix} \tag{2.4}$$

$$Q' = \begin{pmatrix} q_1 & q_2 & \cdots & q_K \end{pmatrix} \tag{2.5}$$

Then the investor's views can be expressed as

$$P\mu = Q + \epsilon^{(v)} \tag{2.6}$$

[1] Note that this is a very strong, but needed, assumption.

[2] Note that there is no single correct way to compute τ. Black and Litterman originally suggested setting τ to a small value. Other practitioners prefer to set $\tau = 1/T$, where T is the sample size used to compute the covariance matrix in a practical setting, where we must work with empirical data.

[3] In essence, each view is a statistical model for the observable returns.

where $\epsilon^{(v)}$ is an unobservable normally distributed random vector with zero mean and a diagonal covariance matrix Ω. Is is further assumed that $\epsilon^{(e)}$ and $\epsilon^{(v)}$ are independent

$$\begin{pmatrix} \epsilon^{(e)} \\ \epsilon^{(v)} \end{pmatrix} \sim N\left(0, \begin{bmatrix} \tau\Sigma & 0 \\ 0 & \Omega \end{bmatrix}\right) \tag{2.7}$$

These views are then combined with the CAPM prior in the Bayesian framework. The result is that the expected returns are distributed as $N(\bar{\mu}, \bar{M}^{-1})$, where the mean $\bar{\mu}$ is given by

$$\bar{\mu} = [(\tau\Sigma)^{-1} + P'\Omega^{-1}P]^{-1}[(\tau\Sigma)^{-1}\Pi + P'\Omega^{-1}Q] \tag{2.8}$$

and the covariance matrix $\bar{M}^{-1}$ is given by

$$\bar{M}^{-1} = [(\tau\Sigma)^{-1} + P'\Omega^{-1}P]^{-1} \tag{2.9}$$

It is important to notice that in the BL approach a view that, for example, a stock A will outperform a stock B is expressed as an expectation of a positive return on a portfolio consisting of a long position in the stock A and a short position in the stock B. This view is then translated through Equation 2.8, which appropriately takes volatilities and correlations into account, into adjustments to expected returns in all of the markets. These adjustments are those needed to suggest the optimizer that the best opportunity is a simple over-weighting of stock A, financed by under-weighting stock B. To show this, let us consider the case of an unconstrained investor with representative risk aversion parameter δ.

Because the expected returns are random variables themselves in the BL model, the distribution of returns is no longer simply $N(\bar{\mu}, \Sigma)$. According to Equations 2.1, 2.8, and 2.9, the distribution for the returns is

$$r \sim N(\bar{\mu}, \bar{\Sigma}) \tag{2.10}$$

where $\bar{\Sigma} = \Sigma + \bar{M}^{-1}$. Given the mean and the covariance matrix, the optimal portfolio can be constructed using the standard mean-variance (MV) optimization method (Markowitz (1952)). For an investor with risk aversion parameter δ, the maximization problem can be written as

$$\max w'\bar{\mu} - \frac{\delta}{2} w'\bar{\Sigma}w \tag{2.11}$$

The first order condition yield that

$$\bar{\mu} = \delta\bar{\Sigma}w^* \tag{2.12}$$

or equivalently,

$$w^* = \frac{1}{\delta}\bar{\Sigma}^{-1}\bar{\mu} \tag{2.13}$$

where w^* is the vector of optimal portfolio weights. Substituting equation (2.8), w^* can be written as

$$w^* = \frac{1}{\delta}\bar{\Sigma}^{-1}\bar{M}^{-1}[(\tau\Sigma)^{-1}\Pi + P'\Omega^{-1}Q] \tag{2.14}$$

Note that

$$\bar{\Sigma}^{-1} = (\Sigma + \bar{M}^{-1})^{-1} = \bar{M} - \bar{M}(\bar{M} + \Sigma^{-1})^{-1}\bar{M} \tag{2.15}$$

hence the term $\bar{\Sigma}^{-1}\bar{M}^{-1}$ can be simplified as

$$\bar{\Sigma}^{-1}\bar{M}^{-1} = \frac{\tau}{1+\tau}\left(I - P'A^{-1}P\frac{\Sigma}{1+\tau}\right) \tag{2.16}$$

where the matrix A is given by $A = \Omega/\tau + P\Sigma/(1+\tau)P'$. The optimal portfolio weights now can be written as

$$w^* = \frac{1}{1+\tau}(w_{eq} + P' \times \Lambda) \tag{2.17}$$

where w_{eq} is the market equilibrium portfolio from equation (2.2) and Λ is a vector defined as

$$\Lambda = \tau\Omega^{-1}Q/\delta - A^{-1}P\frac{\Sigma}{1+\tau}w_{eq} - A^{-1}P\frac{\Sigma}{1+\tau}P'\tau\Omega^{-1}Q/\delta \tag{2.18}$$

Interpretation of equation (2.17) is straightforward. Since each column of matrix P' is a view portfolio, then (2.17) shows that the investor's optimal portfolio w^* is the market equilibrium portfolio w_{eq} plus a weighted sum of the portfolios forming the views, then scaled by a factor $1/(1+\tau)$. The weight for each view portfolio is given by the corresponding element in vector Λ.

Interpretation of equation (2.18) is less straightforward, but still we can wrap our heads around some intuitions. The first term shows that the stronger the view is, the more weight it carries in the final portfolio. The second term shows that the weight of a view is penalized for the covariance between the view portfolio and the market equilibrium. Since the market equilibrium information is already present in the prior, the covariance of a view portfolio with the market equilibrium indicates that the view carries less new information and the penalty makes sense. Furthermore, the last term shows also that the weight is penalized for the covariance of a view portfolio with other view portfolios, so to avoid double counting identical views.

To the extent of our study, the Black-Litterman model presents a clear distinction between the role of the data and the role of an agent, which in our case will be a RL one. In practice, the data will be used to estimate the CAPM prior, while the agent will have to be trained to the extent of generating coherent views based on the state of the market at the current time step, which will be integrated by the BL model to infer the posterior distribution. Then, the optimal portfolio w^* can either be computed as in equation (2.11), in the classical Markowitz framework, or by making good use of equation (2.17).

2.2 From the Perceptron to ANNs

Machine Learning—hereinafter ML—is an extraordinarily vast field. There are four main families of ML: supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning (RL). To achieve our research objectives we will focus solely on the latter.

As we will discuss in Section 2.3, RL often involves taking actions following a policy on an extremely large—and often infinite—state space. In many RL frameworks, the agent follows a policy by estimating the utility of its environment through learned state-value or action-value functions. However, since the aforementioned functions are depending on potentially infinite state spaces, or infinitely many state-action pairs, it is often unfeasible to learn the exact function in finite time. Therefore, mathematical tools called *function approximators* are needed to approximate the chosen value function or, in other words, to achieve *generalization*.

This section is dedicated to the class of function approximators we will use to train the agents object of our study: Artificial Neural Networks—hereinafter ANNs. Specifically, in Section 2.2.1 we will cover the history of ANNs by presenting the Perceptron. In Section 2.2.2 we present the specific type of ANN we will use to train our agents: the Multi Layer Perceptron (MLP). Finally, Section 2.2.3 provides a brief discussion of contemporary developments and alternative ANN architectures.

2.2.1 The Perceptron

During the early 1940s research in neurology had shown that the human brain was an electrical network of neurons that fired in all-or-nothing pulses. Naturally, this discovery sparked great interest in formalizing the basic principles of biological neural network operation. To this extent, McCulloch and Pitts (1943) introduced the *perceptron*, a simple model of a biological neuron capable to conduct binary classification.

The perceptron works by considering a set of inputs $x_i \in \mathbb{R}$, with $i = 0, 1, ..., n$ and $x_0 = 1$ (bias), and a set of weights w_i, and using them to compute a weighted sum:

$$\sum_{i=0}^{n} w_i x_i = w_0 x_0 + \cdots + w_n x_n \tag{2.19}$$

Then, the weighted sum is transformed into an output through a transformation function $o(\cdot)$ of the form:

$$o(\cdot) = \begin{cases} +1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{if } \sum_{i=0}^{n} w_i x_i \leq 0 \end{cases} \tag{2.20}$$

As mentioned, the perceptron is a linear classifier able to specify one of the hyperplanes in the n-dimensional space of the instances. To be able to serve its function, the perceptron must be trained. Rosenblatt (1958) proposed the Delta rule for training a perceptron. The training algorithm proposed by Rosenblatt initializes randomly the vector of weights, then it enters three nested loops. Each training iteration is repeated for each weight, for each input-output training example, for K times. Any weight w_i is updated through the Delta rule:

$$w_i^{l+1} = w_i^l + \Delta w_i^l = w_i^l + \alpha(y_i - o_j^l)x_{i,j} \tag{2.21}$$

where $l = 1, ..., k$, α is the learning rate, and o_j^l is the j-th output computed by the perceptron at the l-th iteration.

According to the *perceptron convergence theorem*, the Delta rule terminates the updating of the weights w_i, with $i = 0, 1, ..., n$, after a finite number of iterations for any set of *linearly separable* instances. This implies that a perceptron can learn any classification in finite time, provided that the two classes are linearly separable.

As pointed out in Minsky and Papert (1969), requiring linear separability between the two classes makes the perceptron a weak learner—the classic example of this instance being the inability of the perceptron to learn the function XOR. Perceptron being a weak learner was a significant matter at that time. Thankfully, perceptron limitations can be overcome by employing a network of properly connected perceptrons, called a Multilayer Perceptron.

2.2.2 The Multilayer Perceptron

There are two ways to overcome perceptron's limitations, which are oftentimes employed simultaneously: employing different network topologies and using different transformation/squashing functions. As we will discuss in this section, the combination of an improved network topology and an improved squashing function makes the resulting Multilayer Perceptron—hereinafter MLP—a universal function approximator.

Table 2.1 contains a summary of the most common transformation/squashing functions used in MLPs. The one we will specifically focus on is the latter: the sigmoid squashing function.

Function	**Definition**	**Properties**
Step	$o(\cdot) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{if } a > 0 \end{cases}$	Neither continuous nor differentiable. Bounded, $o \in \{0, 1\}$.
Linear	$o(\cdot) = g \cdot a$	Continuous and differentiable in its domain. Unbounded, $o \in (-\infty, +\infty)$.
Stepwise Linear	$o(\cdot) = \begin{cases} 0 & \text{if } a \leq -s \\ g \cdot a & \text{if } -s < a \leq s \\ 1 & \text{if } a > s \end{cases}$	Continuous but not differentiable. Bounded, $o \in [0, 1]$.
Sigmoid	$o(\cdot) = \dfrac{1}{1 + \exp(-a)}$	Continuous and differentiable over its domain. Bounded, $o \in (0, 1)$.

Table 2.1: Different types of squashing functions. Consider $a = \sum_{i=0}^{n} w_i x_i$.

In general, an Artificial Neural Network—hereinafter ANN—is a net of artificial neurons. Topology is mainly concerned with the various ways in which neurons can be interconnected to form such a net. The specific ANN architecture we are interested in is called Multilayer Perceptron (MLP), an example of which is provided in Figure 2.1.

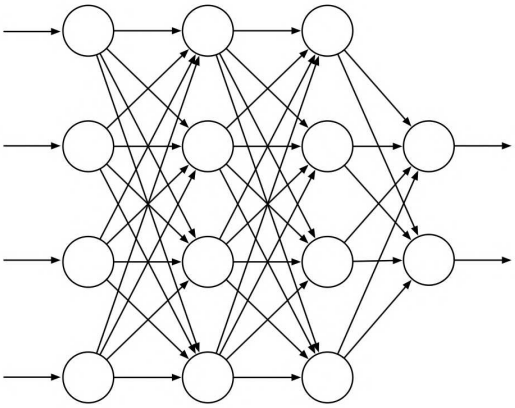


Figure 2.1: A generic MLP feedforward ANN with four input units, two output units and two hidden layers. Taken from Sutton et al. (1998).

The first layer characterizing a MLP feedforward ANN is the *input* layer. The input layer is constituted by the artificial neurons—oftentimes called also *nodes* or *units*—which receive the inputs from the external environment. The number of such neurons is $n + 1$, with n being the number of inputs plus one for the bias. Next, *hidden* layers can be found. Hidden layers are constituted by a given number $h_j \in \mathbb{N}^+ + 1$ of *thinking neurons*, with $j = 1, ..., H$, where H is the number of hidden layers. Both H and h_j are, generally, user specified. Finally, the *output* layer is constituted by the q artificial neurons responsible for releasing the output in the external environment. Those neurons are as many as the number of outputs.

Each layer in a MLP, apart the output layer, is fully connected with the next layer. The flow of information is forward, hence the name feedforward ANN. To each connection—oftentimes called also *arc* or *link*—is associated a real-valued weight. Each hidden neuron is usually characterized by a sigmoid squashing function, while each neuron in the output layer is characterized by a linear squashing function.

Each node in the MLP is a perceptron by itself. By Cybenko convergence theorem, posited in Cybenko (1989), MLP feedforward ANNs are universal function approximators. In Cybenko's own words:

> [The] networks with one internal layer and an arbitrary continuous sigmoidal function can approximate continuous function with arbitrary precision providing that no constraints are placed on the number of nodes or the size of the weights.

The same year, Hornik et al. (1989) extended this results to the broader class of measurable functions.

As for the perceptron, MLPs also need to be trained. In general, stochastic gradient methods are used to train MLPs. On of the first algorithms created specifically to train MLPs was the Error Backpropagation Algorithm (EBA), proposed by Rumelhart et al. (1985), which is based on the gradient descent method and set the stage for all future training algorithms for deep networks. We present a brief overview of the stochastic gradient descent method applied to RL in Section 2.3.9.

In brief, the EBA initializes random weights to then iteratively, for each weight for K

times, compute the error:

$$E = \frac{1}{2}\sum_{i=0}^{n}(y_i - o_i)^2 \tag{2.22}$$

To then solve the minimization problem:

$$\min_{w_1,\ldots,w_W} E \tag{2.23}$$

by updating each weight through the generalized Delta rule:

$$w_i^{l+1} = w_i^l + \Delta w_i^{l+1} = w_i^l - \alpha\frac{\partial E}{\partial w_i} \tag{2.24}$$

Sometimes, a momentum term is added to the generalized Delta rule to speed up the Error Backpropagation Algorithm. Doing so modifies Equation 2.24 to:

$$w_i^{l+1} = w_i^l - \alpha\frac{\partial E}{\partial w_i} + \beta w_i^l, \quad \beta \in [0, 1) \tag{2.25}$$

The backpropagation algorithm can produce good results for shallow networks having 1 or 2 hidden layers, but it may not work well for deeper ANNs. First, the large number of weights in a typical deep ANN makes it difficult to avoid the problem of *overfitting*, that is, the problem of failing to generalize correctly to cases on which the network has not been trained, which is a major issue considering that the main aim of any ANN is to generalize the patterns present in the data. Second, backpropagation does not work well for deep ANNs because the partial derivatives computed by its backward passes either decay rapidly toward the input side of the network, making learning by deep layers extremely slow, or the partial derivatives grow rapidly toward the input side of the network, making learning unstable.

We will not use the EBA to train our MLPs. Instead, we will rely on more modern training algorithms which are known to provide more stable and efficient learning.The training algorithms we will use throughout our study, PPO and SAC, are presented in Sections 2.4.1 and 2.4.2 respectively.

2.2.3 Other ANNs

Feedforward MLPs are not the only kind of artificial neural networks. Within the past eighty years since the first artificial neuron formalization, researchers and computer scientists have developed numerous different architectures of ANNs. The goal of this section is to present briefly some of the most common ANN topologies used within computational finance.

Recurrent Neural Network

Unlike MLPs, Recurrent Neural Network—hereinafter RNNs—are characterized by backward connections, where the output of a neuron at one time step is fed back as input to the network at the next time step. These backward connections enable RNNs to capture temporal dependencies and patterns within sequences. In finance this makes RNNs particularly good at time series forecasting and modeling volatility dynamics.

The concept of RNNs was first proposed in Rosenblatt (1960), and was improved in Rosenblatt et al. (1962). However, the concept was subsequently modified and improved thanks to the contributions of several authors, such as Nakano (1971) and Amari (1972).

While being impressive tools, traditional RNNs suffer from the vanishing gradient problem, which limits their ability of learning long-term dependencies. This issue was addressed by the development of LSTM, which is presented in the next paragraph.

Long Short-Term Memory

Long Short-Term Memory—hereinafter LSTM—was first proposed in Hochreiter and Schmidhuber (1997). LSTM is a type of RNN aimed at mitigating the vanishing gradient problem frequently encountered by traditional RNNs. It aims to provide a short-term memory for RNN that can last thousands of time steps.

A LSTM unit is typically composed of a cell and three gates: input, output, and forget. The cell remembers values over arbitrary time intervals, and the gates regulate the flow of information into and out of the cell. Forget gates decide what information to discard from the previous state, by mapping the previous state and the current input to a value between 0 and 1. Output gates control which pieces of information in the current cell state to output, by assigning a value from 0 to 1 to the information, considering the previous and current states.

Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time-steps.

Autoencoder

Originally proposed in Rumelhart et al. (1986), and then first implemented in Elman and Zipser (1988), originally for speech recognition purposes, an autoencoder is a type of ANN used for unsupervised learning.

An autoencoder learns two functions: an encoding function that transforms the input data, and a decoding function that recreates the input data from the encoded representation. The autoencoder learns and efficient representation for a set of data, typically for dimensionality reduction, to generate lower-dimensional embeddings for subsequent use by other machine learning algorithms. Autoencoders can be also used to randomly generate new data similar to the training data. This feature has made them very useful for generating synthetic data to train other ML models. In finance, autoencoders are vastly employed for tasks such as factor extraction, risk modeling, and covariance estimation.

Convolutional Neural Network

The first trainable Convolutional Neural Network—hereinafter CNN—was proposed in Fukushima (1980). A CNN is a type of feedforward neural network—hence it does not present any recurrences—that learns features through kernel optimization.

A CNN consists of an input layer, one or more hidden layers, and an output layer. Hidden layers perform convolutions through trainable kernels that scan local regions of the input, enabling automatic feature extraction. The output of each convolutional layer is then passed to the next layer. Convolutional layers work similarly to neurons in the visual cortex.

Originally developed for image processing and recognition, CNNs are becoming interesting from a financial perspective. Their impressive feature extraction ability makes them adapt to be used for tasks such as pattern recognition or technical analysis encoding.

Generative Adversarial Network

Originally proposed in Goodfellow et al. (2014), Generative Adversarial Networks—GANs for short—are a class of artificial neural networks consisting of two competing networks: the generative network and the discriminative network. The generator learns to produce synthetic data resembling the training samples, similarly to autoencoders, while the discriminator learns to distinguish generated data from real observations. Through this adversarial process, both networks iteratively improve their performance, allowing the generator to produce increasingly realistic outputs.

The ability of generating high quality synthetic data is particularly valuable in financial settings, where GANs are used in portfolio stress testing, option pricing, and, in the context of our thesis, generate investors' views.

2.3 Reinforcement Learning

While the Black-Litterman model serves as the theoretical foundation for our portfolio allocation problem, reinforcement learning (hereinafter RL) serves as the workhorse for our experiment. The goal of this section is to provide the reader with some intuitions and concepts about the vast field of reinforcement learning, which will be handy to have a better understanding of the models and methods used throughout this study, the description of which is relegated to the next sections. This section is based on the second edition of Sutton et al. (1998). For a more thorough discussion of reinforcement learning, the reader shall refer to the aforementioned book.

2.3.1 What is Reinforcement Learning?

Reinforcement learning is learning how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Actions may not affect only the immediate reward, but also the next situation and all subsequent rewards. Trial-and-error and delayed reward are the two most important features of reinforcement learning. A learning agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state. The agent must also have a goal or goals relating to the state of the environment.

Reinforcement learning is different from *supervised learning*, which consists in learning from a training set of labeled examples provided by a knowledgeable supervisor. RL is also different from *unsupervised learning*, which is typically about finding structure hidden in collections of unlabeled data, in which RL is trying to maximize a reward signal instead of trying to find hidden structures. We therefore consider reinforcement learning to be a third machine learning paradigm, alongside supervised and unsupervised learning, and perhaps other paradigms.

One of the challenges that arise in reinforcement learning is the tradeoff between exploration and exploitation. To obtain a lot of reward, an agent must prefer actions it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions never selected before. The agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. Neither exploration nor exploitation can be pursued exclusively without failing the task.

Another key feature of reinforcement learning is that it starts with a complete, interactive, and goal-seeking agent. All RL agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed that the agent has to operate despite significant uncertainty about the environment it faces.

2.3.2 Elements of Reinforcement Learning

Beyond the agent and the environment, one can identify four main sub-elements of a reinforcement learning system: a *policy*, a *reward signal*, a *value function*, and a *model* of the environment.

A *policy* defines the learning agent's way of behaving at a given time. A policy is a mapping from perceived states of the environment to actions to be taken when in those

states. In our case, the policy for our RL system will map the state of the market to the views the agent will form about the performances of the stocks composing our portfolio, which will be integrated in the BL model.

A *reward signal* defines the goal of the RL problem. On each time step the environment sends to the agent a single number called the *reward*. The agent sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. In financial applications the Sharpe ratio of the portfolio (Sharpe (1998)) has been widely used as a reward signal, or, more specifically, its variation between rebalancings. However, our approach is different, and it will be discussed in Subsection 3.5.1.

Whereas the reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. The *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Values indicate the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states.

Models mimic the behavior of the environment, or more generally, allow inferences to be made about how the environment will behave. Given a state and action, the model might predict the resultant next state and next reward. Models for solving RL problems that use models are called *model-based* methods, and are opposed to *model-free* methods, that are explicitly trial-and-error learners.

2.3.3 Markov Decision Processes

Markov Decision Processes (hereinafter MDPs) are a mathematically idealized form of the reinforcement learning problem from which precise theoretical statements can be made. MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called the *agent*, and the thing it interacts with is called the *environment*, which also gives rise to rewards, as described in the previous two subsections.

More specifically, the agent and the environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, ...$ At each time step t the agent receives some representation of the environment *state*, $S_t \in \mathcal{S}$, and on that basis selects an *action*, $A_t \in \mathcal{A}(s)$. One time step later, the agent receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

In a *finite* MDP, the sets of states, actions, and rewards ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) all have finite number of elements. In this case the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. For a particular values of those random variables, $s' \in \mathcal{S}$ and $r' \in \mathcal{R}$, there is a probability of those values of occurring at time t, given particular values of the preceding state and action:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{2.26}$$

for all s', $s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$. The function p defines the *dynamics* of the MDP.

In a MDP, the probabilities given by p characterize the environment's dynamics. That is, the probability of each possible value for S_t and R_t depends only on the immediately

preceding state and action, S_{t-1} and A_{t-1}. This is best viewed as a restriction on the state, and not on the decision process. The state must include all information about all aspects of the past agent-environment interaction that make a difference for the future. If it does, it is said to have the *Markov property*.

2.3.4 Returns and Episodes

In RL problems, one wants to maximize *expected return*, where the return G_t is defined as some specific function of the reward sequence. In the simplest case, the return is the sum of the rewards:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots R_T \tag{2.27}$$

where T is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into sub-sequences, which we call *episodes*, terminating in a *terminal state* followed by a reset to a standard state. Tasks with episodes of this kind are called *episodic tasks*.

However, in many cases the agent-environment interaction does not break naturally into identifiable episodes but goes on continually without limit. We call these *continuing tasks*. It is evident that the return formulation (2.20) is problematic for continuing tasks, since without a final time step the return we seek to maximize could be infinite [4]. The return formulation must be modified incorporating the concept of *discounting*. According to this approach the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses A_t to maximize the expected *discounted return*:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.28}$$

where $0 \leq \gamma \leq 1$ is a parameter called the *discount rate*. If $\gamma < 1$, the infinite sum in (2.21) has a finite value as long as the reward sequence $\{R_k\}$ is bounded. If $\gamma = 0$ the agent is "myopic" in being concerned only with maximizing immediate rewards. As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted. Notice that the returns at successive time steps can be related as:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2.29}$$

2.3.5 Policies and Value Functions

Almost all RL algorithms involve estimating *value functions*. Value functions are functions of states, or of state-action pairs, that estimate how good is for the agent to be in a given state, or how good is to perform a given action in a given state. The "goodness" here is defined in terms of expected rewards. The rewards the agent can expect to receive in the future depend on what actions it will take. Hence, value functions are defined with respect to particular ways of acting, called policies.

As introduced in subsection 2.2.2, a *policy* is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a|s)$

[4] The final time step would be $T = \infty$

is the probability that $A_t = a$ if $S_t = s$. In other words, $\pi(a|s)$ is the probability of the agent taking a given action conditional on a given state.

The *value function* of a state s under a policy π, denoted $v_\pi(s)$ is the expected return when starting in s and following π thereafter. Formally, for MDPs:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \quad \forall s \in \mathcal{S} \tag{2.30}$$

We call $v_\pi(s)$ the *state-value function* for policy π. We can also rewrite $v_\pi(s)$ as:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \quad \forall s \in \mathcal{S} \tag{2.31}$$

where it is implicit that the actions, a, are taken from the set $\mathcal{A}(s)$, the next states, s', are taken from the set $\mathcal{S}$, and the rewards, r, are taken from the set $\mathcal{R}$. This equation is known as the *Bellman equation* for v_π.

Similarly, we define the value of taking action a in state s under a policy π, as the expected return starting from s, taking action a, and thereafter following policy π:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \tag{2.32}$$

We call q_π the *action-value function* for policy π.

Solving a RL task means finding a policy that achieves a lot of reward over the long run. For finite MDPs, value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if the expected return is greater than or equal to that of π' for all states. There is always one policy which is better than or equal to all other policies. This is an *optimal policy*, denoted by π_*. All optimal policies share the same state value function, called the *optimal state-value function*, denoted v_*, and defined as

$$v_*(s) \doteq \max_\pi v_\pi(s), \tag{2.33}$$

for all $s \in \mathcal{S}$.

Optimal policies also share the same *optimal action-value function*, denoted q_*, and defined as

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \tag{2.34}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. For the state-action pair (s, a), this function gives the expected return for taking action a in state s and thereafter following the optimal policy. Thus, we can write q_* in terms of v_* as follows:

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]. \tag{2.35}$$

From equation 2.24 one can derive:

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right] \tag{2.36}$$

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \tag{2.37}$$

2.3.6 Dynamic Programming

The term Dynamic Programming (hereinafter DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP. The key idea of DP is the use of value functions to organize and structure the search for good policies. Despite their theoretical importance, DP models are rarely used in practice due to both their assumption of a perfect model and because of their great computational expense. When presenting DP algorithms we'll make the assumption that the environment is a finite MDP.

Policy evaluation refers to the iterative computation of the value functions for a given policy. *Policy improvement* refers to the computation of an improved policy given the value function for that policy. Putting these two computation together we encounter our first DP method, *policy iteration*.

Iterative policy evaluation begins with an arbitrary initial approximation for the value function of policy π, v_0. Each successive approximation is obtained by using the Bellman equation for v_π as an update rule:

$$v_{k+1} = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right], \tag{2.38}$$

for all $s \in \mathcal{S}$. The sequence $\{v_k\}$ can be shown to converge to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π. To produce each successive approximation, v_{k+1} from v_k, iterative policy evaluation applies the same operation to each state s: it replaces the old value of s with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. The general idea of updating estimates on the basis of other estimates is called *bootstrapping*.

Policy improvement, as the name implies, improves the deterministic policy π on the basis of the value function v_π we have computed. The *policy improvement theorem* states that if:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s),$$

where π and π' are any pair of deterministic policies, then π' must be as good as, or better than, π. In other words:

$$v_{\pi'}(s) \geq v_\pi(s), \quad \forall s \in \mathcal{S}.$$

The way policy improvement works is by selecting the new *greedy* policy, π', given by

$$\pi'(s) \doteq \operatorname*{argmax}_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]. \tag{2.39}$$

The greedy policy takes the action that looks best in the short term according to v_π. By construction, the greedy policy meets the conditions of the policy improvement theorem, so we know that it is as good as, or better than, the original policy. Policy improvement hence consists in the process of making a policy greedy with respect to the value function of the original policy.

Policy iteration, as anticipated before, consists in the sequence of policy evaluations and policy improvements. In essence, once a policy π has been improved using v_π to yield a better policy π', we can then compute $v_{\pi'}$ and improve it again to yield π'', until we arrive to v_*. We can represent the process of policy improvements as:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*.$$

One major drawback of policy iteration is that its iterations involve policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set, a major computational bottleneck. Thankfully, the policy evaluation step can be truncated after just one sweep without loosing the convergence guarantees of policy iteration. This algorithm is called *value iteration*, and can be written as:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right]. \tag{2.40}$$

The sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_*. Formally, value iteration would require an infinite number of time steps to converge, but in practice we stop once the value function changes only by a small amount in a sweep. Value iteration effectively combine one sweep of policy evaluation and one sweep of policy improvement.

Adding to the theoretical importance of DP methods, is the fact that insights about almost all RL methods can be gained by viewing them as *generalized policy iteration* (GPI). GPI is the general idea of two interacting processes revolving around an approximate policy and an approximate value function. One process performs some form of policy evaluation, while the other performs some form of policy improvement. In some cases, GPI can be proved to converge, while in the cases in which convergence has not been proven GPIs are useful to improve our understanding of the methods.

2.3.7 Monte Carlo Methods

Differently from DP methods, Monte Carlo methods (MC) solve RL problems by averaging sample returns. While DP methods require a complete model for the environment, MC methods require only *experience*, represented by sample sequences of states, actions, and rewards from actual or simulated interaction with the environment. MC methods sample and average *returns* for each state-action pair and average *rewards* for each action. While DP aims at computing value functions from knowledge of the MDP, MC methods aim at *learning* value functions from sample returns with the MDP. To ensure that well-defined returns are available, here we define MC methods only for episodic tasks. MC methods can thus be incremental in an episode-by-episode sense, but not in a step-by-step sense.

Each occurrence of a state s in an episode is called a *visit* to s. Let us call the first time s is visited in an episode the *first visit* to s. The *first-visit MC method* estimates $v_\pi(s)$ as the average of the returns following first visits to s, whereas the *every-visit MC method* averages the returns following all visits to s. Both methods converge to $v_\pi(s)$ as the number of visits to s goes to infinity.

In the absence of a model for the environment it is particularly useful to estimate *action* values rather than *state* values, since without a model state values alone are no longer sufficient. One must explicitly estimate the value of each action in order for the values to be useful in suggesting a policy. Therefore, one of the main goals for MC methods is to estimate q_*. In other words, the policy evaluation problem for action values is to estimate $q_\pi(s, a)$, the expected return when starting in state s, taking action a, and thereafter following policy π. Both first and every visit MC methods converge to the true expected values of q_π as the number of visits to each state-action pair approaches infinity.

The only complication comes from the fact that, if π is a deterministic policy, then in following π one will observe returns only for one of the actions from each state. Obviously, with no returns to average the MC estimates of the other actions will not improve with experience. To compare alternatives we need to estimate the value of *all* the actions from each state, not just the one we currently favor. This is the general problem of *maintaining exploration*. One way to do this is by specifying that the episodes start in a state–action pair, and that every pair has a nonzero probability of being selected as the start. This guarantees that all state–action pairs will be visited an infinite number of times in the limit of an infinite number of episodes. We call this the assumption of *exploring starts*. The most common alternative approach to assuring that all state–action pairs are encountered is to consider only policies that are stochastic with a nonzero probability of selecting all actions in each state.

The idea behind optimal policy approximation with MC methods is similar to the one presented for DP in the previous section, and can be described as a GPI. In a nutshell, we perform alternating complete steps of policy evaluation and policy improvement:

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*$$

Policy evaluation is done exactly as described in the preceding section. Policy improvement is done by making the policy greedy with respect to the current value function. For any action-value function q, the corresponding greedy policy is the one that, for each $s \in \mathcal{S}$, deterministically chooses an action with maximal action-value:

$$\pi(s) \doteq \operatorname*{argmax}_a q(s, a) \tag{2.41}$$

We made two unlikely assumptions above in order to easily obtain this guarantee of convergence for the Monte Carlo method. One was that the episodes have exploring starts, and the other was that policy evaluation could be done with an infinite number of episodes. To obtain practical algorithms those two hypotheses must be removed. While the latter is relatively trivial to remove, since it is possible to truncate the number of iterations and still obtain a good approximation, the former is more intractable, since the only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them. There are two approaches to ensuring this, resulting in what we call *on-policy* methods and *off-policy* methods. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.

2.3.8 Temporal Difference Learning

Temporal Difference (TD) learning is a combination of MC and DP ideas. Like MC methods, TD methods can learn directly from raw experience, and like DP, they update estimates based in part on other learned estimates, without waiting for a final outcome (bootstrapping).

Both TD and Monte Carlo methods use experience to solve the prediction problem. Given some experience following a policy π, both methods update their estimate V of v_π for the non-terminal states S_t occurring in that experience. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$, TD methods need to wait only until the next time step. The simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{2.42}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. This TD method is called TD(0), and it is a special case of the TD(λ) methods. Because TD(0) bases its update in part on an existing estimate, we say that it is a bootstrapping method, like DP. The term between square brackets, which compact notation is δ_t, is called the *TD error*, and it arises in various forms throughout RL.

Importantly, for any fixed policy π, TD(0) has been proven to converge to v_π, in the mean for a constant step-size parameter (α) if it is sufficiently small, and with a probability of 1 if the step-size decreases according to stochastic approximation conditions. Furthermore, in practice TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks.

With regard to the control problem for TD learning, we will present two different approaches. The first one is Sarsa, an on-policy TD control method. Sarsa involves learning an action-value function instead of a state-value function by means of the algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]. \tag{2.43}$$

This update is done after every single transition from a non-terminal state S_t. As in all on-policy methods, we continually estimate q_π for the behavior policy π, and at the same time change π toward greediness with respect to q_π.

The off-policy alternative to Sarsa is *Q-learning* (Watkins and Dayan (1992)), defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t\right]. \tag{2.44}$$

In this case, the learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed.

2.3.9 Function Approximation and Stochastic Gradient Methods

In many of the tasks to which we would like to apply reinforcement learning the state space is combinatorial and enormous; within the scope of this thesis, for example, the state space, represented by the state of the market, is potentially infinitely large, even considering only a small number of stocks. In such cases we cannot expect to find an optimal policy or the optimal value function even in the limit of infinite time and data; our goal instead is to find a good approximate solution using limited computational resources. To make sensible decisions in large state spaces it is necessary to generalize from previous encounters with different states that are in some sense similar to the current one. In other words, the key issue is that of *generalization*. The kind of generalization we require is often called *function approximation* because it takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation of the entire function.

The novelty in this section is that the approximate value function is represented as a parametrized functional form with weight vector $\mathbf{w} \in \mathbb{R}^d$. We will write $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$ for the approximate value of state s given weight vector $\mathbf{w}$. The same applies to the action-value function, which we will write as $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$.

Stochastic Gradient Descent (hereinafter SGD) methods are among the most widely used of all function approximation methods and are particularly well suited to online RL. In gradient-descent methods the weight vector is a column vector with a fixed number of real valued components, $\mathbf{w} \doteq (w_1, w_2, ..., w_d)'$, and the approximate value function $\hat{v}(s, \mathbf{w})$

is a differentiable function of $\mathbf{w}$ for all $s \in \mathcal{S}$. The weight vector at each time step is denoted as $\mathbf{w}_t$, for $t = 0, 1, 2, ...$

Before we jump onto describing SGD methods, it is useful to define the *Mean Squared Value Error*, denoted $\overline{VE}$, which is given by:

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w})\right]^2, \tag{2.45}$$

where $\mu(s) \geq 0$, $\sum_s \mu(s) = 1$, is the state distribution, representing how much we care about the error in each state s. Ideally, we would like to find a *global optimum*, a weight vector $\mathbf{w}^*$ for which $\overline{VE}(\mathbf{w}^*) \leq \overline{VE}(\mathbf{w})$ for all possible $\mathbf{w}$.

SDG methods try to minimize error on the observed sample by adjusting the weight vector after each example by a small amount in the direction that would most reduce the error on that example:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right]^2 \tag{2.46}$$

$$= \mathbf{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right] \nabla \hat{v}(S_t, \mathbf{w}_t), \tag{2.47}$$

where α is a positive step-size parameter, and $\nabla f(\mathbf{w})$, for any scalar expression $f(\mathbf{w})$ that is a function of a vector, denotes the column vector of first partial derivatives of the expression with respect to the components of the vector:

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, ..., \frac{\partial f(\mathbf{w})}{\partial w_d}\right)' \tag{2.48}$$

This derivative vector is also called the *gradient* of f with respect to $\mathbf{w}$. SDG methods are "gradient descent" methods because the overall step in $\mathbf{w}_t$ is proportional to the negative gradient of the example's squared error. This is the direction in which the error falls most rapidly. By iteratively applying equation (2.40), $\mathbf{w}_t$ will eventually converge to $\mathbf{w}^*$.

In the case in which the target output, here denoted $U_t \in \mathbb{R}$, of the tth training example, $S_t \mapsto U_t$, is not the true value, $v_\pi(S_t)$, but some, possibly random, approximation to it, we cannot perform the exact update, but we can approximate it by substituting U_t in place of $v_\pi(S_t)$ in equation (2.40). This yields:

$$\mathbf{w}_t + \alpha \left[U_t - \hat{v}(S_t, \mathbf{w}_t)\right] \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{2.49}$$

If U_t is an unbiased estimate, then $\mathbf{w}_t$ is guaranteed to converge to a local optimum for decreasing α.

2.4 Training Algorithms

As mentioned in Section 2.2.2, the error backpropagation algorithm is unsuitable to train the large neural networks we will use throughout our study, both for its tendency to overfit and, most notably, for its tendency to suffer from the vanishing gradient problem, due to the rapid decay of the partial derivatives toward the input side of the network. Therefore, in our study we will use two different algorithms to train our agents: an on-policy algorithm, PPO, and an off-policy algorithm, SAC. This section aims at presenting in detail the two algorithms.

2.4.1 Proximal Policy Optimization - PPO

Proximal Policy Optimization (PPO) (Schulman et al. (2017)) is a policy gradient method for RL, originally proposed as a successor to TRPO (Schulman et al. (2015)), another policy gradient method, with the aim to retaining TRPO's data efficiency and reliability, while concurrently simplifying the framework. Due to its simplicity and versatility, PPO has rapidly become one of the more widely used RL algorithms, finding applications in diverse fields, from robotics to video-game engines, and so on. The goal of this section is to present a detailed overview of PPO, based on the original paper from Schulman et al. (2017).

It is first useful to recall that policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm, where the most commonly used gradient estimator has the form:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right], \tag{2.50}$$

where $\hat{A}_t$ is an estimator of the advantage function at timestep t.Here, the expectation $\hat{\mathbb{E}}_t$ indicates the empirical average over a finite batch of samples, in an algorithm that alternates between sampling and optimization. Implementations that use automatic differentiation software work by constructing an objective function whose gradient is the policy gradient estimator; the estimator $\hat{g}$ is obtained by differentiating the objective

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \tag{2.51}$$

In TRPO (the predecessor to PPO), an objective function is maximized subject to a constraint on the size of the policy update. Specifically,

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \tag{2.52}$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right] \leq \delta \tag{2.53}$$

where θ_{old} is the vector of policy parameters before the update. According to the authors, the theory justifying TRPO actually suggests using a penalty instead of a constraint, i.e., solving the unconstrained optimization problem

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right] \tag{2.54}$$

for some coefficient β. TRPO uses a hard constraint rather than a penalty because it is hard to choose a single value of β that performs well across different problems—or

even within a single problem, where the the characteristics change over the course of learning. Hence, to achieve the goal of a first-order algorithm that emulates the monotonic improvement of TRPO (i.e. PPO),it is not sufficient to simply choose a fixed penalty coefficient β and optimize the penalized objective equation with SGD, but additional modifications are required.

Now that we have briefly introduced the TRPO framework, it is time to derive the PPO one. First, we let $r_t(\theta)$ denote the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{2.55}$$

so that $r_t(\theta_{\text{old}}) = 1$. We can now say that TRPO maximizes the "surrogate" objective

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t]. \tag{2.56}$$

Without a constraint the maximization of $L^{CPI}(\theta)$ would lead to an excessively large policy update; hence, the authors now considered how to modify the objective, to penalize changes to the policy that move $r_t(\theta)$ away from 1. The main objective they proposed is the following:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{2.57}$$

where ϵ is a hyperparameter [5]. The first term inside the minimum is nothing else than L^{CPI}. The second term modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. Finally, the minimum of the clipped and unclipped objective is taken, so the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective. With this scheme, we only ignore the change in probability ratio when it would make the objective improve, and we include it when it makes the objective worse. This way we avoid having too large of a policy update.

Another alternative approach proposed by the authors is the *adaptive KL penalty coefficient*, which uses a a penalty on KL divergence, and adapts the penalty coefficient so that some target value of the KL divergence d_{targ} are achieved each policy update. This alternative approach is not reported here in full since it has found to perform worse than the clipped surrogate objective just described. For a more in depth perspective on this second approach the reader is referred to Schulman et al. (2017).

For implementations that use automatic differentation, one simply constructs the loss L^{CLIP}, and one performs multiple steps of stochastic gradient ascent on this objective. If using a neural network architecture that shares parameters between the policy and value function, we must use a loss function that combines the policy surrogate and a value function error term. The authors suggest augmenting this objective by adding an entropy bonus to ensure sufficient exploration. The final objective is, therefore:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \tag{2.58}$$

where c_1 and c_2 are coefficients, S denotes the entropy bonus, and L_t^{VF} is a squared error loss $\left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2$.

One style of policy gradient implementation, runs the policy for T time steps (where T is much less than the episode length), and uses the collected samples for an update.

[5] There is no single correct value for ϵ. The authors suggest setting it to 0.2

This style requires an advantage estimator that does not look beyond timestep T. The estimator proposed by Mnih et al. (2016) is

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{2.59}$$

where t specifies the time index in $[0, T]$. Generalizing this choice, we can use a truncated version of generalized advantage estimation, which reduces to equation 2.52 when $\lambda = 1$:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2.60}$$

where $\delta = r_t + \gamma V(s_{t+1}) - V(s_t)$. The PPO algorithm proposed by the authors is contained in the next table.

Algorithm 1 PPO, Actor-Critic Style

for iteration = 1,2,... **do**
  for actor = 1,2,...,N **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

2.4.2 Soft Actor-Critic - SAC

The Soft Actor-Critic (SAC) method has been originally proposed by Haarnoja et al. (2018). SAC is an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework, where the actor aims to maximize expected reward while also maximizing entropy. That is, to succeed at the task while acting as randomly as possible.

Before delving in the technicalities of SAC, it is worth explaining briefly how the Actor-Critic (AC) framework works. In AC algorithms (Konda and Tsitsiklis (1999)) the agent is split in two different components, which can be represented by two neural networks, the *actor*, and the *critic*. The actor is responsible for taking actions with the goal to find the best overall policy. The critic, on the other hand, does not take any actions. Instead, it watches the actor make its choices and evaluates the state[6]. The critic uses TD learning with a linear approximation architecture[7] and the actor is updated in an approximate gradient direction based on information provided by the critic. The interaction between the actor and the critic is what improves the policy.

The SAC method uses function approximators for both the Q-function, the *critic*, and the policy, the *actor*. Furthermore, SAC alternate between optimizing both networks with stochastic gradient descent. The method considers a parametrized state value function $V_\psi(s_t)$, a soft Q-function $Q_\theta(s_t, a_t)$, and a tractable policy $\pi_\phi(a_t \mid s_t)$. The parameters of these networks are ψ, θ, and ϕ. The soft value function is trained to minimize the squared residual error, defined as:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t \mid s_t)])^2\right] \tag{2.61}$$

[6] In the SAC method the critic evaluate also the action value.

[7] In the most modern AC methods the critic more commonly uses deep neural networks.

where $\mathcal{D}$ is the distribution of previously sampled states and actions, or a replay buffer. The gradient of the previous equation can be estimated with the unbiased estimator:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t) \left(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t \mid s_t) \right), \tag{2.62}$$

where the actions are sampled according to the current policy, instead of the replay buffer. The soft Q-function parameters can be trained to minimize the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right], \tag{2.63}$$

with

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}(s_{t+1})} \right], \tag{2.64}$$

which again can be optimized with stochastic gradients

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t) \left(Q(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}(s_{t+1})} \right). \tag{2.65}$$

The update makes use of a target value network $V_{\bar{\psi}}$, where $\bar{\psi}$ can be an exponentially moving average of the value network weights. the policy parameters can be learned by directly minimizing the expected KL-divergence:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi(\cdot \mid s_t) \middle\| \frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)} \right) \right] \tag{2.66}$$

While there are several options for minimizing J_π, in the case of SAC it is useful to re-parametrize the policy using a neural network transformation:

$$a_t = f_\phi(\epsilon_t; s_t) \tag{2.67}$$

where ϵ_t is an input noise vector. The objective in equation (2.59) can be rewritten as:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} [\log \pi_\phi(f_\phi(\epsilon_t; s_t) \mid s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))] \tag{2.68}$$

We can approximate the gradient of this equation with:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t \mid s_t) + (\nabla_{a_t} \log \pi_\phi(a_t \mid s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t) \tag{2.69}$$

The algorithm proposed in Haarnoja et al. (2018) follows in the next table. The algorithm also makes use of two Q-functions to mitigate positive bias in the policy improvement step that is known to degrade performance of value based methods. The authors found two Q-functions significantly speed up training, especially on harder tasks. The method alternates between collecting experience from the environment with the current policy and updating the function approximators using the stochastic gradients from batches sampled from a replay buffer.

Algorithm 2 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t \mid s_t)$
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i), \ i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Chapter 3

Research Goals and the Experiment

As already mentioned in Chapter 1, the goal of this study is to assess the ability of Reinforcement Learning trained agents to manage dynamic equity portfolios by means of an underlying asset allocation model, alongside considering the impact this could have on the theoretical foundations upon which modern finance rests. In order to achieve our goal, an experiment is clearly needed.

This thesis should be understood as experimental in its nature. In brief, our experiment will consist of four families of RL agents managing dynamically a diversified portfolio of eight stocks by outputting absolute views, and associated confidence, to be used as inputs for the BL model for portfolio optimization. Families differ from each other in the algorithm they are trained with, either PPO or SAC, and in the dataset they are fed, either narrow or wide. Each family possess four sub agents with different look-back windows. The experiment will involve the trained agents acting on seven different backtesting windows, spanning from January 1st, 2024, to December 31st, 2025. Results from the experiment and the conclusions we will draw will be presented in Chapter 4.

While in Chapter 2 we presented the theoretical foundations of this study, this Chapter is devolved to the presentation of our experiment. Section 3.1 presents briefly our research goals, provides with a few definitions which will be useful throughout this study, and provides a brief presentation of our agents. Section 3.2 provides with the detailed descriptions of the rationale behind the research questions presented in the preceding section and with the methodology we will use in our experiment to answer them. Section 3.3 will present the data and their organization to the reader, clarifying the distinction between narrow and wide datasets. Subsection 3.5.1 defines the reward function we will use to train our agents. Finally, Section 3.5 gives some technical details about the implementation of the experiment in Python.

3.1 Research Goals

The experiment we propose consists in the development of different reinforcement learning agents, trained on the portfolio selection problem through the application of the Black-Litterman model, which we described in Section 2.1. Recall that the BL model for portfolio selection involves inferring a posterior by incorporating investor's views in the equilibrium prior. Therefore our agents will be trained in the formation of views coherent with the state of the market.

Our agents will differ between one another by the algorithms used for their training, and by the amplitude of the dataset used to train them. To the extent of this thesis, we will define as *narrow* datasets containing only stock prices, and as *narrow agents* the agents trained on such datasets. On the other hand, *wide* datasets will contain, on top of stock prices, also technical indicators, macroeconomic indicators, spreads, and market volatility measures, and are designed to provide the agent with more information about the state of its environment (the market) than narrow datasets. We will call *wide agents* the agents trained on such datasets. A detailed description of the datasets used for our experiment is postponed to later sections. Table 3.1 summarize the four agents we will train.

Agent Name	**Agent Type**	**Training Algorithm**
Agent Alpha	Narrow	PPO
Agent Bravo	Narrow	SAC
Agent Charlie	Wide	PPO
Agent Delta	Wide	SAC

Table 3.1: Summary of reinforcement learning agents.

To the extent of this study, each couple of agents, composed by narrow and wide agents, will serve a specific purpose. Specifically, Table 3.2 provides with a detailed description of the five main goals we aim to achieve with our study. The rationale behind each one of our five goals, alongside the experiment's methodology will be described in the following section.

Goal	**Goal Description**
1.	Evaluate the ability of a *narrow* agent to form accurate views consistent with the state of the market.
2.	Evaluate the ability of a *wide* agent to form accurate views consistent with the state of the market **and** able to generate portfolios performing better than some benchmarks.
3.	Evaluate the ability of both *narrow* and *wide* agents to build portfolios able to outperform naively diversified ones.
4.	Evaluate the performance of dynamic portfolio management conducted by RL agents with respect to buy-and-hold strategies.
5.	Thoroughly analyze the differences between the two training algorithms employed in this study.

Table 3.2: Research goals of this study.

3.2 Methodology of the Experiment

Referring to Table 3.2 for this section, we will discuss each single goal individually. Before delving in depth on the topic of this section, it is useful to briefly mention the assets that will compose our portfolios and the underlying assumptions of our study. First, each one of the portfolios we will build and analyze will be composed of eight different stocks, all of them listed on the two bigger American stock exchanges[1]. Table 3.3 lists all eight companies, alongside their tickers, sector and stock exchange.

Company name	**Ticker**	**Sector**	**Exchange**
Apple	AAPL	Tech	NASDAQ
JPMorgan	JPM	Financials	NYSE
Johnson & Johnson	JNJ	Healthcare	NYSE
Coca Cola	KO	Consumer Staples	NYSE
Walmart	WMT	Consumer Staples	NASDAQ
Caterpillar	CAT	Industrials	NYSE
ExxonMobil	XOM	Energy	NYSE
Air Products & Chemicals	APD	Materials	NYSE

Table 3.3: The stocks considered for this study.

These eight companies allow us to build *decently*-diversified portfolios. Furthermore, a portfolio built using these stocks, each one of them coming from a different industrial sector, can be assumed to be representative enough with respect to the market. Finally, note that all the chosen companies are American. This homogeneity in the geographic distribution of the considered stocks will allow us to draw some interesting insights regarding the first goal of our study, as it will be described in the following paragraphs. Clearly, for a more realistic, and practically applicable, analysis more companies will be taken into consideration. However, limiting at eight the number of considered stocks is needed due to computational constraints.

To the extent of our experiment, two notable assumptions must be made. Following on the example of Sun et al. (2024b), we make two simplifying, yet realistic, assumptions:

1. *Zero Market Impact*: The trading actions made by our agents do not affect the asset price fluctuation.

2. *Zero Slippage*: The selected assets are highly liquid. Hence, both long and short trading orders are executed immediately.

Notice that both assumptions are completely realistic for the considered stocks. First, all of them are *blue chip* stocks with huge trading volumes, therefore they can be considered highly liquid. The first assumption being valid follows from the second one being realistic.

3.2.1 First Goal

It is now time to discuss the rationale behind our goals and the methodology we will follow to achieve them. Our first goal is concerned with the ability of a narrow agent to form views consistent with the state of the market. Recall that our two narrow agents will be

[1]The NYSE and the NASDAQ.

trained on datasets containing solely the time series for the prices of our eight considered stocks, which are transformed to returns internally by the environment for stability. In other words, we are interested in understanding if past returns contain information useful for the agent to make valid predictions about the future performances of our considered assets. Therefore, our first goal can be interpreted as a test of the weak form of market efficiency, as described in Fama (1970). The reader may recall that under weak market efficiency, the current stock price reflects all information contained in past relevant market data. If a market is weakly efficient it is impossible to predict the future performances of a stock based on its past performances. It follows directly that, if the market is weakly efficient, our narrow agents shall not be able to form accurate views, considering that a view is nothing more than a prediction about future performances which may or may not be accurate.

So, how do we plan to tackle our first goal? First, agents Alpha and Bravo will be trained on the same narrow dataset. Multiple agents will be developed, each one of them having a different numbers of neurons in their ANNs. Then, all of the agents will be backtested on the same backtesting window. The performances of the portfolios they will manage will be confronted with the ones coming from passive indexes of our pseudo-market of eight stocks, and with the CAPM prior portfolio of the BL model. Furthermore, to assess whether narrow agents have learned a policy able to extract information reliably from their narrow observation space, the ECDFs of their returns from stochastic simulations will be tested for stochastic greatness and First-order Stochastic Dominance—hereinafter FSD—against the ECDFs of the returns of randomly weighted portfolios.

Clearly, if our pseudo-market is weakly efficient, we expect all our agents' portfolios to underperform market indexes and their returns to be, most of the times, stochastically inferior to random returns. On the other hand, if agents' portfolios consistently outperform indexes and their returns deemed to be stochastically greater, or even achieving FSD, over the random baseline, this could imply some sort of inefficiency within our pseudo-market, considering that the only information our agents could have leveraged to form views was necessarily contained in past returns. Whichever the case, the reader must be advised that any conclusion drawn from our first goal is far from constituting any conclusive evidence in favor or against weak market efficiency, mainly due to the small number of stocks involved. Nevertheless, we believe that interesting insights can be drawn from our experiment regarding this subsection's topic.

A complete and thorough discussion of the answer we provide to our first research goal can be retrieved in Section 4.3.1.

3.2.2 Second Goal

While our first goal is concerned with weak-form market efficiency, our second goal is concerned with the ability of wide agents to form accurate and consistent views. Recall that wide agents are trained on a dataset containing, on top of stock prices, technical and macroeconomic indicators. Wide datasets are therefore designed to provide the agent with information about the overall state of the market. Hence, the aim for wide agents is to learn to form views based on historical market conditions, not only on past stock prices and trends. For example, we expect agents Charlie and Delta to learn policies leading to the overweighting of defensive stocks in periods of market downturns, while preferring aggressive stocks in bullish markets. Furthermore, we expect wide agents to

learn heuristics dependent on the market state, in accordance with the Adaptive Market Hypothesis—hereinafter AMH—proposed in Lo (2004).

The methodology in tackling this second goal follows from the previous subsection. Again, the two wide agents will be trained on the same wide dataset. agents will have different sizes of neural networks. Once the training will be completed the agents will be backtested on the same wide backtesting dataset, and the performances of the portfolios they will manage will be compared to the ones from passive indexes of our pseudo-market, the Equally Weighted portfolio, which serves as a proxy for the BL prior, and random returns in FSD testing.

If our agents are able to form accurate views which are consistent with the state of the market we could expect the performances of the agents' actively managed portfolios to be at least as good as, or even better than, the benchmarks'. This is due to the fact that in the BL model, views are the ones who actually shape the resulting portfolio. Accurate views improve the performance of the portfolio with respect to the prior, while inaccurate, or outright wrong views impair the resulting portfolio's performance.

A complete and thorough discussion of the answer we provide to our second research goal can be retrieved in Section 4.3.2.

3.2.3 Third Goal

To the extent of this discussion, we refer to *naively diversified* portfolios as the static, long only, portfolios built using diversification strategies that a naive, unsophisticated investor could follow. In particular, we consider two naive diversification strategies, namely equal-weighting and random-weighting. Formally, we define an equally weighted portfolio as having weights for each one of the n stocks composing it being:

$$w_i = \frac{1}{n}, \quad \forall i = 1, ..., n. \tag{3.1}$$

Randomly weighted portfolios are instead defined, for the purposes of this discussion, as all of the long only portfolios whose weights are chosen randomly by the investor. In other words, all portfolios built without following any specific diversification strategy are randomly weighted portfolios. The family of randomly weighted portfolios may include the ones which present an overweighting of a particular stock, representing an entire asset class, due to personal preferences of the naive investor. For example, a tech loving investor can build a representative portfolio with an overweighting of Apple stock in it.

Randomly weighted portfolios are built by sampling individual weights for each stock from a uniform distribution bounded between 0 and 1. Then they are standardized to sum to one. Formally, for each weight w_i:

$$W_i \sim \mathcal{U}[0, 1], \quad i = 1, 2, \cdots, 8 \tag{3.2}$$

$$w_i^* = \frac{w_i}{\sum_{i=1}^{8} w_i} \tag{3.3}$$

Our third goal, therefore, investigates whether our sophisticated RL agents can beat naive investors in terms of portfolio returns. The methodology we will follow to investigate this goal is similar to the ones described in the previous two subsections, with the main difference being that the agents' performances will not be compared against pseudo-market indexes and the prior portfolio, but against the naively diversified portfolios as described above. Furthermore, the comparison of agents' performance against

naive portfolios is twofold. First, we will compare deterministic backtesting performance of our agents against the equally weighted portfolio. Then, we will compare the empirical PDFs and CDFs of agents' returns from stochastic simulations against random portfolios, to assess whether our agents' returns are stochastically dominant with respect to random choices.

Rephrasing, our third goal aims at determining whether the added complexity and sophistication of our RL agents is beneficial in terms of performances when compared against naive investors' performances, and at assessing whether agents' returns are stochastically dominant with respect to randomness.

A complete and thorough discussion of the answer we provide to our third research goal can be retrieved in Section 4.3.3.

3.2.4 Fourth Goal

Our fourth goal aims at comparing the performance of our RL agents against static, buy-and-hold, portfolios. The rationale is straightforward: we want to assess wether dynamically managing a portfolio, coming with increased complexity and costs, gives an edge over passive portfolio management. Specifically, we want to make the comparison between our RL augmented BL approach and the standard Global Minimum Variance (GMV) portfolio from Markowitz (1952).

The methodology is analogous to the one described in Subsection 3.2.3, the only differences being the portfolios being compared to the agents' ones, and the fact that we are interested solely on the deterministic agents' runs.

A complete and thorough discussion of the answer we provide to our fourth research goal can be retrieved in Section 4.3.4.

3.2.5 Fifth Goal

Our fifth, and last, goal for this study involves a technical comparison between the two families of agents, the on-policy families, trained through PPO, and the off-policy families, trained instead through SAC. While for the previous goals we were interested almost exclusively on the returns of our agents' portfolios, now we aim at assigning a "best agent" award between our agents Alpha, Bravo, Charlie, and Delta.

Specifically, we will take into consideration the returns generated by the agents in deterministic backtesting, their consistence, and their reliability assessed through FSD or stochastic greatness testing. Furthermore, we are also interested in the practical viability of our results. This implies that we will acknowledge technical differences in training time and stability in our evaluation.

Hopefully, we will be able to assess which agent offers the best complexity per performance tradeoff within the bounds of our experiment.

A complete and thorough discussion of the answer we provide to our fifth research goal can be retrieved in Section 4.3.5.

3.3 The Data

So far in the presentation of our experiment we have mentioned the two groups of agents: narrow and wide agents. We also mentioned that they differ in the datasets used for their training and in the observation spaces in backtesting, namely the narrow and wide datasets. While the narrow dataset contains solely the time series for our eight considered stocks' prices, which are transformed in raw returns by the agents' environments, the wide dataset is more articulated and designed to provide wide agents with a complete overview of the market state.

The goal of this section is twofold. First, in Subsection 3.3.1 we describe thoroughly the wide dataset, explaining its organization, the features it contains, and listing our sources for the reproducibility of this study. Then, in Subsection 3.3.2 we present the training-validation-backtesting structure and its organization in different rolling windows.

3.3.1 Wide Dataset

First, it is important to assess that both the narrow and wide datasets will contain *useful* data from June 1st, 2000, to January 1st, 2026, plus almost two years of *burn-in* data prior to the beginning of the first training window, which are used solely to compute the BL prior and are discarded for training purposes. On top of the daily adjusted closing prices, already present in the narrow dataset, the wide dataset contains a variety of technical indicators and measures, alongside macroeconomic indicators.

For each one of the eight considered stocks from Table 3.3, our wide dataset will contain the daily volume[2], the maximum daily price, the minimum daily price, the trailing PE ratio, and, inspired by Vasilevich (2025), three technical indicators: SMA50, SMA200, and RSI30.

The Simple Moving Average (SMA) is a technical indicator of trending in the stock price. It is simply the average price of the stock in the n previous days. Hence, SMA50 is the average price for the stock in the 50 days preceding the current one, while SMA200 is the average price for the stock in the 200 days preceding the current one. A general formula for SMA is:

$$\mathrm{SMA}_n = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{3.4}$$

The Relative Strength Index (RSI) is a technical indicator for momentum in the stock price. A general formula for RSI is the following:

$$\mathrm{RSI}_n = 100 - \left[\frac{100}{1 + \frac{\bar{g}_n}{\bar{l}_n}}\right] \tag{3.5}$$

where $\bar{g}_n$ is the average gain over the previous period of length n, and $\bar{l}_n$ is the average loss over n. In our study we set $n = 50$.

On top of technical stock data, our wide dataset will contain macroeconomic indicators, a volatility measure, and risk-free measures. Specifically, the wide dataset will contain the monthly Consumer Price Index (U.S. Bureau of Labor Statistics (2026a)), as a measure of inflation of the American economy, and the monthly American Unemployment Rate(U.S. Bureau of Labor Statistics (2026b)), as macroeconomic indicators. As a

[2]The number of shares traded during one trading day.

volatility measure for the American market, the dataset will include the daily VIX index (Cboe Global Markets (2026)). Lastly, the dataset will include the three months T-Bill rates daily time series (Board of Governors of the Federal Reserve System (US) (2026b)), to be used as risk-free rates, and the 10Y-2Y spread (Board of Governors of the Federal Reserve System (US) (2026a)). All the stock-specific data and technical indicators were obtained through Bloomberg. Table 3.4 summarizes the content of our wide dataset.

Variable Name	**Symbol**	**Variable Type**
Variables for Each Stock		
Adjusted Closing Price	P	Technical
Daily Volume	V	Technical
Maximum Daily Price	$\max(P)$	Technical
Minimum Daily Price	$\min(P)$	Technical
Trailing PE Ratio	PE	Financial
Price to Book Ratio	PB	Financial
Simple Moving Average 50	SMA50	Trending
Simple Moving Average 200	SMA200	Trending
Relative Strength Index 30	RSI30	Momentum
Global Variables		
Consumer Price Index	CPI	Macroeconomic
Unemployment Rate	UNR	Macroeconomic
CBOE Volatility Index	VIX	Volatility
3-Month T-Bill Rate	r	Risk-Free Rate
10Y–2Y Spread	S	Risk-Free Rate

Table 3.4: Variables contained in the wide dataset.

3.3.2 Data Organization

Having spent the last section discussing the content of our wide dataset, it is now time to discuss the organization of our data, both for narrow and for wide agents, with regard to training, validation, and backtesting windows.

First, it is worth mentioning that burn-in data are included in the datasets. While training begins for all agents on June 1st, 2000, data from January 1st, 1998, are present in both the narrow and wide datasets. Burn-in data will be used solely to compute the variance-covariance matrix for the prior portfolio at the beginning of the training period. These auxiliary data will be then discarded and will not be used for training, validation, or backtesting purposes.

The training will follow the same windows for both narrow and wide agents. The training will happen on rolling windows of training and validation. First, the agent will be trained on 19 months of data, with validation happening on the next 6 months, then the window will slide 6 months forward and the training-validation process will be repeated. Training will occur on data from June 1st, 2000, to December 31st, 2023. Once the training has been completed, the agents will undergo backtesting on, yet unseen, out-of-sample data, spanning from January 1st, 2024, to December 31st, 2025. Figure 3.1 summarizes the training and backtesting process.

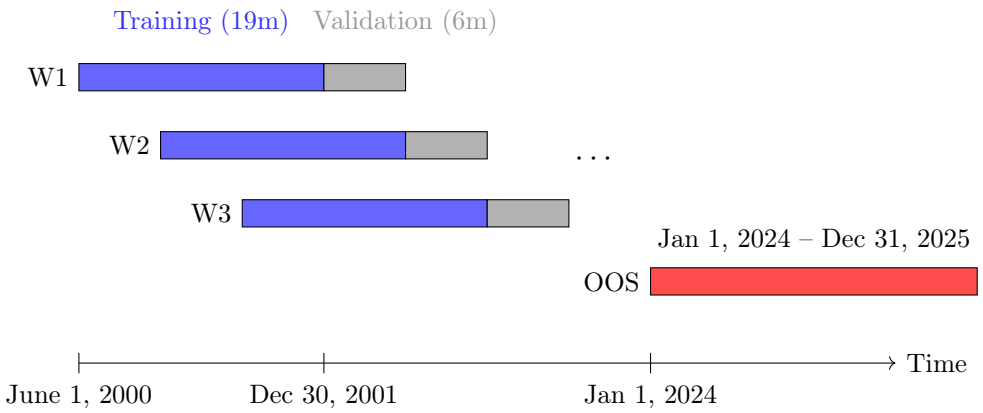


Figure 3.1: Rolling 19-month training and 6-month validation windows shifted forward by six months over the period June 1st, 2000 to December 31st, 2023. After model selection, agents are evaluated on unseen out-of-sample data from January 1st, 2024 to December 31st, 2025.

Organizing the training through rolling windows allows the agent to be periodically retrained as time moves forward. Furthermore, having an out-of-sample backtesting window ensures that agents will not suffer from look-ahead bias, testing the true out-of-sample performance of our agents with fixed parameters.

3.4 The Agents

As it could be inferred from our research goals, the cornerstone of this study is represented by our two groups of Reinforcement Learning agents: the narrow group—composed by families Alpha and Bravo—and the wide group—composed by families Charlie and Delta. Table 3.1 provided a brief overview of the main characteristic of agents' families.

Agents are organized in *families*. Families differ from each other in two dimensions: by the algorithm they use, being either PPO or SAC, and by the dataset serving as their observation space, being either the wide or the narrow dataset. Agents, to which we occasionally refer as *children* of a given family, are indexed by numbers from one to four written in letters, ordering them in ascending order of look-back window length. Agents which are children of the same family are said to be *siblings* with one another. Narrow families (Figure 3.2) have four children each. Conversely, wide families (Figure 3.3) have four couples of *twins*. Twin agents are identical between each other, hence they possess the same index, despite for the reward function they employ, being denoted either with *R1* or *R2*. Subsection 3.5.1 provides a comprehensive analysis of the two reward function we will employ for wide agents.

Subsection 3.4.1 provides a deep overview of narrow families: Alpha and Bravo. Concurrently, Subsection 3.4.2 presents the two families of wide agents: Charlie and Delta.

3.4.1 Narrow Agents

Agents Alpha and Bravo are narrow agents. The two families differ in the algorithm used for their training, the former being trained through PPO, while the latter is trained through SAC. The agents constituting Alpha and Bravo families, to which we will refer as *children agents* in this study, differ from each other in the length of their look-back windows and, consequently, for the architecture of their artificial neural networks.

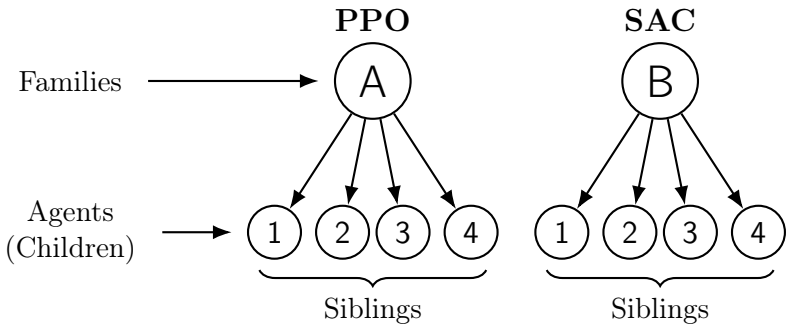


Figure 3.2: Trees showing narrow agents' hierarchical structure. Each *Family* contains four *Agents*, indexed with numbers between one and four, in ascending order of their respective look-back windows. Agents are sometimes referred as *Children* of a given family. Agents of the same family are *Siblings* with one another.

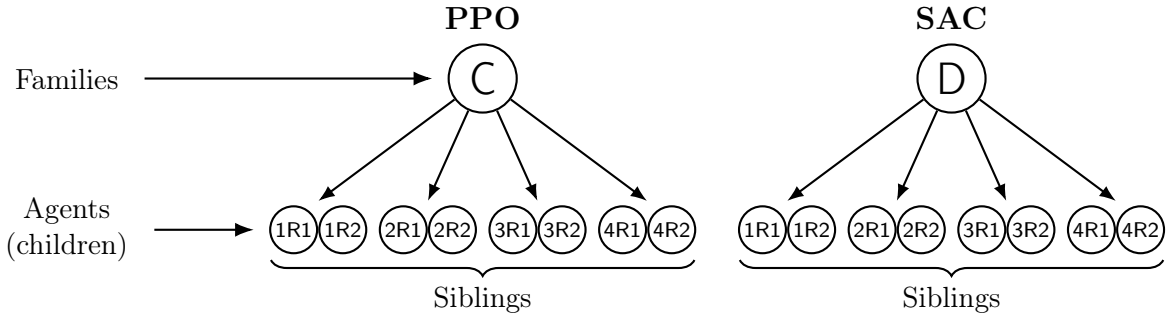


Figure 3.3: Trees showing wide agents' hierarchical structure. As in Figure 3.2, *Agents* are organized in *Families*. Wide families contain four couples of *twins* agents. Agents in each twin couple are identical in everything except for the reward function they are given, denoted by either *R1* or *R2*.

Each family has four children. Each child has a specific look-back window, i.e. the number of past trading days it considers in forming its views. If we denote with η_i the length of the look-back window for the i-th child, the four children will have $\eta_1 = 10$, $\eta_2 = 20$, $\eta_3 = 40$, and $\eta_4 = 60$.

The first agent will consider two trading weeks of adjusted closing prices, the second will consider one trading month, the third two trading months, and the fourth will consider three trading months. Since we consider eight stocks in our study, the input for each child will be a matrix of adjusted closing prices with η_i rows and 8 columns. Therefore, the number of inputs for each child will be $n = 8 \times \eta_i$.

It follows from the number of inputs, that the input layer in each ANN will be $n + 1$. While the Python library `stable-baselines3`, which we will use to implement the agents' training, automatically manages the size of the input layer, it does not do so for the number and sizes of hidden layers.

If we considered ANNs with a single hidden layer, the $2n+1$ rule of thumb coming from the Kolmogorov representation theorem (Kolmogorov (1957)) would provide us with a simple heuristic to choose the number of hidden nodes. However, it is customary practice in the literature to treat portfolio selection tasks with reinforcement learning using ANNs with more than one hidden layer. Specifically, for our experiment we settled for two-layered ANNs. Clearly, within this deep learning framework, the $2n+1$ heuristic becomes impractical. The most complex wide agents would require 9241 hidden nodes per hidden layer, which implies having to train more than 120 million weights. For comparison, the

base version of Google's BERT LLM has approximately 110 million parameters (Devlin et al. (2019)). Although training such an MLP is computationally infeasible given our computational resources and time, such a vast neural network would be highly prone to overfitting with our limited amount of data.

For our narrow agents we decided to adopt two different architectures. For shorter look-back agents (η_1, η_2) we will use two layered MLPs with 128 hidden nodes per layer, while for longer look-back agents (η_3, η_4) the first hidden layer will have 256 nodes, while the second will have 128 nodes. Using funneled MLPs as function approximators for higher-dimensional agents is likely to reduce the risk of overfitting. Furthermore, all our narrow agents are given 150 thousand time-steps per training window, with validation happening every one thousand time steps. An early stopping criterion is in place to prevent over-training the network in each single window.

Tables 3.5 offers a summary of the characteristics of narrow agents. Note that the number of weights reported in the tables (**W**) refers to the parameters of a single MLP. Because both PPO and SAC are actor-critic methods, multiple MLPs are instantiated simultaneously during training. Specifically, `stable-baselines3` creates two neural networks for PPO (an actor and a critic) and three for SAC (an actor and two twin Q-networks).

Child Name	η_i	**H**	**Input Neurons**	$\mathbf{h_1}$	$\mathbf{h_2}$	**W**
One	10	2	81	128	128	$\approx 29K$
Two	20	2	161	128	128	$\approx 39K$
Three	40	2	321	256	128	$\approx 117K$
Four	60	2	481	256	128	$\approx 158K$

Table 3.5: Network architectures for narrow agents, families Alpha and Bravo. η_i denotes the length of the look-back window. h_i denotes the number of neurons in the i-th layer. W denotes the total approximate number of weights.

Comparing agents with the same network architecture but different look-back window allows us to determine whether looking further into the past has a positive impact on performance. Furthermore, by comparing different architectures we can assess whether or not added complexity is beneficial with respect to achieving the same goal. For a detailed analysis of our results, the reader is referred to Chapter 4.

3.4.2 Wide Agents

Agents Charlie and Delta are wide agents. As before, the two families differ for the algorithm used in training, the former being trained through PPO, while the latter being trained through SAC. Again, each family has four children, which differ from one another by the length of the look-back window. The four different look-back windows are the same defined for narrow families.

Wide agents consider seventy-seven different features per trading day to output views and confidences about the eight considered stocks. Clearly, more complex neural networks, with respect to the ones used for narrow agents, are needed. Specifically, shorter look-back wide agents (η_1, η_2) will be equipped with MLPs with two hidden layers with 256 nodes per layer, while longer look-back wide agents (η_3, η_4) will be equipped with two-layered MLPs with 512 nodes in the first layer and 256 nodes in the second. Once again,

we expect that using funneled MLPs as function approximators for higher-dimensional agents is likely to reduce the risk of overfitting. As for narrow agents, also wide agents are given 150 thousand time-steps per training window, with validation happening every one thousand time steps. An early stopping criterion is in place to prevent over-training the network in each single window.

Tables 3.6 offers a summary of the characteristics of wide agents.

Child Name	η_i	**H**	**Input Neurons**	$\mathbf{h_1}$	$\mathbf{h_2}$	**W**
One	10	2	771	256	256	$\approx 267K$
Two	20	2	1541	256	256	$\approx 464K$
Three	40	2	3081	512	256	$\approx 1.71M$
Four	60	2	4621	512	256	$\approx 2.5M$

Table 3.6: Network architectures for wide agents, families Charlie and Delta. η_i denotes the length of the look-back window.h_i denotes the number of neurons in the i-th layer. W denotes the total approximate number of weights.

3.5 Technical Implementation

In the previous sections we defined our research goals, the methodology we will follow to achieve them, the data we will use, and the agents that will allow us to answer our research questions. However, no experiment can yield results if it is not properly implemented.

This section aims at providing the reader with information regarding the technical implementation of our experiment, both for context and for reproducibility. First, Subsection 3.5.1 defines the most important piece of our experimental setup: the reward function. Secondly, Subsection 3.5.2 provides the reader with notes regarding the practical implementation of the experiment.

3.5.1 The Reward Function

Arguably the most important piece of our study, the reward function is the cornerstone of every reinforcement learning algorithm. As mentioned in Subsection 2.3.2, the sole goal of any reinforcement learning agent is to maximize the total reward in the long run.

When designing a reward function, it is important to make sure that the agent gets rewarded only for its actions. In other words, our reward function shall avoid rewarding the agent for generic environment developments. To the aim of our study, this translates to the necessity of isolating the outcomes of our agents' actions form the conditions of the market. In order to do so, we propose the following reward function:

$$R_{t+1} = \alpha \left(w_*^\top r_{t+1} - w_{eq}^\top r_{t+1} - c \right) \tag{3.6}$$

where w_* is the vector of portfolio weights from Equation 2.17, which is the agent's output, w_{eq} is the vector of the prior's portfolio weights, which can be obtained form Equation 2.2, r_{t+1} is the vector of stock returns from timestep t to $t+1$, and c is a constant representing the percentage transaction cost. α serves the role of a scaling constant, and can be set equal to the amount of capital initially invested by the agent.

Interpreting our reward function is trivial. First, notice that it is solely a function of w_*, which is the agent's output. Aside from the scaling constant α, the first term,

$w_*^\top r_{t+1}$, is nothing more than the return of the view-enhanced portfolio obtained through the Black-Litterman model, with the views generated by the agent. The second term, $w_{eq}^\top r_{t+1}$, is trivially the return of the prior portfolio. Therefore, our reward function can be interpreted as rewarding the agent for the excessive return of its view enhanced portfolio, minus a penalty for the transaction costs occurred.

Since the only difference between the BL portfolio and the prior portfolio is that the first has been enhanced with the views from the agent, the reward function can be interpreted as rewarding the agent directly for its views. If the agent formed good views by correctly predicting future performances, the agent's portfolio would have outperformed the prior's due to views adding information to the prior allocation, leading to a positive reward, provided that returns were high enough to offset transaction costs. On the other hand, if the agent's views were incorrect, the prior portfolio would have outperformed the posterior's, leading to a negative reward.

Isolating the effect of views in our reward function has the beneficial effect of avoiding compensating the agent for market trends for which it was not responsible, hence assuring that the algorithm has a clear understanding of the consequences of its actions.

It is clear that the main focus of our analysis is on wide agents, which we expect to be able to reliably generate alpha. In order to deepen our analysis of families Charlie and Delta, we decided to use an alternative reward function for their training, alongside the one presented in Equation 3.6: the Sharpe ratio. The choice of using the Sharpe ratio as an alternative reward function comes from the fact that it has been widely used in the literature as a reward function for RL agents trained in portfolio optimization.

While we do not expect the Sharpe reward function to train a *bad* policy—after all, it trains the agents to maximize the reward-to-risk ratio—we expect agents trained with Sharpe reward to perform marginally worse than agents trained through our own reward function.

The rationale behind our expectation comes from the fact that the Sharpe ratio does not isolate agents' actions properly. Since agents are tasked with generate views, not portfolio weights directly, our own reward function isolates the effect of views by subtracting prior's returns and transaction costs from agents' returns. The Sharpe reward considers only the return of the agent's portfolio, transaction costs, and the standard deviation of agent's returns. When in presence of strong upward trends in the market—which is often the case—the Sharpe ratio will, almost always, be high and positive. Conversely, if agent's returns are positive due to upward market trends, our own reward function will be high and positive if and only if agent's returns are higher than prior's. We believe that isolating agents' actions in our reward function will result in better policies.

For the purposes of our analysis we will call the reward function proposed in Equation 3.6 *Isolated Returns Reward*—hereinafter IRR. For simplicity, agents trained through IRR will be denoted by a R1—which stands for Reward 1—appended at the end of their names in results tables. Conversely, agents trained with the Sharpe reward function will be denoted by a R2—Reward 2—appended at the end of their names.

3.5.2 Technical Notes

Key to ensuring the reproducibility of the study, this section aims at providing the reader with the details regarding the technical implementation of the experiment.

The programming language of our choice for the implementation of the presented experiment was Python, specifically in its version 3.11.15. Python was the software of

choice for its versatility, ease of use, open source nature, and for its natural compatibility with computational clusters' hardware. Table 3.7 offers a summary of the most important libraries we used throughout our study.

Library	**Authors**	**Purpose**
`gymnasium`	Towers et al. (2025)	Implementing environments.
`PyPortfolioOpt`	Martin (2021)	BL optimization.
`Stable-Baselines3`	Raffin et al. (2021)	Agent training and simulation.
`NumPy`	Harris et al. (2020)	Data analysis.
`pandas`	Wes McKinney (2010)	Data handling.
`PyTorch`	Paszke et al. (2019)	Training optimization.

Table 3.7: Summary of the Python libraries used to implement the experiment.

Narrow families share the same narrow environment. Similarly, wide families share the same wide environment. Both environments may differ slightly from one family to the other due to the different way PPO and SAC handle input data in training and simulation.

Both the narrow and wide environments possess the same action space. The agents are instructed to output a continuous vector of size $2 \times N$, where N is the number of stocks considered, containing the absolute views, expressed as the expected return on a long position in each stock, and the associated confidence. We must disclose that confidences are passed through a sigmoid transformation to map them in a $(0, 1)$ range, which is more suitable for the Idzorek method (Idzorek (2007)) used in our implementation of the BL model.

The main difference between the narrow and wide environments is in the agents' observation spaces. The narrow environment feds the agents only normalized returns plus the current portfolio weights for awareness. The wide environment feeds the agents our wide dataset, which has been pre-normalized. The reader can retrieve a full length discussion of agents' training in Section 4.1, and the full length validation reward tables in Section A.

In order to shorten the time taken by computations, the training of the agents and the backtesting simulations have been conducted on Ca'Foscari University's computational cluster. Both training and simulation scripts have been adapted to run multiples environments simultaneously, so to fully extract the cluster's computational power. A more detailed description of agents' training is provided in Section 4.1.

Chapter 4

Experiment and Results

While in Chapter 3 we presented the rationale, methodology, and the tools for our experiment, we devolve this chapter to its thorough discussion.

It is important to mention that the experiment *per se* is constituted exclusively by the backtesting of our agents, and the analysis which follows. However, to backtest RL agents we unavoidably have to train them. Since good training is essential to flawlessly conduct our experiment, we dedicate Section 4.1 to discuss all aspects of agents' training, both for context and for reproducibility of our study.

Backtesting is discussed in Section 4.2. First, we present the subdivision of the backtesting period in seven different windows. Then, in Subsection 4.2.1 we deal with deterministic runs, where agents exploit their optimal policy deterministically to dynamically manage their equity portfolios. Agents' performances are evaluated against buy-and-hold portfolios being, in particular, two different GMV portfolios and an equally weighted (EW) portfolio. In Subsection 4.2.2 we are concerned with stochastic simulations instead, where agents' actions are sampled from their policies' distributions. We evaluate the ECDFs of agents' returns against the ECDFs of returns from randomly weighted portfolios, to assess whether agents have actually learned viable policies or are just acting randomly—and being lucky.

In Section 4.3 we evaluate the research goals set out in Section 3.2 in light of the evidence collected through our experiment. Consequently, in Section 4.4 we express some concluding remarks about the goals we set out for this study in Section 1.1, we address future research, and we leave five open hypotheses, which seem to be empirically supported by our results, that we deem worthy of being explored. Finally, we conclude with a final, open, remark on the paradigm shift we are witnessing in finance.

4.1 Training the Agents

All our agents were trained on Ca'Foscari virtual cluster. Specifically, the node we used was equipped with dual AMD EPYC 7413 CPUs, providing a total of 40 physical cores. The system architecture utilized the Zen 3 micro-architecture with a combined 640 MiB of L3 cache, optimized for high-throughput computational workloads. The environment was hosted on a KVM-based virtualization layer with support for AVX2 vector instructions. In addition, the node was equipped with 393 GB of RAM.

We must disclose that this entire study would have been nearly impossible to conduct without having access to the aforementioned virtual cluster. The most complex agents, like DeltaThree and Four, were trained by running up to twenty environments simultaneously and dedicating multiple cores solely to backpropagation tasks. These training instances not only drained computational power, but also massive amounts of RAM. Still, high-complexity agents took up to three days each to train.

Full tables of hyperparameter specifications per each agent can be found in Section A.1. It should be noted that hyperparameter tuning was conducted through a *trial and error* procedure. The definitive hyperparameters presented in the aforementioned section were the ones deemed capable to provide our agents with stable and efficient learning.

During training, early stopping mechanisms were in place to avoid over-training our agents. Early stopping mechanisms prescribed, generally, a minimum number of evaluations per window, generally between four and six, and a maximum number of consecutive evaluations in which improvements were not detected, usually four or five. The maximum number of time-steps per window varied per each family of agents, due to the difference in the handling of episodes between PPO and SAC. However, due to the aforementioned early stopping mechanisms, very rarely agents drained all the time-steps at their disposal in each window.

Section A.2 reports complete tables of validation rewards, alongside the average validation reward, per each agent in each training window. One interesting insight we can gain from those tables, is that PPO trained agents gained, on average, lower validation rewards compared to their SAC counterparts. As we will notice in the following sections, this difference in validation behavior between PPO and SAC families is hinting at superior out-of-sample backtesting performance from SAC trained agents.

One last observation regarding agents training, which could affect real-world applicability of our results, regards the training time for the two different algorithms. Clearly, wide agents took significantly more to train with respect to narrow agents. However, we shall report that PPO agents were significantly faster to converge to an optimal policy than SAC trained agents. As mentioned before, complex SAC wide agents took up to three days to train. Conversely, their PPO counterparts—CharlieThree and Four—took significantly less time, converging in approximately ten to twelve hours. When transitioning from a backtesting perspective to a real world live implementation of our study, one must consider this massive difference in training time. We must also mention that no considerable difference in training time was observed between R1 and R2 wide agents.

4.2 Backtesting

Backtesting was conducted on an out of sample window spanning two years from January 1st, 2024 to December 31st, 2025[1]. The full backtesting window was further divided in several sub windows, in order to test the performance of our agents on both the short and medium period. Table 4.1 offers a summary of all the considered backtesting windows.

Window	**Start**	**End**
H1 2024	01-01-2024	30-06-2024
H2 2024	01-07-2024	31-12-2024
H1 2025	01-01-2025	30-06-2025
H2 2025	01-07-2025	31-12-2025
Y 2024	01-01-2024	31-12-2024
Y 2025	01-01-2025	31-12-2025
2Y	01-01-2024	31-12-2025

Table 4.1: Backtesting windows for all agents.

In Section 4.2.1 we present deterministic backtesting runs, providing the reader with some key results. In Section 4.2.2 we present the stochastic simulations we ran for our agents, their random baselines, and some key insights we gained.

4.2.1 Deterministic Runs

When our agents are tasked to conduct deterministic backtesting runs, they choose the single best action accordingly with the policy they have learned. In practice, our agents typically handle their actions—outputting views and associated confidence—by outputting a normal distribution (clipped), for PPO agents, or a squashed normal (via a hyperbolic tangent function), for SAC agents. When asked to behave deterministically, the agents will output the mean of the aforementioned distributions.

In our deterministic backtesting, all agents are tasked to manage an initial capital of 10000 across each single window defined in Table 4.1. Their performance in each window is compared with the ones from four selected benchmarks: two Global Minimum Variance (GMV) portfolios, an equally weighted portfolio, and the S&P500 index.

The two GMV portfolios differ between each other for the look-back of the covariance matrix used to compute the weights at the beginning of each window, one being computed on six trading months of returns and the other on one trading year. The equally weighted portfolio is defined as in Equation 3.1, assigning the same weight of 0.125 to each stock in the portfolio.

We must pay close attention to the interpretation of our agents' performance against the S&P500 index. While the previously discussed benchmarks include just the same eight stocks we consider, SPY's universe is wider, including also some heavy losers. We include SPY in our study solely because it is widely recognized as a proxy for a *market portfolio*, but we acknowledge that the simple comparison between S&P500's and agents' performances is little indicative. However, any diversified portfolio able to beat the market consistently, even if it is not the best comparison, can be considered remarkable.

[1]Recall that agents' training took place through rolling windows between June 1st, 2000, and December 31st, 2023.

In our analysis, we should also remark that none of our benchmarks is affected by transaction costs. However, to our agents we apply a transaction cost of 10 basis points of the total turnover of the portfolio for each rebalancing. Transaction costs are also considered for stochastic simulations, which are discussed in the next section.

The rationale behind conducting deterministic runs to backtest our agents is given by the need to assess the overall performance of the best strategy they have learned during training. Deterministic runs are especially useful with regards to goals 1, 2, and 4 of our study (Table 3.2). Furthermore, deterministic runs give us an indication of the behavior of the agent if put into a live trading environment, when instead of evaluating its performance on a past closed window the agent is tasked to manage a portfolio today, with results being observable only in the future.

What follows are the main results and some key insights on the performance of deterministic runs of each family of agents. A full detailed discussion of the performance per each sub agent, alongside all of their plots can be found in Section B.

Alpha Family

Among Alpha family, only AlphaTwo got close to obtaining satisfactory performances across the majority of the considered backtesting windows, especially with regards to the full two-years run. However, despite its best efforts, AlphaTwo could not outperform its benchmarks reliably.

The reader, by observing Table 4.2, may notice that AlphaThree's deterministic policy was, in multiple instances, superior to AlphaTwo's in generating returns. However, AlphaThree was unable to sustain this performance into 2025, ultimately trailing AlphaTwo in the 2Y total.

On the other hand, the performances of AlphaOne's and AlphaFour's deterministic strategies were generally unsatisfactory with respect to the proposed benchmarks, especially when considering the full two-years run.

When evaluating these results, it is crucial to account for the impact of transaction costs. The agents instead were affected by a transaction cost computed as ten basis points of the value of the portfolio turnover between rebalancings, while the benchmarks were calculated gross of costs. Engle et al. (2006) estimated for stocks traded in the NYSE an average transaction cost of 8.82 bps, and an average transaction cost for NASDAQ traded stocks of 13.84 bps. However, those estimates are from April 2006. We believe that the massive leap forward in technology the Financial Industry witnessed from 2006 onward has contributed to lowering transaction costs. Furthermore, we deal with highly liquid and large capitalization stocks. In light of those considerations, we deem realistic a transaction cost of 10 bps on portfolio turnover.

However, acknowledging the burden of transaction costs our agents have to carry does not make Alpha's results satisfactory, since the deterministic policies of all agents generally fell behind our considered benchmarks. Nonetheless, the performances obtained by Alpha's deterministic policies seem to be aligned with the weak form of the EMH. Recall that each agent contributes to portfolio selection by generating absolute views—and their associated confidence—about the future performance of each stock. Views and confidence are then used as input for the Black-Litterman model, which completes the asset allocation. Since agents from the Alpha family have at their disposal only past returns to generate views, it is likely that their observation space does not contain much signal, which our Alpha agents were not able to extract properly, resulting in weakly

confident and potentially erroneous views, which in turns affected negatively the final asset allocation.

All equity line plots for agents of the Alpha family and the benchmarks, including the *improper benchmark* SPY, referred to each backtesting window, can be retrieved in Section B.1.

Strategy/Agent	H1 2024	H2 2024	H1 2025	H2 2025	Y 2024	Y 2025	2Y
EW	0.1141	0.1180	0.0727	0.1759	0.2472	0.2471	0.5554
GMV-6M	0.1287	0.1104	0.0616	0.1800	0.2660	0.2299	0.5668
GMV-1Y	0.1229	0.1115	0.0745	0.1712	0.2504	0.2574	0.5633
AlphaOne	0.0920	0.1258	0.0159	0.0475	0.2213	0.1821	0.1028
AlphaTwo	0.1210	0.1194	-0.0237	0.2913	0.2122	0.2378	0.5065
AlphaThree	0.2261	0.1703	0.0081	0.1523	0.3044	0.2499	0.3661
AlphaFour	0.0511	0.1880	-0.0535	0.0522	0.1723	0.0673	0.2114

Table 4.2: Deterministic returns by window - Alpha family vs benchmarks.

Bravo Family

Among Bravo family performances of deterministic strategies varied massively. As one may observe from Table 4.3, there was not a single agent able to outperform reliably its benchmarks across the majority of backtesting windows. Arguably, the weakest child in the family was BravoOne, which was capable of outperforming its benchmarks in one single occasion, H1 2025.

BravoThree was able to outperform its benchmarks in a couple of occasions—H1 2024 and in the full year 2024—but struggled in all other windows. Across the full two-years run, BravoThree concluded with a mere 1.53% return.

BravoTwo and Four offer more interesting cases. Both agents' deterministic policies often lagged behind benchmarks, beating them in just a few occasions. Nevertheless, both agents performed remarkably on the full two-years run, obtaining returns of 78% and 86% respectively. Despite those incredible full run returns, we should stay cautious when generalizing the performances from BravoTwo and Four. Our two Bravo *golden children* are far from consistently outperforming the considered benchmarks, and it is likely that the extreme performances obtained over two years of backtesting are more strokes of luck than a consistent an functional policy.

Overall, the performances obtained by Bravo's deterministic strategies can be deemed in line with Alpha's. Additionally, the performance inconsistencies plaguing Bravo family seem to be aligned with weak form market efficiency. Once again, we can speculate that the negligible amount of signal contained in past returns, which constitute the entire observation space for Bravo agents, leads to low-confidence or outright wrong views, which negatively contribute to asset allocation through the BL model. The only difference with Alpha family being that, occasionally, SAC trained agents might exploit some signal profitably, leading to massive returns.

All equity line plots for agents of Bravo family and the benchmarks referred to each backtesting window, can be retrieved in Section B.2.

Strategy/Agent	H1 2024	H2 2024	H1 2025	H2 2025	Y 2024	Y 2025	2Y
EW	0.1141	0.1180	0.0727	0.1759	0.2472	0.2471	0.5554
GMV-6M	0.1287	0.1104	0.0616	0.1800	0.2660	0.2299	0.5668
GMV-1Y	0.1229	0.1115	0.0745	0.1712	0.2504	0.2574	0.5633
BravoOne	0.0251	0.0434	0.0911	-0.0982	0.0952	0.1897	0.0338
BravoTwo	0.0987	0.1524	-0.0150	0.1812	0.2202	0.0931	0.7808
BravoThree	0.1481	0.1165	0.0160	0.0202	0.2723	0.0737	0.0153
BravoFour	0.2069	-0.0081	-0.0013	0.0258	0.2613	-0.0797	0.8612

Table 4.3: Deterministic returns by window - Bravo family vs benchmarks.

Charlie Family

When observing the returns achieved in deterministic runs by Charlie agents, one may notice that agents trained with IRR reward function—Equation 3.6—performed generally better than agents trained with the Sharpe reward. The only exception is CharlieThree, where the R2 twin seemed to have outperformed R1 by a slim margin. The most noticeable improvement attributable by the change in reward function concerns CharlieFour, its returns being roughly 15% larger for R1 than R2 in the full two-years run. From now onward our analysis focus almost entirely on the R1 agents.

Among the Charlie family, the best performing agent in deterministic runs was CharlieOne. One can appreciate, by observing Table 4.4, that CharlieOne deterministic policy was able to generate returns outperforming all of its benchmarks in five out of seven backtesting windows. Two exceptions stand out: in the first half of 2025, where all agents had negative returns, and in the full year 2025 CharlieOne's performance fell short only to the one-year look-back GMV portfolio.

Surprisingly, the second best performer among the Charlie family was CharlieFour, which was the worst performer within R2 agents. CharlieFour performed extremely well, sometimes outperforming even CharlieOne, and outperforming its benchmarks in multiple occasions.

CharlieTwo's performances were solid but inferior to CharlieOne's and Four's. In most occasions CharlieTwo performed in line with its benchmarks. While it might be tempting to disregard CharlieTwo's performances as unremarkable, it is important to remember that agents carry the burden of transaction costs, while benchmarks do not. For this reason, CharlieTwo's performances can be deemed remarkable.

CharlieThree's performances tell a different story. While it was able to generate positive—and large—returns, its returns most often fell behind benchmarks'. Interestingly, CharlieThree performed poorly both when trained with R1 and with R2 reward functions.

Notice that all Charlie agents, either trained under R1 or under R2, performed poorly on the same window: H1 2025. It is likely that at the beginning of this particular window a market regime shift happened. The heuristics agents used before, which where adapted to the previous market regime suddenly stopped working, hence agents performed poorly. In H2 2025, agents were, generally, able to modify their heuristics, quickly recovering from their losses. This observation is aligned with the Adaptive Market Hypothesis—hereinafter AMH—proposed in Lo (2004). We will discuss thoroughly this aspect of agents' behaviors in Section 4.2.2.

All equity line plots for agents of the Charlie family and the benchmarks, including

the *improper benchmark* SPY, referred to each backtesting window, can be retrieved in Section B.3.

Strategy/Agent	H1 2024	H2 2024	H1 2025	H2 2025	Y 2024	Y 2025	2Y
EW	0.1141	0.1180	0.0727	0.1759	0.2472	0.2471	0.5554
GMV-6M	0.1287	0.1104	0.0616	0.1800	0.2660	0.2299	0.5668
GMV-1Y	0.1229	0.1115	0.0745	0.1712	0.2504	0.2574	0.5633
CharlieOneR1	0.1545	0.1405	-0.0141	0.2693	0.3168	0.2494	0.6029
CharlieTwoR1	0.1435	0.1080	-0.0269	0.2704	0.2654	0.2302	0.5327
CharlieThreeR1	0.1104	0.1164	-0.0456	0.2467	0.2370	0.1879	0.4413
CharlieFourR1	0.1708	0.1386	-0.0357	0.2567	0.3285	0.2114	0.5764
CharlieOneR2	0.1648	0.1284	-0.0131	0.2565	0.3102	0.2423	0.5964
CharlieTwoR2	0.1277	0.1087	-0.0155	0.2611	0.2469	0.2389	0.5306
CharlieThreeR2	0.1417	0.0939	-0.0349	0.2260	0.2431	0.1888	0.4564
CharlieFourR2	0.1011	0.1180	-0.0426	0.2469	0.2243	0.1924	0.4274

Table 4.4: Deterministic returns by window - Charlie family vs benchmarks.

Delta Family

Differently from PPO trained agents from the Charlie family, Delta agents performed generally better when trained with the Sharpe reward function (R2). The only exception is constituted by DeltaOne which, when trained with IRR reward (R1), outperformed considerably its R2 twin, and registered the second best overall performance within the Delta family.

The observed reward function reversal is likely due to the off-policy nature of SAC and its defining entropy maximization framework. First, because SAC samples past experiences from a replay buffer, the highly specific, isolated nature of IRR might lose its contextual validity when evaluated later, likely due to shifts in market regime over time. The Sharpe ratio, which evaluates broader portfolio-level risk-adjusted returns, is arguably more temporally robust and provides a smoother learning signal when drawing from mixed historical data. Furthermore, SAC explicitly optimizes for a trade-off between maximizing the expected return and maximizing the policy's entropy. A generalized, holistic reward function like the Sharpe ratio naturally synergizes with this objective, giving the agent a wider mathematical landscape to explore than the highly constrained, isolated IRR.

Table 4.5 contains complete returns for our agents and their benchmarks over all considered backtesting window. As mentioned, R1 agents had overall worse performances than their R2 counterparts, the sole exception being DeltaOne, which, under R1, was able to outperform its benchmarks reliably in the majority of backtesting windows. Within the entire Delta family the best performing agent was DeltaThreeR2, which outperformed its benchmarks in six out of seven backtesting windows and, most notably, was the only among its siblings to return positively in the first half of 2025. Furthermore, DeltaThreeR2 generated a stunning 65.99% return over the full two-years run.

Observing the IRR trained agents (R1), we may notice that DeltaTwo and Three outperformed their benchmarks in all 2024 backtesting windows and in the second half of

2025. However, both suffered heavily from the regime shift within the first half of 2025, which led them to underperform their benchmarks in the full two-years run. Conversely, DeltaFour had difficulties in recovering from the regime shift, leading to poor returns in the second half of 2025, despite showing solid performances in 2024.

Among the Sharpe trained agents (R2), DeltaFour had worse than benchmark performance in 2024, but resisted very well the regime shift, which allowed it to outperform its benchmarks in the full two-years run. DeltaTwo had outstanding performances in 2024, but suffered more than DeltaFour in 2025 as a result of the regime shift, resulting in its full run returns falling short from its benchmarks. DeltaOne's performances were the weakest among the R2 trained agents, its returns rarely outperforming the benchmarks.

Overall, the performances obtained by Delta agents can be regarded as highly satisfactory, with three agents reliably outperforming the proposed static benchmarks, three agents showing mild performances—mainly harmed by the market regime shift—and only two systematic under-performers. When making the comparison with the Charlie family, we can assess that, on average, Delta agents performed better than Charlie agents when in their best configuration. Furthermore, it seems that for Delta agents a lower agent complexity was beneficial to returns only when trained using IRR, while Sharpe reward tended to benefit larger, more complex agents.

In conclusion, it is likely that, when using the optimal reward function, SAC trained agents learned a more generalized optimal policy compared to the one learned by PPO trained agents. This, in turns, led Delta agents to extract more information from their observation space compared to Charlie agents, leading to better views and, consequently, higher returns. We leave further arguments to this regard to Section 4.2.2.

All equity line plots for agents of the Delta family and the benchmarks referred to each backtesting window, can be retrieved in Section B.4.

Strategy/Agent	H1 2024	H2 2024	H1 2025	H2 2025	Y 2024	Y 2025	2Y
EW	0.1141	0.1180	0.0727	0.1759	0.2472	0.2471	0.5554
GMV-6M	0.1287	0.1104	0.0616	0.1800	0.2660	0.2299	0.5668
GMV-1Y	0.1229	0.1115	0.0745	0.1712	0.2504	0.2574	0.5633
DeltaOneR1	0.1546	0.1155	-0.0070	0.2599	0.2958	0.2522	0.5860
DeltaTwoR1	0.1424	0.1129	-0.0400	0.2576	0.2641	0.2125	0.5110
DeltaThreeR1	0.1465	0.1182	-0.0506	0.2755	0.2786	0.2073	0.5195
DeltaFourR1	0.1336	0.1155	-0.0168	0.2095	0.2636	0.1758	0.4663
DeltaOneR2	0.1423	0.1086	-0.0385	0.2807	0.2638	0.2304	0.5007
DeltaTwoR2	0.1762	0.1218	-0.0468	0.2361	0.3228	0.1674	0.5507
DeltaThreeR2	0.1809	0.1168	0.0243	0.2444	0.3223	0.2651	0.6599
DeltaFourR2	0.1464	0.0870	-0.0003	0.2515	0.2494	0.2583	0.5743

Table 4.5: Deterministic returns by window - Delta family vs benchmarks.

4.2.2 Stochastic Simulations

As mentioned in the previous section, agents handle their actions by outputting a specific distribution. For a given state of the environment, each agent will always output the same distribution for each action. When conducting deterministic runs, actions are chosen as

the mean of the output distributions. When tasked with acting stochastically, agents choose their actions by sampling from those distributions. Therefore, each single run will deviate from the others based on the standard deviation of the learned distributions.

Differently from deterministic backtesting, now the agents are not concerned with managing an initial capital of 10000 units. Rather, we are interested solely on the total cumulative return obtained by our agents at the end of each backtesting window, defined as in Table 4.1. As for deterministic runs, agents are charged a transaction cost of ten basis points of the turnover of their portfolio per each rebalancing.

When conducting stochastic simulations, we simulate 2500 runs per each agent in each backtesting window. The empirical distributions of agents' returns stemming from the simulations will be tested for stochastic dominance against the empirical distributions of returns from 2500 randomly weighted portfolios in the same windows, as defined in Section 3.2.3.

We want to test for First-Order Stochastic Dominance (FSD) of our agents' returns over random returns. Formally, random variable A has first-order stochastic dominance over random variable B if for any outcome x, A gives at least as high a probability of receiving at least x as does B, and for some x, A gives a higher probability of receiving at least x. In notation form:

$$P[A \geq x] \geq P[B \geq x], \quad \forall x \in \mathcal{X},$$

where $\mathcal{X}$ denotes the common support of A and B. In terms of the cumulative distribution functions, $A \succeq_1 B$ means that $F_A(x) \leq F_B(x)$ for all $x \in \mathcal{X}$.

We evaluate dominance using a two-pronged statistical approach. First, we utilize the one-sided Kolmogorov-Smirnov Test, proposed in Smirnov et al. (1939), to determine if the agent's CDF is significantly shifted to the right of the random baseline (testing the FSD condition directly). Second, we apply the Mann-Whitney U Test (Mann and Whitney (1947)) to assess whether the agent's returns are systematically higher than the baseline, providing a robust measure of stochastic superiority.

If we indicate with A the random variable generating agents' returns, and with R random variable generating random returns, then the null and alternative hypotheses for the Kolmogorov-Smirnov test are:

$$H_0 : F_A(x) \leq F_R(x) \ \forall x \in \mathcal{X}$$
$$H_1 : \exists x \in \mathcal{X} : F_A(x) > F_R(x)$$

In plain English, under the null hypothesis (H_0) the agent's returns exhibit First-Order Stochastic Dominance (FSD) over the random baseline. Conversely, the alternative H_1, asserts that the agent does not achieve FSD. This implies there is at least one threshold of return that can be exceeded with higher probability by the random portfolios rather than by the agent's portfolios.

The null and alternative hypotheses for the Mann-Whitney U test are instead defined as:

$$H_0 : P(A > R) \leq 0.5$$
$$H_1 : P(A > R) > 0.5$$

Under the MW null hypothesis, we are stating that the distribution underlying the agent's returns is equal to or stochastically less than the random portfolio's returns. Therefore,

under H_0, the chance that a randomly drawn return from the agent beats a randomly drawn return form the baseline is no greater than 0.5. However, we must be careful in evaluating the alternative hypothesis. In particular, if we fail to reject the null from the KS test, this implies that the alternative from the MW test is true, but the vice-versa does not hold. Therefore, the KS test is needed to assess whether agent's returns achieve stochastic dominance over the random baseline, and the MW serves the function of verifying that agent's returns are also stochastically greater.

Throughout our analysis we shall be aware that FSD is a significantly more strict condition than stochastic greatness. In fact, FSD disappears the moment F_A crosses above F_R, even at a single point. Stochastic greatness is significantly easier to achieve. Agents testing positive for stochastic greatness over the baseline implies that, over the same backtesting window, agents will obtain returns higher than the random baseline with a probability higher than 0.5. Agents testing positive for FSD over the baseline implies that for any return threshold, the agent provides a probability of success that is equal to or greater than the random baseline. Economically, this suggests that any rational investor with a non-decreasing utility function would prefer the agent's return distribution over the random baseline. We will deem agents' performance satisfactory if we will observe consistency in positive assessments of stochastic greatness, occasionally validated by FSD. The reason for which we seem to care more about the outcome of the MW test, rather than one of the KS test, is that there is, mostly for wide agents, a compelling explanation for CDF crossings. While the random baseline is characterized by high entropy and variance, the RL agents have converged toward a disciplined policy. Consequently, the agent's return distributions are significantly tighter (lower variance) than the random portfolios. This implies that random portfolios are more likely to obtain not only lower returns than the agents, but also extremely higher returns. This difference in the tails of the distributions leads to inevitable CDF crossings, which invalidate FSD even when the agent demonstrates clear superiority in terms of average win-probability and risk-adjusted returns.

The following subsections present the key results of stochastic runs and stochastic dominance testing for each family of agents. Full tables of results and the plots of estimated PDFs and empirical CDFs for each agent against the random baselines can be retrieved in Section C.

Alpha Family

As discussed in Section 4.2.1, deterministic runs from the Alpha family did not yield any satisfactory results. Performing stochastic simulations for agents of the Alpha family over the same backtesting windows, and testing their returns against random portfolios for stochastic dominance—through the Kolmogorov-Smirnov test—and stochastic greatness—through the Mann-Whitney U test—can help us shed light on said poor performances.

Tables 4.6 and 4.7 contain the results of the KS and MW tests for all Alpha agents, across all backtesting windows. In Section C.1 the reader can instead retrieve the empirical distribution plots for all Alpha agents. A caveat regarding terminology: throughout this analysis we may want to refer to the empirical cumulative distribution functions of returns simply as *returns* for brevity. Therefore, the reader is reminded that any statement about *stochastic dominance* or *stochastic greatness* of *returns* refers to the cumulative distribution functions (CDFs) of the random variables from which the returns are drawn.

As we can observe from the aforementioned tables, agents' returns never tested as stochastically dominant over random returns. Furthermore, the same almost never tested as stochastically greater than the same random baseline, the only exception being the H1 2024 window. In general, Alpha's performances were highly unsatisfactory, validating what we observed in deterministic backtesting.

The results obtained with respect to the Alpha family align with weak form market efficiency. While they might have learned simple heuristics that allow them to form *some* views—after all their returns are almost always positive and seem to follow market trends— but their observation space, composed only of past returns, does not allow them to form *good* views, which would instead bring excess returns by efficiently generating a better posterior portfolio through the BL model, in any reliable way. Occasionally, agents from the Alpha family are able to generate returns which are stochastically greater than random returns, however, due to the limited amount of information contained in their observation space they are unable to repeat good performances reliably.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
AlphaOne	H1 2024	0.124	0.107	0.203	7.330e-46	F	3.508e+06	3.067e-14	T
AlphaTwo	H1 2024	0.135	0.107	0.145	1.444e-23	F	3.787e+06	9.766e-39	T
AlphaThree	H1 2024	0.151	0.107	0.124	2.366e-17	F	4.115e+06	4.601e-84	T
AlphaFour	H1 2024	0.124	0.107	0.196	7.161e-43	F	3.652e+06	2.614e-25	T
AlphaOne	H2 2024	0.091	0.116	0.377	1.024e-158	F	2.208e+06	1.000	F
AlphaTwo	H2 2024	0.109	0.116	0.244	5.197e-66	F	2.864e+06	1.000	F
AlphaThree	H2 2024	0.111	0.116	0.278	5.541e-86	F	3.061e+06	0.894	F
AlphaFour	H2 2024	0.099	0.116	0.330	1.758e-121	F	2.525e+06	1.000	F
AlphaOne	H1 2025	-4.680e-04	0.075	0.665	0.000	F	1.152e+06	1.000	F
AlphaTwo	H1 2025	-0.018	0.075	0.740	0.000	F	7.601e+05	1.000	F
AlphaThree	H1 2025	0.013	0.075	0.586	0.000	F	1.487e+06	1.000	F
AlphaFour	H1 2025	8.668e-04	0.075	0.657	0.000	F	1.190e+06	1.000	F
AlphaOne	H2 2025	0.130	0.168	0.376	1.099e-157	F	2.412e+06	1.000	F
AlphaTwo	H2 2025	0.148	0.168	0.314	1.383e-109	F	2.650e+06	1.000	F
AlphaThree	H2 2025	0.130	0.168	0.400	9.880e-180	F	2.270e+06	1.000	F
AlphaFour	H2 2025	0.107	0.168	0.460	1.000e-239	F	1.871e+06	1.000	F

Table 4.6: Dominance Results - Agents form the Alpha Family, half yearly runs

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
AlphaOne	2024	0.238	0.243	0.204	3.210e-46	F	2.904e+06	1.000	F
AlphaTwo	2024	0.239	0.243	0.208	3.222e-48	F	3.122e+06	0.527	F
AlphaThree	2024	0.239	0.243	0.232	1.090e-59	F	3.031e+06	0.967	F
AlphaFour	2024	0.227	0.243	0.239	3.132e-63	F	2.827e+06	1.000	F
AlphaOne	2025	0.117	0.252	0.613	0.000	F	1.273e+06	1.000	F
AlphaTwo	2025	0.091	0.252	0.672	0.000	F	8.259e+05	1.000	F
AlphaThree	2025	0.102	0.252	0.668	0.000	F	1.015e+06	1.000	F
AlphaFour	2025	0.079	0.252	0.733	0.000	F	7.900e+05	1.000	F
AlphaOne	Full	0.290	0.550	0.677	0.000	F	8.854e+05	1.000	F
AlphaTwo	Full	0.321	0.550	0.653	0.000	F	1.029e+06	1.000	F
AlphaThree	Full	0.247	0.550	0.738	0.000	F	6.742e+05	1.000	F
AlphaFour	Full	0.298	0.550	0.680	0.000	F	9.534e+05	1.000	F

Table 4.7: Dominance Results - Agents form the Alpha Family, Y 2024, Y 2025, and 2Y.

Bravo Family

As discussed in Section 4.2.1, the performances of Bravo agents were characterized by strong heterogeneity among agents. Stochastic simulations, then used for testing stochas-

tic dominance and greatness against the usual random baseline, confirm the observed heterogeneity in performance. To the extent of this discussion, Tables 4.8 and 4.9 contain the results of the KS and MW tests for all Alpha agents, across all backtesting windows. In Section C.2 the reader can instead retrieve the empirical distribution plots for all Alpha agents.

With respect to Alpha agents, one Bravo agent, BravoTwo, tested positive for FSD in one backtesting window, H2 2024. While BravoOne and Three had comparable performance to Alpha agents, BravoTwo and Four performed slightly better, especially in the full two-years run, where the returns from both tested stochastically greater than the baseline.

While claiming that BravoTwo and Four are able to form good views due to their superior policies may be tempting, we should acknowledge that the occasional stochastic greatness of their returns is more a sign of stable learning, rather than of high predictive ability. Despite the apparent superiority to Alpha agents' results, Bravo's are also aligned with weak market efficiency. We can claim that Bravo agents too have learned simple heuristics allowing them to form some views, which often provide weaker results than Alpha's due to the exploring nature of weakly converged SAC models. However, due to the more stable nature of SAC, occasionally those weak heuristics are able to extract higher quality information from noisy observation spaces, especially in presence of evident patterns, such as the upward market trend present in the full two-years run. Those occasional instances of predictive lucidity from some Bravo agents allow them to form *good* views which can positively improve the prior through the BL model. Nevertheless, due to the scarce amount of information contained in Bravo's observation space, more often than not agents cannot form consistent views.

Despite showing some occasional good performance, Bravo agents ultimately succumb to weakly efficient markets, making them unreliable in forming views. This unreliability, combined with the exploratory nature of SAC, ultimately affects negatively Bravo's returns, making them often stochastically inferior and lower than random portfolio's. In general, we can conclude that the performances obtained by both families of narrow agents in stochastic simulations is unsatisfactory.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
BravoOne	H1 2024	0.033	0.107	0.684	0.000	F	7.777e+05	1.000	F
BravoTwo	H1 2024	0.077	0.107	0.439	1.985e-217	F	1.674e+06	1.000	F
BravoThree	H1 2024	0.120	0.107	0.035	0.048	F	3.788e+06	7.023e-39	T
BravoFour	H1 2024	0.194	0.107	0.074	1.302e-06	F	5.092e+06	0.000	T
BravoOne	H2 2024	0.118	0.116	0.176	2.257e-34	F	3.290e+06	6.160e-04	T
BravoTwo	H2 2024	0.157	0.116	0.000	1.000	T	5.182e+06	0.000	T
BravoThree	H2 2024	0.097	0.116	0.255	5.521e-72	F	2.140e+06	1.000	F
BravoFour	H2 2024	0.023	0.116	0.771	0.000	F	4.295e+05	1.000	F
BravoOne	H1 2025	0.005	0.075	0.653	0.000	F	9.202e+05	1.000	F
BravoTwo	H1 2025	0.008	0.075	0.664	0.000	F	9.092e+05	1.000	F
BravoThree	H1 2025	0.027	0.075	0.522	1.161e-311	F	1.787e+06	1.000	F
BravoFour	H1 2025	-0.010	0.075	0.830	0.000	F	3.379e+05	1.000	F
BravoOne	H2 2025	-0.039	0.168	0.943	0.000	F	3.208e+04	1.000	F
BravoTwo	H2 2025	0.163	0.168	0.212	1.065e-49	F	2.825e+06	1.000	F
BravoThree	H2 2025	0.056	0.168	0.723	0.000	F	6.078e+05	1.000	F
BravoFour	H2 2025	0.094	0.168	0.508	7.257e-294	F	1.647e+06	1.000	F

Table 4.8: Dominance Results - Agents form the Bravo Family, half yearly runs.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
BravoOne	2024	0.143	0.243	0.556	0.000	F	1.119e+06	1.000	F
BravoTwo	2024	0.206	0.243	0.319	5.328e-113	F	2.310e+06	1.000	F
BravoThree	2024	0.205	0.243	0.282	3.115e-88	F	2.080e+06	1.000	F
BravoFour	2024	0.241	0.243	0.207	1.142e-47	F	3.044e+06	0.945	F
BravoOne	2025	0.005	0.252	0.867	0.000	F	1.947e+05	1.000	F
BravoTwo	2025	0.210	0.252	0.387	8.406e-168	F	2.394e+06	1.000	F
BravoThree	2025	0.127	0.252	0.670	0.000	F	9.927e+05	1.000	F
BravoFour	2025	0.026	0.252	0.804	0.000	F	4.692e+05	1.000	F
BravoOne	Full	0.060	0.550	0.926	0.000	F	5.081e+04	1.000	F
BravoTwo	Full	0.634	0.550	0.162	2.431e-29	F	3.871e+06	1.041e-48	T
BravoThree	Full	0.032	0.550	0.988	0.000	F	1068.000	1.000	F
BravoFour	Full	0.566	0.550	0.202	2.514e-45	F	3.219e+06	0.032	T

Table 4.9: Dominance Results - Agents form the Bravo Family, Y 2024, Y 2025, and 2Y.

Charlie Family

Among the Charlie family, the best performing agent in stochastic simulations, both when trained under R1 and R2, was CharlieOne—the smallest agent in the pack, with a look-back window of just ten trading days and a two-layered ANN with 256 neuron per layer. CharlieOne's returns proved to be stochastically greater than the random baseline in five out of seven windows—the exception being H1 2025 and the full 2025 yearly run, where its total returns were brought down by the first under-performing half of the year. Furthermore, CharlieOne's returns achieved FSD over the random baseline in two occasions, H1 2024 and H2 2025, alongside its siblings.

CharlieTwo, both under R1 and under R2, performed worse than CharlieOne, its returns being stochastically greater than the random baseline in three out of seven occasions achieving stochastic dominance in two of them, with the R1 twin performing marginally better in terms of average return than R2. CharlieThree performed worse than CharlieTwo, although this time the R2 twin performed fractionally better than R1. Furthermore, among the R1 agents, CharlieThree was the worst performing sibling.

CharlieFour offers an interesting insight among the Charlie family. The most complex agent in the pack is also the worst performing when trained using Sharpe reward, while simultaneously being the second best when trained using our IRR reward function. When trained under IRR, CharlieFour's returns are stochastically greater than the baseline in five out of seven windows, achieving FSD in three of them. However, despite attaining FSD one time more than CharlieOne, the estimated PDFs of its returns often have lower medians than CharlieOne's, hence we consider CharlieFour the second best performing agent.

With respect to R2 agents, we can observe that agents' performances were stronger as complexity of the agent decreased. Conversely, for R1 agents this observation seems to hold only up to CharlieThree, with CharlieFour, the most complex agent, being able to perform exceptionally well. Overall, R1 agents had better performances on average than R2 agents, both in deterministic backtesting, as discussed in Section4.2.1, and in stochastic dominance and greatness test against the same baseline. This reinforces our hypothesis that our IRR reward function is more suited to train view-forming agents than the Sharpe reward function. From this point onward, our analysis will focus mainly on R1 agents.

The satisfactory results obtained by CharlieOne and Four prove that wide agents trained through PPO are able to consistently form good views. Interestingly, all Charlie

agents—both R1 and R2—suffered losses in H1 2025, including the top performers CharlieOne and Four. While this may seem a deficiency of our agents, in reality this instance is aligned with the Adaptive Market Hypothesis proposed in Lo (2004). It is likely that during the first half of 2025 a market regime shift happened. Since agents' heuristics were adapted to the previous market regime, their performance suffered in H1 2025 as a direct consequence of the regime shift. What add value to our interpretation is the swift recovery from almost all agents during the second half of 2025, which was faster and stronger for CharlieOne and Four. We can speculate that the recovery happened because agents' heuristics adapted to the new market regime, and that CharlieOne and Four emerged better than their siblings due to them possessing better generalization, which allows them to exploit their observation space more efficiently.

Tables 4.10 and 4.11 contain the results of the KS and MW tests for all Alpha agents, across all backtesting windows. In Section C.3 the reader can instead retrieve the empirical distribution plots for all Alpha agents.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
CharlieOneR1	H1 2024	0.152	0.107	0.018	0.429	T	5.985×10^{6}	0.000	T
CharlieTwoR1	H1 2024	0.139	0.107	0.033	0.064	T	5.479×10^{6}	0.000	T
CharlieThreeR1	H1 2024	0.120	0.107	0.113	1.148×10^{-14}	F	4.255×10^{6}	5.865×10^{-109}	T
CharlieFourR1	H1 2024	0.162	0.107	0.001	0.996	T	6.107×10^{6}	0.000	T
CharlieOneR2	H1 2024	0.163	0.107	0.002	0.990	T	6.123×10^{6}	0.000	T
CharlieTwoR2	H1 2024	0.135	0.107	0.013	0.647	T	5.252×10^{6}	0.000	T
CharlieThreeR2	H1 2024	0.143	0.107	0.004	0.961	T	5.619×10^{6}	0.000	T
CharlieFourR2	H1 2024	0.117	0.107	0.120	2.142×10^{-16}	F	3.968×10^{6}	1.164×10^{-61}	T
CharlieOneR1	H2 2024	0.137	0.116	0.115	4.595×10^{-15}	F	4.938×10^{6}	1.226×10^{-276}	T
CharlieTwoR1	H2 2024	0.106	0.116	0.432	9.897×10^{-210}	F	2.230×10^{6}	1.000	F
CharlieThreeR1	H2 2024	0.112	0.116	0.354	4.496×10^{-140}	F	2.723×10^{6}	1.000	F
CharlieFourR1	H2 2024	0.131	0.116	0.125	8.704×10^{-18}	F	4.455×10^{6}	5.030×10^{-150}	T
CharlieOneR2	H2 2024	0.124	0.116	0.196	1.584×10^{-42}	F	3.882×10^{6}	4.464×10^{-50}	T
CharlieTwoR2	H2 2024	0.111	0.116	0.364	1.406×10^{-147}	F	2.609×10^{6}	1.000	F
CharlieThreeR2	H2 2024	0.101	0.116	0.531	7.905×10^{-323}	F	1.794×10^{6}	1.000	F
CharlieFourR2	H2 2024	0.115	0.116	0.305	3.389×10^{-103}	F	3.030×10^{6}	0.969	F
CharlieOneR1	H1 2025	-0.017	0.075	0.962	0.000	F	6.720×10^{4}	1.000	F
CharlieTwoR1	H1 2025	-0.022	0.075	0.997	0.000	F	3.234×10^{3}	1.000	F
CharlieThreeR1	H1 2025	-0.041	0.075	0.999	0.000	F	223.000	1.000	F
CharlieFourR1	H1 2025	-0.033	0.075	0.998	0.000	F	2.267×10^{3}	1.000	F
CharlieOneR2	H1 2025	9.131	0.075	0.952	0.000	F	1.055×10^{5}	1.000	F
CharlieTwoR2	H1 2025	1.927×10^{13}	0.075	0.992	0.000	F	1.496×10^{4}	1.000	F
CharlieThreeR2	H1 2025	-0.031	0.075	0.999	0.000	F	1040.000	1.000	F
CharlieFourR2	H1 2025	-0.040	0.075	1.000	0.000	F	4.000	1.000	F
CharlieOneR1	H2 2025	0.267	0.168	0.003	0.981	T	6.189×10^{6}	0.000	T
CharlieTwoR1	H2 2025	0.266	0.168	0.004	0.968	T	6.186×10^{6}	0.000	T
CharlieThreeR1	H2 2025	0.244	0.168	0.010	0.794	T	6.006×10^{6}	0.000	T
CharlieFourR1	H2 2025	0.253	0.168	0.006	0.903	T	6.106×10^{6}	0.000	T
CharlieOneR2	H2 2025	0.249	0.168	0.007	0.878	T	6.063×10^{6}	0.000	T
CharlieTwoR2	H2 2025	0.254	0.168	0.006	0.903	T	6.110×10^{6}	0.000	T
CharlieThreeR2	H2 2025	0.228	0.168	0.031	0.088	T	5.733×10^{6}	0.000	T
CharlieFourR2	H2 2025	0.246	0.168	0.008	0.852	T	6.026×10^{6}	0.000	T

Table 4.10: Dominance Results - Agents form the Charlie Family, half yearly runs.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
CharlieOneR1	2024	0.310	0.243	0.052	0.001	F	5.640×10^{6}	0.000	T
CharlieTwoR1	2024	0.259	0.243	0.203	1.106×10^{-45}	F	3.886×10^{6}	1.440×10^{-50}	T
CharlieThreeR1	2024	0.242	0.243	0.312	1.848×10^{-108}	F	3.091×10^{6}	0.750	F
CharlieFourR1	2024	0.311	0.243	0.024	0.249	T	5.631×10^{6}	0.000	T
CharlieOneR2	2024	0.303	0.243	0.038	0.029	F	5.468×10^{6}	0.000	T
CharlieTwoR2	2024	0.257	0.243	0.180	4.333×10^{-36}	F	3.794×10^{6}	1.570×10^{-39}	T
CharlieThreeR2	2024	0.254	0.243	0.194	5.182×10^{-42}	F	3.658×10^{6}	7.218×10^{-26}	T
CharlieFourR2	2024	0.240	0.243	0.312	6.718×10^{-108}	F	2.974×10^{6}	0.998	F
CharlieOneR1	2025	0.238	0.252	0.284	3.045×10^{-89}	F	2.639×10^{6}	1.000	F
CharlieTwoR1	2025	0.231	0.252	0.466	2.923×10^{-246}	F	2.093×10^{6}	1.000	F
CharlieThreeR1	2025	0.191	0.252	0.780	0.000	F	6.364×10^{5}	1.000	F
CharlieFourR1	2025	0.210	0.252	0.646	0.000	F	1.185×10^{6}	1.000	F
CharlieOneR2	2025	0.316	0.252	0.343	6.547×10^{-131}	F	2.256×10^{6}	1.000	F
CharlieTwoR2	2025	0.234	0.252	0.420	6.863×10^{-198}	F	2.231×10^{6}	1.000	F
CharlieThreeR2	2025	0.192	0.252	0.774	0.000	F	6.517×10^{5}	1.000	F
CharlieFourR2	2025	0.194	0.252	0.751	0.000	F	7.022×10^{5}	1.000	F
CharlieOneR1	Full	0.584	0.550	0.226	7.489×10^{-57}	F	4.038×10^{6}	7.662×10^{-72}	T
CharlieTwoR1	Full	0.525	0.550	0.441	4.545×10^{-219}	F	2.465×10^{6}	1.000	F
CharlieThreeR1	Full	0.452	0.550	0.732	0.000	F	9.103×10^{5}	1.000	F
CharlieFourR1	Full	0.555	0.550	0.296	1.537×10^{-97}	F	3.250×10^{6}	0.007	T
CharlieOneR2	Full	0.577	0.550	0.206	4.014×10^{-47}	F	3.850×10^{6}	4.644×10^{-46}	T
CharlieTwoR2	Full	0.533	0.550	0.374	1.169×10^{-156}	F	2.674×10^{6}	1.000	F
CharlieThreeR2	Full	0.474	0.550	0.648	0.000	F	1.284×10^{6}	1.000	F
CharlieFourR2	Full	0.451	0.550	0.729	0.000	F	8.962×10^{5}	1.000	F

Table 4.11: Dominance Results - Agents form the Charlie Family, Y 2024, Y 2025, and 2Y.

Delta Family

Among the Delta family, as already seen in deterministic runs, the results are heterogeneous between the two different reward functions we employed. In Section 4.2.1, we observed that, in general, Sharpe rewarded agents performed better than their IRR rewarded counterparts. This behavior carries also to the stochastic simulations framework.

Among the agents rewarded through our IRR reward function, DeltaOne was the only to obtain remarkable performances in terms of stochastic greatness against the random baseline. Its returns being deemed stochastically greater than the baseline through the Mann-Whitney U test in four out of seven backtesting windows, also attaining FSD in two of those occasions. The performances obtained by the other R1 agents are unremarkable.

As observed in deterministic backtesting, Sharpe rewarded agents performed much better also in stochastic simulations, compared to R1 agents. Unsurprisingly, the best performing agent—not only among its R2 siblings, but within the entire Delta family—was DeltaThree. Specifically, DeltaThree's returns were tested positive for stochastic greatness with respect to random returns in five out of seven occasions, achieving FSD in two of them. Furthermore, DeltaFour and DeltaTwo scored good performances, with DeltaFour being slightly better performing than Two. However, DeltaOne show the weakest performances among the R2 Delta agents.

Overall, within the entire Delta family, four agents had satisfactory performances when compared against the same random baseline: DeltaOneR1, DeltaTwoR2, ThreeR2 and FourR2. Conversely to what observed for agents in the Charlie family, also in stochastic simulations Sharpe rewarded agents in general outperform their IRR counterparts. As discussed when considering deterministic runs, this is likely due to both the off-policy nature of SAC, and to the maximum entropy framework characterizing SAC.

The results obtained with respect to the Delta family are also aligned with Lo's Adaptive Market Hypothesis. Regardless of the degree of generalization of different Delta

agents, all of them struggled massively during the first half of 2025, likely due to a market regime shift. In the H1 2025 backtesting window, all the estimated PDFs of Delta agents' returns are concentrated around an often negative mean—the sole exception being DeltaThreeR2's, which was positive but still close to zero. However, due to the good generalization of Delta policies, all agents recovered exceptionally fast from the regime shift, their returns being always stochastically greater than the baseline in the H2 2025 backtesting window. The agents suffered throughout the regime shift due to their reliance on outdated heuristics. However, once new heuristics, adapted to the new market regime, emerged, agents were able to quickly recover from their struggles.

Agents from the Delta family proved to be able to generate stochastically greater returns than random portfolios, their superior performance being validated by occasional instances of FSD. This implies that they have actually learned a policy able to outperform the market by extracting relevant information from their observation spaces, which is then used to form accurate views, leading to improved posterior portfolios through the BL model. Despite suffering from market shifts, as predicted by the Adaptive Market Hypothesis, the superior generalization provided by the off-policy SAC model allows them to recover quickly, leading them to generate returns outperforming the market.

Tables 4.12 and 4.13 contain the results of the KS and MW tests for all Alpha agents, across all backtesting windows. In Section C.4 the reader can instead retrieve the empirical distribution plots for all Alpha agents.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
DeltaOneR1	H1 2024	0.154	0.107	0.026	0.185	T	$6.013e+6$	0.000	T
DeltaTwoR1	H1 2024	0.143	0.107	0.055	$4.906e-4$	F	$5.678e+6$	0.000	T
DeltaThreeR1	H1 2024	0.146	0.107	0.051	0.001	F	$5.833e+6$	0.000	T
DeltaFourR1	H1 2024	0.134	0.107	0.150	$3.044e-25$	F	$5.251e+6$	0.000	T
DeltaOneR2	H1 2024	0.142	0.107	0.076	6.165e-07	F	5.644e+06	0.000	T
DeltaTwoR2	H1 2024	0.176	0.107	0.001	0.996	T	6.224e+06	0.000	T
DeltaThreeR2	H1 2024	0.180	0.107	0.002	0.990	T	6.231e+06	0.000	T
DeltaFourR2	H1 2024	0.146	0.107	0.044	0.009	F	5.805e+06	0.000	T
DeltaOneR1	H2 2024	0.115	0.116	0.391	$2.274e-171$	F	$2.960e+6$	0.999	F
DeltaTwoR1	H2 2024	0.112	0.116	0.494	$1.291e-277$	F	$2.735e+6$	1.000	F
DeltaThreeR1	H2 2024	0.118	0.116	0.405	$3.509e-184$	F	$3.352e+6$	$4.236e-6$	T
DeltaFourR1	H2 2024	0.115	0.116	0.413	$2.386e-191$	F	$3.038e+6$	0.956	F
DeltaOneR2	H2 2024	0.109	0.116	0.554	0.000	F	2.421e+06	1.000	F
DeltaTwoR2	H2 2024	0.121	0.116	0.306	2.713e-104	F	3.581e+06	1.882e-19	T
DeltaThreeR2	H2 2024	0.117	0.116	0.384	1.261e-164	F	3.170e+06	0.187	F
DeltaFourR2	H2 2024	0.093	0.116	0.686	0.000	F	1.185e+06	1.000	F
DeltaOneR1	H1 2025	-0.008	0.075	0.998	0.000	F	$1.884e+3$	1.000	F
DeltaTwoR1	H1 2025	-0.040	0.075	1.000	0.000	F	0.000	1.000	F
DeltaThreeR1	H1 2025	-0.048	0.075	1.000	0.000	F	0.000	1.000	F
DeltaFourR1	H1 2025	-0.017	0.075	1.000	0.000	F	156.000	1.000	F
DeltaOneR2	H1 2025	-0.039	0.075	1.000	0.000	F	0.000	1.000	F
DeltaTwoR2	H1 2025	-0.045	0.075	1.000	0.000	F	0.000	1.000	F
DeltaThreeR2	H1 2025	0.021	0.075	0.970	0.000	F	5.094e+04	1.000	F
DeltaFourR2	H1 2025	-0.003	0.075	0.999	0.000	F	2.661e+03	1.000	F
DeltaOneR1	H2 2025	0.260	0.168	0.008	0.852	T	$6.164e+6$	0.000	T
DeltaTwoR1	H2 2025	0.256	0.168	0.012	0.698	T	$6.136e+6$	0.000	T
DeltaThreeR1	H2 2025	0.275	0.168	0.005	0.944	T	$6.216e+6$	0.000	T
DeltaFourR1	H2 2025	0.210	0.168	0.128	$1.133e-18$	F	$5.261e+6$	0.000	T
DeltaOneR2	H2 2025	0.280	0.168	0.003	0.975	T	6.222e+06	0.000	T
DeltaTwoR2	H2 2025	0.233	0.168	0.038	0.025	F	5.850e+06	0.000	T
DeltaThreeR2	H2 2025	0.246	0.168	0.022	0.285	T	6.043e+06	0.000	T
DeltaFourR2	H2 2025	0.252	0.168	0.010	0.794	T	6.102e+06	0.000	T

Table 4.12: Dominance Results - Agents form the Delta Family, half yearly runs.

Agent	Window	Agent Mean	Random Mean	KS Statistic	KS P-Value	FSD	MW Statistic	MW P-Value	SG
DeltaOneR1	2024	0.292	0.243	0.116	$2.893e-15$	F	$5.216e+6$	0.000	T
DeltaTwoR1	2024	0.264	0.243	0.271	$2.476e-81$	F	$4.132e+6$	$5.497e-87$	T
DeltaThreeR1	2024	0.278	0.243	0.207	$1.142e-47$	F	$4.796e+6$	$1.763e-235$	T
DeltaFourR1	2024	0.263	0.243	0.294	$5.915e-96$	F	$4.081e+6$	$1.236e-78$	T
DeltaOneR2	2024	0.263	0.243	0.296	1.537e-97	F	4.110e+06	2.666e-83	T
DeltaTwoR2	2024	0.322	0.243	0.032	0.077	T	5.834e+06	0.000	T
DeltaThreeR2	2024	0.322	0.243	0.040	0.017	F	5.836e+06	0.000	T
DeltaFourR2	2024	0.255	0.243	0.288	8.561e-92	F	3.702e+06	5.922e-30	T
DeltaOneR1	2025	0.250	0.252	0.381	$3.534e-162$	F	$3.031e+6$	0.968	F
DeltaTwoR1	2025	0.212	0.252	0.741	0.000	F	$1.211e+6$	1.000	F
DeltaThreeR1	2025	0.210	0.252	0.764	0.000	F	$1.149e+6$	1.000	F
DeltaFourR1	2025	0.177	0.252	0.907	0.000	F	$3.420e+5$	1.000	F
DeltaOneR2	2025	0.230	0.252	0.603	0.000	F	1.994e+06	1.000	F
DeltaTwoR2	2025	0.169	0.252	0.936	0.000	F	2.449e+05	1.000	F
DeltaThreeR2	2025	0.261	0.252	0.330	3.492e-121	F	3.660e+06	4.784e-26	T
DeltaFourR2	2025	0.253	0.252	0.340	9.508e-129	F	3.182e+06	0.131	F
DeltaOneR1	Full	0.577	0.550	0.289	$1.445e-92$	F	$3.866e+6$	$4.099e-48$	T
DeltaTwoR1	Full	0.510	0.550	0.610	0.000	F	$2.042e+6$	1.000	F
DeltaThreeR1	Full	0.522	0.550	0.569	0.000	F	$2.387e+6$	1.000	F
DeltaFourR1	Full	0.466	0.550	0.768	0.000	F	$1.119e+6$	1.000	F
DeltaOneR2	Full	0.500	0.550	0.677	0.000	F	1.778e+06	1.000	F
DeltaTwoR2	Full	0.551	0.550	0.369	2.995e-152	F	3.147e+06	0.335	F
DeltaThreeR2	Full	0.656	0.550	0.084	1.812e-08	F	5.461e+06	0.000	T
DeltaFourR2	Full	0.571	0.550	0.303	7.801e-102	F	3.692e+06	6.238e-29	T

Table 4.13: Dominance Results - Agents form the Delta Family, Y 2024, Y 2025, and 2Y.

4.3 Results

In Section 3.1 we set out five research questions we aimed at answering with our experiment. The reader is referred to Table 3.2 for a summary of our five research goals. The aim of this section is to answer the aforementioned questions thoroughly based on the evidence collected by our experiment and presented in Section 4.2.

4.3.1 Evaluating Narrow Agents

Our first research goal, among the ones set out in Table 3.2, is the following:

> Evaluate the ability of a *narrow* agent to form accurate views consistent with the state of the market.

In light of the results we obtained through our experiment, in this section we aim at evaluating the ability of narrow agents to form consistent views.

Initially, recall that all our agents, narrow and wide, are trained to output absolute views and their associated confidence with respect to the future performance of each single considered stock. The agent-generated views are then used as input for the Black-Litterman model for asset allocation, which integrates them with the prior portfolio to generate posterior weights. The observation space of narrow agents is composed by the past returns of the eight considered stocks. Agents within the same family vary by the length of their look-back windows.

When analyzing the performance of both Alpha and Bravo family, both in deterministic runs—Section 4.2.1—and stochastic simulations—Section 4.2.2—we pointed out the inconsistency and unreliability of narrow agents' performances. It is likely that their observation space does contain very little signal alongside a lot of noise, as predicted by weak market efficiency.

We pointed out that narrow agents likely have learned some simple heuristics, which allow them to manage left tail risk and realize positive returns overall. However, apart from following general market trends, portfolios managed by narrow agents often trail behind benchmarks, and almost always their returns are not stochastically dominant over randomly generated portfolios, and rarely narrow agents' returns are stochastically greater than random returns.

We can claim that views generated by narrow agents are, in a sense, *consistent* with the state of the market—since their portfolios are still able to follow market trends, almost never wandering erratically. However, we cannot claim these views are *accurate*. If they were, narrow agents' managed portfolios would outperform benchmarks and be different from random in multiple occasions, which is almost never the case.

Unfortunately, it seems that narrow agents cannot escape the EMH trap. Our considerations may be regarded as another nail in the coffin of pure technical analysis, which aims at predicting future market trends on the basis of past returns. Regrettably for technical traders, our sophisticated *feeling-less* computerized agents were not able to find meaningful patterns in past returns, in turns demonstrating incapacity to form accurate market views. This would suggest that profits made through technical analysis are fortunate occurrences, rather than the outcome of working strategies.

4.3.2 Evaluating Wide Agents

Our second research goal, as set out in Table 3.2, is the following:

> Evaluate the ability of a *wide* agent to form accurate views consistent with the state of the market **and** able to generate portfolios performing better than some benchmarks.

As for narrow agents in the previous section, in light of the results obtained through our experiment, in this section we aim at assessing the ability of wide agents to form accurate and consistent views.

Unlike narrow agents, when analyzing the performances of wide agents, both in deterministic backtesting and in stochastic simulations, we found that some agents were able to consistently outperform their benchmarks. Furthermore, some wide agents frequently tested positive for stochastic greatness and, occasionally, for the stricter FSD.

The first main difference we pointed out between the families of wide agents is constituted by the reward function inversion between PPO and SAC trained agents. The best performing PPO agents were IRR rewarded, while the best performers within the SAC agents were Sharpe rewarded. We observed that IRR, due to its highly specific, isolated nature, is likely more suited to on-policy algorithms, such as PPO, due to it being less temporally robust than the Sharpe ratio. IRR's limited temporal robustness is likely due to the existence of market regime shifts. An action which was rewarded positively during a past market regime may lead to a negative reward once the regime shifts. This characteristic of IRR makes it ill suited for replay buffer based, off-policy RL algorithms such as SAC.

Furthermore, we observed that, when trained using the best performing reward function, Charlie's best performer was the least complex CharlieOne, while Delta's best performer was the second most complex DeltaThree. Overall, it seems that PPO favors low complexity agents, while SAC tends to reward higher complexity.

Despite the aforementioned differences in agents' behavior between the two wide families, we pointed out that multiple agents per family performed satisfactorily, both when compared against static benchmarks and when tested for stochastic greatness and dominance against random portfolios. In particular, within the Charlie family three agents had satisfactory performances: CharlieOneR2, CharlieOneR1, and CharlieFourR1. Concurrently, four Delta agents were deemed capable to reliably form good views: DeltaOneR1, DeltaTwoR2, ThreeR2, and FourR2.

We can therefore claim that wide agents, regardless of their optimal degree of complexity and the optimal reward function, are able to form views which are both *consistent* and *accurate*. Furthermore, wide agents' generated portfolios are often able to reliably outperform their benchmarks.

Since the only difference between wide and narrow agents is in their observation spaces, we can confidently claim that the heuristics learned by wide agents are able to extract the information needed to form *good* views because their observation spaces contain more signal than the one of narrow agents.

We also concluded that wide agents' view forming heuristics are likely dependent on the current market regime, in complete accordance with the Adaptive Market Hypothesis. In fact, when shifts in market regime happen, wide agents always underperform random portfolios and their static benchmarks. Once the shift has been completed, their heuristics are able to extract reliable information from their observation space, which lead to a fast recovery, the speed of which is depending on how well generalized is the agent's policy.

In conclusion, not only wide agents are able to form accurate views, which through the BL model translate in excess returns, but we also assessed that their behavior is fully

consistent with the AMH, leading us to conclude that the empirical evidence we collected is in support of Lo's conjecture.

4.3.3 Agents vs Randomness

Our third research goal, as stated in Table 3.2, follows:

> Evaluate the ability of both *narrow* and *wide* agents to build portfolios able to outperform naively diversified ones.

Recall that in Section 3.2.3 we defined *naively diversified portfolios* as all portfolios that could be built through the application of unsophisticated investment rules. In particular, in our study we considered two different classes of naive portfolios: the equally weighted (EW) portfolio and randomly weighted portfolios.

The two classes of naive portfolios serve two different purposes. The EW portfolio serves as a benchmark in deterministic runs, while randomly weighted portfolios' returns were used in stochastic dominance and greatness testing.

First, we consider the two families of narrow agents: Alpha and Bravo. With respect to the EW benchmark, we pointed out that agents of the Alpha family very rarely were able to outperform the equally weighted portfolio in deterministic backtesting. However, BravoTwo and Four were occasionally able to outperform their benchmarks in deterministic runs, including the EW portfolio. In stochastic simulations, both Alpha's and Bravo's returns were almost never stochastically dominant over the random baseline, being also rarely deemed stochastically greater with respect to the same random returns.

So, are narrow agents able to build portfolios able to outperform naive investors? The answer is, most likely, not. Narrow agents often tail far behind their benchmarks in terms of returns. While it is unclear whether investors would be better off tossing darts blindly towards stock sheets rather than letting our technical-analysis narrow agents to manage their portfolio—occasionally agents' returns are stochastically greater than the baseline, and show lower variance than random returns—it is clear that an investing strategy as simple as equally weighting a portfolio could return higher than our narrow agents.

The two families of wide agents, Charlie and Delta, had vastly superior performances. We pointed out that different agents per family—CharlieOneR2, OneR1, and FourR1 for the Charlie family, and DeltaOneR1, TwoR2, ThreeR2, and FourR2 for the Delta family—were able to both massively outperform is deterministic benchmarks and test positive stochastic greatness, occasionally validated by FSD, reliably across the majority of backtesting windows. Despite the presence of under-performers in wide families, the mere existence of at least one agent capable to generate excess returns consistently proves that, given optimal complexity and the use of the correct reward function, wide agents are able to manage portfolios able to beat naively diversified ones.

4.3.4 Dynamic vs Static

Our fourth research goal, as stated in Table 3.2, follows:

> Evaluate the performance of dynamic portfolio management conducted by RL agents with respect to buy-and-hold strategies.

The main critique moved to dynamic portfolio management regards the impossibility of reliably generating excess returns capable to offset transaction costs. Buy-and-hold strategies, on the other hand, are virtually immune to transaction costs.

When comparing the performance attained by our narrow agents in deterministic runs, we observed that Alpha and Bravo agents' portfolios frequently trailed behind static GMV portfolios. At the same time, we found our wide agents to be capable of reliably outperforming the same benchmarks. We concluded that the difference in performance between narrow and wide agents is due to their different observation spaces. Due to the limited amount of information contained in past returns, as predicted by the weak EMH form, narrow agents are incapable to extract the information they need to generate good views, which would lead to improved posterior portfolios through the BL model. Concurrently, we observed that within the more information dense wide observation space, agents were able to extract more information and, in turns, consistently beat their static benchmarks.

While we can confidently conclude that narrow agents are unlikely to manage portfolios able to beat static strategies consistently, we deem wide agents to possess the ability to do so, provided that they achieve optimal complexity and the most suitable reward function is used. Furthermore, we may point out that, if acting in absence of transaction costs, optimal complexity may not be needed at all, since most trailing wide agents were still close to benchmark performance. Considering the immense burden of transaction costs, which may reduce annual returns by several percentage points, it is likely they are able to generate better investment strategies than static portfolios, despite sub-optimal complexity. Conversely, the use of a suitable reward function is pivotal for training capable agents.

Despite the remarkable results obtained by wide agents, we have to acknowledge that our RL agents are extremely more sensitive to market regime shifts than static portfolios. This is probably due to their heuristics taking time to adapt to new market regimes, in accordance with the Adaptive Markets Hypothesis. However, their superior generalized policies allow them to recover quickly from shifts in market regimes, eventually outperforming benchmarks in the long run.

4.3.5 PPO or SAC?

Our fifth research goal, as stated in Table 3.2, follows:

> Thoroughly analyze the differences between the two training algorithms employed in this study.

The two algorithms we employed to train our agents were PPO—Alpha and Charlie families—and SAC—Bravo and Delta families. The first, PPO, is an on-policy algorithm, while the second, SAC, is off-policy.

The first, and perhaps most consequential difference between PPO and SAC is their different aptitude for different reward functions. As mentioned while discussing wide agents, PPO trained a better policy when using the IRR reward function. Conversely, SAC trained a better policy under Sharpe reward. While the isolated and highly specific nature of our IRR reward function proved to be highly beneficial for the on-policy PPO, the replay buffer based, off-policy SAC converged better while rewarded using the more general Sharpe ratio. Recall that in Section 1.2 we pointed out that the literature presents different claims regarding the supposed convergence struggles of SAC. It is likely that SAC failed to converge because an ill-suited reward function was used.

The second major difference between the two algorithms, one which could potentially impact real-world applicability of our findings, was pointed out in Section 4.1. PPO is

significantly faster to converge to an optimal policy than SAC. This is both due to the on-policy nature of PPO and to the fact that PPO has to train just two ANNs, the actor and the critic, while SAC has to train three: the actor and two twin critic networks. The difference in training time is not tiny either. The largest PPO agents took just a few hours to train, while the largest SAC models took up to three days.

From the training logs—which are not reported due to each one of them having as many lines as twenty *The Lord of the Rings* volumes—we could observe that both PPO and SAC, with their hyperparameters properly tuned, provided with very stable learning, characterized by rapidly dropping entropy, which controls the exploration of the agents, actor loss function value, and critic loss function value. Most models triggered the early stopping mechanism on most windows.

We may point out that, from the tables contained in Section A.2, the reward obtained in validation by SAC agents was, on average, slightly greater than the one obtained by the respective PPO agents. This may be interpreted as a sign of mildly superior training performance from SAC over PPO.

While we refrain from comparing the performances obtained by Alpha and Bravo families, since they both were deemed largely unsatisfactory, we may want to use Charlie and Delta for comparison. While both families' performances were assessed as satisfactory, we can notice that, on average, Delta's performances were superior to Charlie's in multiple occasions, both in deterministic runs and in stochastic simulations.

Can we establish which algorithm is superior within our experimental framework? The answer is nuanced. While SAC provided better overall performances and a more stable learning, it was significantly more time demanding to train with respect to PPO. Moving from a backtesting to a live application framework could be non-trivial for SAC, since adequate training may be sacrificed in favor of acting fast. We can state that it seems off-policy algorithms, like SAC, have a slight edge over on-policy algorithms, like PPO, in our framework, mainly given by the superior generalization ability of off-policy algorithms, which is more than needed when decision making passes through noise dense environments.

4.4 Final Considerations

Throughout this thesis we studied thoroughly the view generating capability of reinforcement learning agents. We studied said ability on a multilayered perspective, uncovering its working principles.

First, we related the view generating ability of agents to the information density of their observation spaces by training agents on narrow and wide datasets. The results obtained with this regard are unequivocal: agents can form views reliably if and only if their observation space is information dense. In our study, the wide dataset we employed contained just nine stock-specific features per each stock, plus five global macroeconomic features, for a total of seventy-seven features. We had to limit the number of features used due to time and computational constraints—the higher the number of features the longer the computational time needed to train the agents—but a natural extension of this study would be to replicate it using more features, being both stock-specific and global macroeconomic.

The empirical evidence we gathered with respect to the information density of agents' observation spaces seems to support the weak form of the Efficient Market Hypothesis (EMH), while at the same time it seems to contradict the semi-strong form of market efficiency. This inference is drawn by the fact that, despite narrow agents being unable to generate views reliably, wide agents were instead able to do so. Since the only difference between the two classes of agents was in their respective observation spaces, we can conclude that the wide dataset was more information dense than the narrow one. However, while the narrow dataset is composed only of past returns, hence falling under the weak market efficiency wing, the wide dataset contains multiple features, all of which are publicly available, so wide agents should be governed by the semi-strong form of market efficiency. The fact that wide agents were able to generate reliable views contradicts semi-strong market efficiency.

Do our results imply we should outright reject semi-strong market efficiency? Not necessarily. While our results—and our narrative–are compelling, we should remember that our experiment included only eight stocks. Despite the considered stocks being all high capitalization blue chips—theoretically possessing high informational efficiency—a more thorough study, involving a higher number of stocks, potentially from different markets, is needed to definitively reject the semi-strong—consequently, also the strong—form of the EMH. Our study shows that the developed framework is highly viable for testing the Efficient Market Hypothesis, and that such study is possible.

Secondly, we related the view generating ability of agents to their neural network architecture and their temporal awareness. We achieved this by training agents both with different architectures and different look-back windows. Despite different families of agents behaving differently with respect to the optimal network size or the optimal length of their look-back windows, our results show that agents can form views reliably and consistently, in presence of transaction costs, if and only if they possess optimal, or near-optimal, network architecture and look-back window.

Furthermore, we studied how the use on-policy and off-policy algorithms to train our agents affected their out of sample performances. While we assessed that both the proposed on-policy algorithm (PPO) and the off-policy algorithm (SAC) were able to deliver highly satisfactory results, we also acknowledged that the two classes of algorithms work best with differently designed reward functions. In particular, the highly specific IRR reward function we proposed was well suited for training the agents through PPO,

while we found that SAC training delivered better policies when using a more general and time robust reward function, such as the Sharpe ratio. We regard this point to be of exceptional importance, since it could potentially explain why early research found off-policy methods to show convergence hurdles. Future research should address in more detail the relationship between the class of RL algorithms and its most suitable reward function design, perhaps replicating this experiment taking into consideration multiple RL algorithms, both on and off-policy, and multiple reward functions with varying levels of specificity.

The way we addressed the view forming ability of our RL agents was by letting them dynamically manage diversified equity portfolios through the Black-Litterman (BL) model for asset allocation. The BL model was a natural candidate to support the study of view generation by RL agents due to its two-steps mechanism—which begins with the prior portfolio, which is then *enriched* with agent's views. Furthermore, the BL model served as a safeguard in case agents were not able to form any views at all, since with no views only the prior portfolio became the agents' portfolio, avoiding risky erratic behavior[2]. The evidence we collected supports the hypothesis that optimal reinforcement learning agents indeed possess highly functional view generating abilities, which allows them to successfully manage dynamic equity portfolios able to beat buy-and-hold strategies and random allocations also in presence of transaction costs. However, the same evidence we collected tells us that RL agents managing portfolios through the BL model are especially sensitive to market regime shifts as predicted by the Adaptive Market Hypothesis (AMH). Future research should, in our opinion, move in two directions. First, future research should assess whether using a support asset allocation model is beneficial for the agents, or if they would be better off choosing portfolio weights directly. Second, future research should formalize the effect of market regime shifts on RL agents' view generation ability, in order to develop more shift-robust frameworks than the one we used.

Hypotheses supported by our results	
1.	RL agents are capable of forming reliable views provided that they possess: an information dense observation space, optimal network architecture, and optimal temporal awareness.
2.	Both on-policy and off-policy algorithms can train highly functional policies, provided that the adequate reward function for each class is used. On-policy methods require a highly specific reward function, while off-policy methods work best with a more generic reward function.
3.	Due to their functional view generating abilities, RL agents can dynamically manage diversified equity portfolios able to outperform buy-and-hold benchmarks in most cases, even in presence of transaction costs.
4.	The view generating ability of RL agents is dependent on heuristics which are sensitive to shifts in market regime, in accordance with the AMH.
5.	The financial market seems to be at most weakly efficient, with the semi-strong form of the EMH being rejected by our empirical results.

Table 4.14: Main findings.

Table 4.14 summarizes the five main hypotheses supported by our study, which we hope future research will focus on, both for validating them, to expand them, or, eventu-

[2]However, we must say, it did not offer any safeguard with respect to *bad* views.

ally, to disprove them. Notice that we refer to them as hypotheses since they are nothing more than apparently empirically supported conjectures and, for most of them, rigorous dedicated studies will be needed to fully explore them, and perhaps to provide a rigorous analytical proof where applicable.

In conclusion, are we in front of a paradigm shift in how we interpret finance? The answer is, almost certainly, yes. Recall what we observed in Section 1.2. The financial academic world, and consequently the financial industry, is witnessing an ever-growing interest in machine learning methods applied to financial problems. With our study, we provided additional evidence that a path towards autonomous, data-driven, and model-free finance is possible. While we are likely still distant from achieving perfect automated methods, it is beyond doubt that the potential contained within this new approach to finance is enormous. Probably, we are still far from dropping restrictive models entirely; however, it is evident that we should start rethinking finance with a bottom-up approach, substituting the human pretense of describing thoroughly the mechanisms driving the market through restrictive, and often over-simplified models, with new data-driven approaches to then rebuilding our knowledge on top.

Appendix A

Training

This section contains tables relative to the training of our agents. Tables from A.1 to A.4 contain the hyperparameters specifications used for training each agent from each family. Tables from A.5 to A.10 contain instead the validation rewards obtained by each agent per each training window, alongside the mean validation reward displayed at the bottom of each table. Please notice that families Charlie and Delta have two validation logs tables each due to them being trained using both the IRR reward function and the Sharpe reward function. The reader may also observe that the validation rewards vary massively with the change in reward function. This is due exclusively to the use of a different reward scaling parameter for the two reward functions.

A more comprehensive discussion of agents' training is provided in Section 4.1. There the reader may find information about the technical specifications of our experimental setup, hyperparameter tuning, and the differences in training time for the two algorithms employed in this study.

A.1 Hyperparameters

Hyperparameter	**Value**
Learning Rate	0.0001
Number of Steps (`n_steps`)	512
Batch Size	256
Number of Epochs	10
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Clip Range	0.1
Entropy Coefficient	0.01
Max Gradient Norm	0.5

Table A.1: Shared hyperparameter specifications for the Alpha agent family.

Hyperparameter	Small Agents	Large Agents
Learning Rate	0.0003	0.0003
Entropy Coefficient (`ent_coef`)	'auto'	'auto'
Train Frequency (`train_freq`)	8	8
Gradient Steps	32	32
Batch Size	512	1024
Buffer Size	200,000	200,000

Table A.2: Hyperparameter specifications for the Bravo agent family using the Soft Actor-Critic (SAC) algorithm. One and Two are *small* agents, due to their ANN size. For the same reason, Three and Four are considered *large* agents.

Hyperparameter	Small Agent	Large Agents
Learning Rate	3×10^{-4}	3×10^{-5}
Number of Steps (`n_steps`)	512	4096
Batch Size	64	512
Discount Factor (γ)	0.995	0.995
GAE Lambda (λ)	0.97	0.97
Clip Range	0.1	0.1
Entropy Coefficient (`ent_coef`)	0.005	0.02

Table A.3: Hyperparameter specifications for the Charlie agent family using the Proximal Policy Optimization (PPO) algorithm. One is a *small* agent, due to its ANN size. For the same reason, Two, Three, and Four are considered *large* agents.

Hyperparameter	Small Agents	DeltaThree	DeltaFour
Learning Rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Buffer Size	300,000	300,000	300,000
Batch Size	512	1024	1024
Train Frequency (`train_freq`)	1	1	8
Gradient Steps	4	4	32
Learning Starts	5,000	5,000	5,000
Target Smoothing Coefficient (τ)	0.005	0.005	0.005
Discount Factor (γ)	0.99	0.99	0.99
Entropy Coefficient (`ent_coef`)	'auto'	'auto'	'auto'

Table A.4: Hyperparameter specifications for the Delta agent family using the Soft Actor-Critic (SAC) algorithm. The agents are categorized by size: Small Agents share identical parameters, while the Large Agents feature a larger batch size and differ from one another in their training frequency and gradient steps.

A.2 Validation Rewards

W	Train Start	Train End	Val Start	Val End	Two	One	Four	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	-0.043	-0.065	-0.157	0.346
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-0.147	-0.068	0.556	-0.375
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	-0.420	-0.160	-0.118	-0.280
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	-0.373	-0.836	-0.221	-0.548
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	-0.407	-0.228	0.123	-0.315
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	0.544	1.041	-0.452	0.144
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	0.205	-0.418	0.342	-0.113
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	-0.136	-0.075	0.204	0.287
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	0.157	-0.465	-0.095	-0.074
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	-0.029	0.279	0.065	-0.558
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	-0.210	0.025	-0.232	-0.199
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	-0.319	-0.058	-0.472	-0.091
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	-0.616	-0.105	0.135	-0.684
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	-0.564	-0.400	-0.067	1.341
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	-0.403	-0.959	-0.803	-0.485
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	0.069	-0.659	-0.160	0.290
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	-0.039	0.045	-0.447	-0.266
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	-0.235	0.051	-0.197	-0.245
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	-0.108	0.308	-0.291	0.484
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	-0.269	0.191	-0.028	-0.296
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	0.131	0.276	-0.493	1.361
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	-0.416	-0.144	-0.433	-0.128
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	-0.212	-0.199	-0.715	-0.019
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	-0.197	-0.131	0.109	-0.417
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	0.385	-0.538	-0.389	0.153
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	-0.120	-0.298	0.387	0.466
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	-0.577	-0.494	-0.752	0.361
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	0.413	-0.267	-0.271	-0.899
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	-0.711	0.074	-0.242	-0.426
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	-0.322	-1.127	-0.555	-0.745
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	-0.230	-0.500	0.258	0.286
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	-0.084	-0.225	-0.120	0.012
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	-0.106	-0.109	0.222	-0.219
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	0.114	0.406	0.337	0.734
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	-0.219	-0.403	0.038	0.706
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	-0.228	-0.774	-0.178	-0.525
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	-0.384	-0.275	-0.285	-1.410
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	-1.169	-0.539	-0.438	-0.660
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	-0.373	0.313	0.408	-0.371
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	0.131	0.090	0.400	0.201
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	0.396	-0.025	0.070	0.143
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	-0.563	0.661	-0.181	-0.488
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	-0.573	-1.776	-1.148	-0.951
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	-0.145	0.253	-0.286	-0.324
Mean Reward					-0.191	-0.189	-0.149	-0.109

Table A.5: Complete validation logs for agents of the Alpha Family

W	Train Start	Train End	Val Start	Val End	Four	Two	One	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	0.478	0.020	-0.031	-0.200
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-0.847	-0.551	-0.294	-0.099
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	0.399	-1.685	-0.350	-0.046
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	0.517	0.111	0.243	-0.144
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	-0.389	0.311	0.819	0.212
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	-0.615	-0.433	-0.242	-0.183
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	-0.235	-0.165	-0.145	-0.392
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	0.476	0.358	-0.698	-0.227
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	0.079	-0.374	0.852	-1.117
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	-0.240	-0.417	-0.161	-0.324
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	-0.059	-0.363	-0.601	-0.170
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	-0.197	0.542	0.683	1.182
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	0.129	0.127	0.959	1.186
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	0.917	-0.221	-0.914	1.898
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	-0.997	0.923	-0.154	-0.593
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	-0.885	-0.353	0.660	0.497
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	-0.425	-0.667	-1.476	-0.453
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	-0.266	-0.503	-0.076	-0.327
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	-0.249	-0.287	-0.896	0.221
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	0.427	-0.863	-0.711	-0.197
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	-1.137	-0.891	0.576	0.045
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	-0.047	1.319	1.145	-0.460
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	-1.295	0.275	0.022	-0.589
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	0.079	-0.330	0.479	-0.250
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	-0.537	-0.511	0.505	-0.390
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	0.185	0.675	-0.898	0.526
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	-0.359	0.894	0.278	-0.389
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	0.613	-0.035	0.044	-0.558
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	0.538	-0.884	-1.171	-0.037
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	0.280	0.589	-0.771	-0.063
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	-0.256	-0.100	-0.075	-0.856
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	-0.244	0.094	-1.021	-0.068
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	0.124	-0.338	-1.031	-0.516
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	0.985	0.194	1.413	0.746
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	0.518	1.458	-0.469	0.457
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	-0.776	-0.726	1.243	-1.155
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	0.838	-0.944	-1.008	-1.933
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	-1.923	-0.123	1.181	-0.076
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	-0.277	0.063	-0.482	-0.902
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	-0.220	-0.210	-0.462	-0.983
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	1.234	-0.383	-1.134	0.482
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	0.197	-0.022	0.126	-0.822
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	-0.299	-0.525	-0.786	-1.179
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	-1.253	-0.076	-0.902	-0.324
Mean Reward					-0.114	-0.114	-0.130	-0.195

Table A.6: Complete validation logs for agents of the Bravo Family

W	Train Start	Train End	Val Start	Val End	Four	One	Two	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	1.025	1.237	1.094	1.037
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-2.473	-2.518	-2.566	-2.465
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	2.512	2.496	2.827	3.227
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	0.204	0.263	0.780	0.546
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	1.918	2.410	2.151	1.659
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	-0.248	0.039	-1.187	-0.944
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	1.267	-2.370	-1.754	0.848
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	-0.433	3.882	0.603	2.527
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	-0.172	1.608	0.499	-0.020
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	2.027	5.728	3.122	3.262
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	-0.266	1.164	0.509	0.641
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	5.183	4.570	2.801	3.736
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	-2.754	-4.078	-2.373	-3.514
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	-2.879	-1.114	-2.060	-5.838
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	2.541	2.248	4.168	2.765
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	0.196	1.414	0.240	1.438
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	-1.078	0.357	-2.578	-0.746
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	0.535	1.575	1.216	1.086
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	0.902	0.478	0.448	2.306
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	0.524	2.989	0.920	0.448
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	0.935	0.671	1.738	0.780
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	1.703	-0.542	-0.316	-0.329
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	0.024	-1.058	-1.107	-0.948
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	-0.635	3.436	-0.836	-1.346
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	1.703	3.710	1.755	1.079
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	0.603	2.668	-2.082	-0.366
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	-0.694	-1.580	-1.847	0.530
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	-0.424	-0.333	-0.194	-1.230
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	-0.821	2.046	0.585	-0.962
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	-0.327	3.397	0.337	-0.779
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	-2.161	3.538	-2.943	-2.670
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	-1.344	-0.510	-1.022	-1.116
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	4.303	4.589	1.016	2.188
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	0.981	1.929	-1.668	-2.809
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	2.668	-0.714	2.073	2.118
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	-2.108	-2.052	-1.337	1.699
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	-3.332	-3.804	-1.128	0.396
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	-3.844	-4.752	-6.886	-6.216
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	-3.263	-2.501	-2.087	-2.038
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	1.657	2.917	1.077	1.629
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	-2.444	2.333	-2.293	-1.910
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	3.607	2.848	1.181	0.796
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	2.940	6.663	1.912	0.059
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	3.178	1.493	2.292	0.396
Mean					0.260	1.063	-0.066	0.022

Table A.7: Complete validation logs for agents of the Charlie Family—IRR reward function.

W	Train Start	Train End	Val Start	Val End	Four	One	Two	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	6.774	7.110	7.096	6.605
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-10.584	-9.886	-10.432	-10.531
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	2.838	1.323	2.617	2.955
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	1.775	-0.429	1.985	1.634
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	22.109	24.741	25.074	23.306
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	6.654	5.718	3.198	4.754
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	12.565	14.652	12.895	13.738
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	14.070	13.688	13.120	13.683
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	7.641	6.410	6.673	8.078
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	27.597	27.690	29.558	27.595
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	26.647	24.261	24.641	21.882
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	26.842	26.465	26.568	28.633
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	9.360	5.165	9.367	8.434
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	-13.013	-11.180	-9.429	-12.286
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	2.295	2.227	2.497	1.718
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	28.053	22.020	25.870	28.161
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	9.266	11.707	7.043	10.682
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	12.131	16.123	17.004	14.721
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	38.843	37.040	33.153	38.144
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	3.940	4.664	2.748	4.176
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	21.052	21.919	22.343	22.149
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	13.709	12.443	14.569	13.491
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	1.966	8.617	9.086	3.522
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	23.337	28.113	26.668	23.026
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	17.456	19.579	19.574	18.840
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	36.561	44.006	33.739	35.404
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	-1.846	-1.647	-4.973	0.255
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	-5.430	-6.122	-7.042	-7.630
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	1.839	4.090	6.748	-0.470
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	5.686	6.711	8.469	7.678
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	23.061	26.802	28.148	26.136
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	31.254	31.954	34.815	31.666
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	3.299	7.520	3.327	8.547
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	11.491	10.170	14.351	12.296
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	10.564	10.542	11.303	9.810
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	37.320	38.503	32.662	35.311
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	23.030	21.975	15.950	20.912
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	33.505	32.445	35.443	27.030
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	2.283	2.291	1.177	5.282
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	24.905	27.347	26.505	27.712
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	0.865	7.280	4.441	2.720
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	7.638	5.070	7.039	7.545
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	3.455	3.032	7.721	6.996
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	20.944	18.403	25.693	23.539
Mean Reward					13.267	13.876	13.841	13.587

Table A.8: Complete validation logs for agents of the Charlie Family—Sharpe reward function.

W	Train Start	Train End	Val Start	Val End	One	Two	Four	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	2.084	1.852	1.464	1.861
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-0.507	-2.127	-0.250	0.685
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	-0.605	0.071	1.721	2.315
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	1.212	-0.146	1.666	1.099
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	3.296	0.829	3.506	1.094
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	0.988	1.110	2.525	2.005
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	1.209	3.626	3.544	4.272
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	1.751	3.159	-0.316	1.162
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	1.537	0.513	1.037	1.441
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	4.826	4.679	1.375	2.409
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	0.253	1.620	0.820	2.312
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	3.384	1.628	0.394	5.800
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	-1.239	0.155	-3.092	1.292
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	1.224	4.062	-1.481	-0.751
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	-1.019	4.709	0.111	0.615
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	6.091	4.035	7.930	1.964
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	5.394	0.976	5.376	2.243
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	1.611	2.168	0.553	2.196
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	0.722	1.274	0.004	0.503
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	1.219	0.631	1.402	3.409
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	-4.676	2.038	3.609	5.458
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	-0.830	-0.081	-0.421	-1.201
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	5.531	-3.180	-0.661	4.589
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	0.035	-0.514	3.885	1.266
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	1.630	0.827	2.341	1.450
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	2.099	-1.839	4.542	1.189
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	4.069	2.686	4.601	0.475
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	0.164	1.197	0.253	0.337
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	-3.200	0.257	-5.602	2.467
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	2.379	1.145	0.876	-2.651
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	-2.662	-1.718	-4.383	-4.509
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	1.799	0.597	0.432	0.533
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	2.259	0.141	-1.380	2.876
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	0.283	-1.845	2.627	-0.792
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	1.993	0.445	-1.161	2.854
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	-0.851	1.364	-5.316	5.198
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	4.021	2.474	-3.022	3.501
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	1.563	3.471	-6.048	2.934
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	-3.243	-1.585	-0.808	0.097
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	-4.430	-0.830	-6.025	0.421
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	5.345	-0.277	2.225	-1.525
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	2.704	-3.555	3.238	-2.367
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	4.564	5.425	0.446	7.515
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	3.373	2.461	1.500	1.740
Mean					1.303	0.998	0.546	1.586

Table A.9: Complete validation logs for agents of the Delta Family—IRR reward function.

W	Train Start	Train End	Val Start	Val End	One	Two	Four	Three
1	2000-06-01	2001-12-01	2001-12-01	2002-06-01	9.437	8.963	7.991	6.657
2	2000-12-01	2002-06-01	2002-06-01	2002-12-01	-9.412	-9.855	-10.862	-9.813
3	2001-06-01	2002-12-01	2002-12-01	2003-06-01	-0.766	1.566	2.668	3.280
4	2001-12-01	2003-06-01	2003-06-01	2003-12-01	-0.370	0.567	3.492	1.083
5	2002-06-01	2003-12-01	2003-12-01	2004-06-01	21.227	23.189	23.768	21.892
6	2002-12-01	2004-06-01	2004-06-01	2004-12-01	7.132	6.778	7.521	6.768
7	2003-06-01	2004-12-01	2004-12-01	2005-06-01	15.053	15.260	13.880	13.774
8	2003-12-01	2005-06-01	2005-06-01	2005-12-01	15.548	16.211	14.390	15.617
9	2004-06-01	2005-12-01	2005-12-01	2006-06-01	10.351	8.287	5.914	8.802
10	2004-12-01	2006-06-01	2006-06-01	2006-12-01	23.756	21.177	23.903	23.570
11	2005-06-01	2006-12-01	2006-12-01	2007-06-01	30.411	27.095	24.657	25.953
12	2005-12-01	2007-06-01	2007-06-01	2007-12-01	24.633	25.682	26.615	25.838
13	2006-06-01	2007-12-01	2007-12-01	2008-06-01	8.912	7.629	9.399	13.035
14	2006-12-01	2008-06-01	2008-06-01	2008-12-01	-10.103	-9.733	-10.028	-7.732
15	2007-06-01	2008-12-01	2008-12-01	2009-06-01	2.064	4.956	7.596	3.272
16	2007-12-01	2009-06-01	2009-06-01	2009-12-01	27.007	25.380	24.975	23.170
17	2008-06-01	2009-12-01	2009-12-01	2010-06-01	17.778	19.168	17.692	19.748
18	2008-12-01	2010-06-01	2010-06-01	2010-12-01	14.014	12.488	13.635	19.795
19	2009-06-01	2010-12-01	2010-12-01	2011-06-01	33.351	36.406	32.500	35.868
20	2009-12-01	2011-06-01	2011-06-01	2011-12-01	5.437	5.779	6.414	7.127
21	2010-06-01	2011-12-01	2011-12-01	2012-06-01	23.987	25.980	25.852	17.741
22	2010-12-01	2012-06-01	2012-06-01	2012-12-01	12.101	10.593	12.286	13.106
23	2011-06-01	2012-12-01	2012-12-01	2013-06-01	9.890	9.381	17.432	15.001
24	2011-12-01	2013-06-01	2013-06-01	2013-12-01	30.608	29.051	27.081	21.828
25	2012-06-01	2013-12-01	2013-12-01	2014-06-01	17.499	15.404	17.446	17.460
26	2012-12-01	2014-06-01	2014-06-01	2014-12-01	51.495	50.338	47.873	52.054
27	2013-06-01	2014-12-01	2014-12-01	2015-06-01	3.656	2.080	1.513	-1.470
28	2013-12-01	2015-06-01	2015-06-01	2015-12-01	-4.965	-6.103	-7.579	-2.011
29	2014-06-01	2015-12-01	2015-12-01	2016-06-01	4.619	-2.689	-0.225	8.261
30	2014-12-01	2016-06-01	2016-06-01	2016-12-01	5.497	9.621	4.264	3.449
31	2015-06-01	2016-12-01	2016-12-01	2017-06-01	29.530	20.263	20.537	37.130
32	2015-12-01	2017-06-01	2017-06-01	2017-12-01	37.994	39.720	42.765	41.534
33	2016-06-01	2017-12-01	2017-12-01	2018-06-01	6.845	10.429	5.450	8.284
34	2016-12-01	2018-06-01	2018-06-01	2018-12-01	9.213	10.030	17.778	12.016
35	2017-06-01	2018-12-01	2018-12-01	2019-06-01	8.990	9.059	9.123	9.423
36	2017-12-01	2019-06-01	2019-06-01	2019-12-01	41.121	36.694	37.359	38.363
37	2018-06-01	2019-12-01	2019-12-01	2020-06-01	20.403	23.319	22.344	20.711
38	2018-12-01	2020-06-01	2020-06-01	2020-12-01	36.031	39.195	34.269	36.402
39	2019-06-01	2020-12-01	2020-12-01	2021-06-01	0.979	4.856	6.842	8.015
40	2019-12-01	2021-06-01	2021-06-01	2021-12-01	23.006	24.617	22.410	23.265
41	2020-06-01	2021-12-01	2021-12-01	2022-06-01	3.347	3.209	4.303	7.723
42	2020-12-01	2022-06-01	2022-06-01	2022-12-01	6.734	6.333	7.551	6.653
43	2021-06-01	2022-12-01	2022-12-01	2023-06-01	12.246	9.809	10.087	4.492
44	2021-12-01	2023-06-01	2023-06-01	2023-12-01	20.381	20.668	21.487	22.440
Mean Reward					14.924	14.747	14.872	15.445

Table A.10: Complete validation logs for agents of the Delta Family—Sharpe reward function.

Appendix B

Deterministic Backtesting Runs

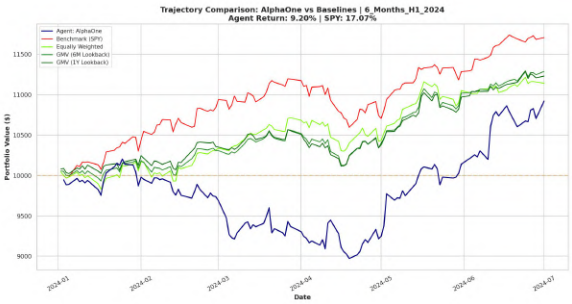
B.1 Alpha Family

AlphaOne's deterministic policy—Figures from B.1 to B.3—had poor performance with respect to its benchmarks. AlphaOne was able to beat its benchmarks in just one occasion, the second half of 2024. In all other backtesting windows, AlphaOne obtained lower returns than its benchmarks. Notably, AlphaOne massively underperformed in the full two-years run. Nevertheless, it should be noted that AlphaOne never obtained negative returns in backtesting.

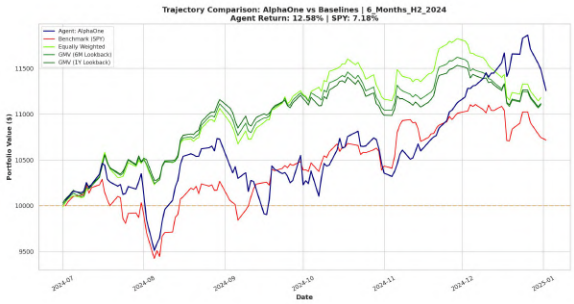
AlphaTwo's deterministic policy—Figures from B.4 to B.6—was the best performing among the Alpha family. Despite obtaining negative returns in the first half of 2025, AlphaTwo was able to keep pace with its benchmarks in the full two-years run.

AlphaThree's deterministic policy—Figures from B.7 to B.9—performed worse than AlphaTwo's but significantly better than AlphaOne's. However, AlphaThree's performance in the full two-years run was far from its benchmarks'.

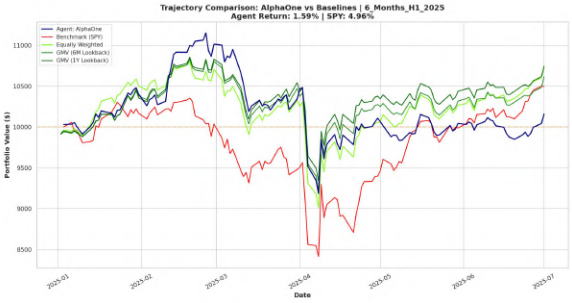
AlphaFour's deterministic policy—Figures from B.10 to B.12—is second-to-last within the Alpha family in terms of performance. Despite it performing better than AlphaOne's, it still massively underperformed its benchmarks in the full two-years-run.



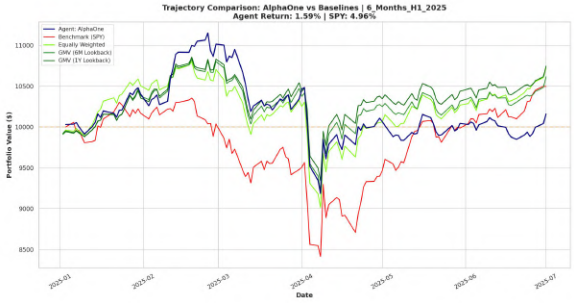
(a) H1 2024



(b) H2 2024

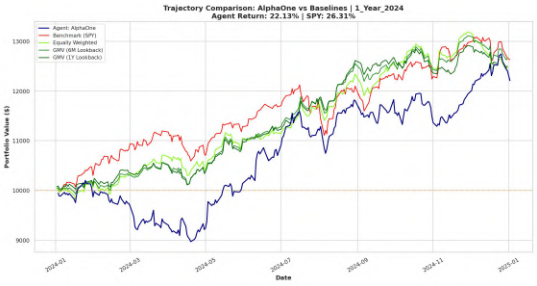


(c) H1 2025



(d) H2 2025

Figure B.1: Equity curves of AlphaOne's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024

(b) 2025

Figure B.2: Equity curves of AlphaOne's portfolio against benchmarks on a yearly basis.

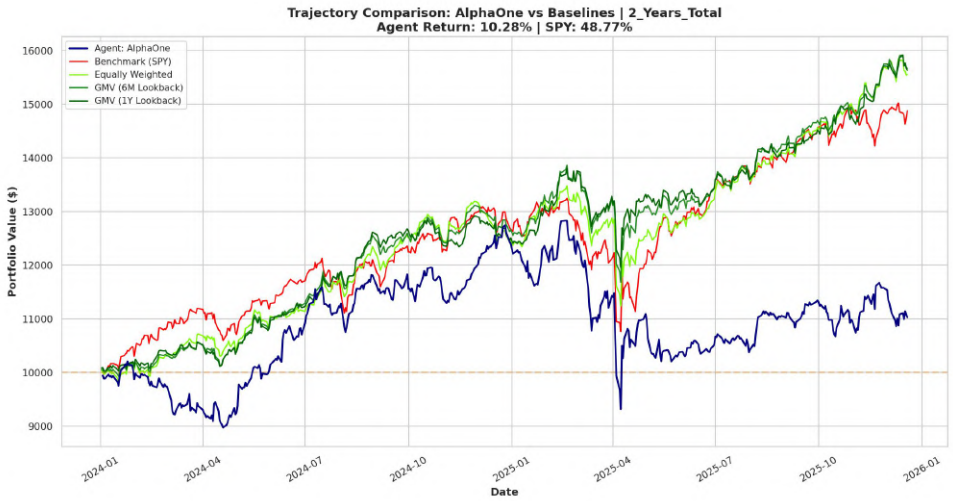
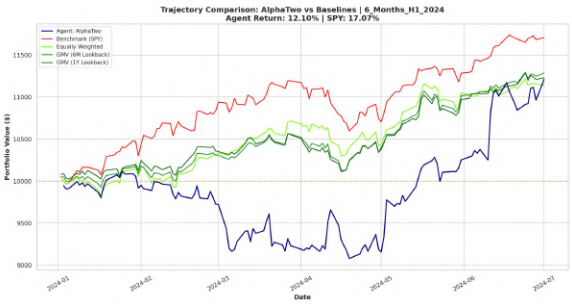
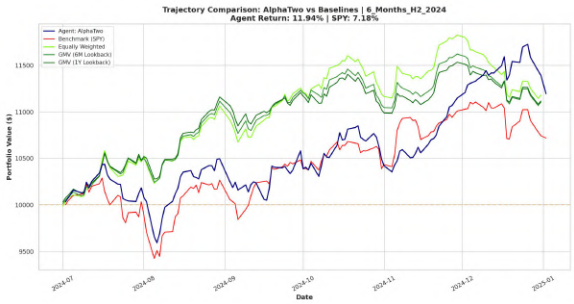


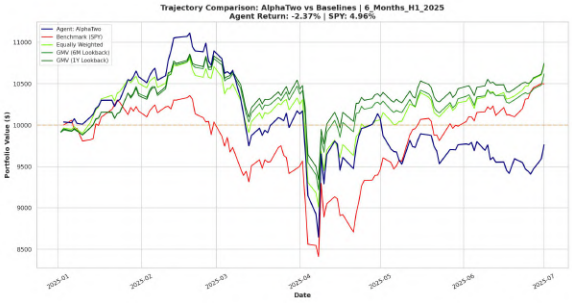
Figure B.3: Equity curve of AlphaOne's portfolio against benchmarks over the full evaluation period (2024–2025).



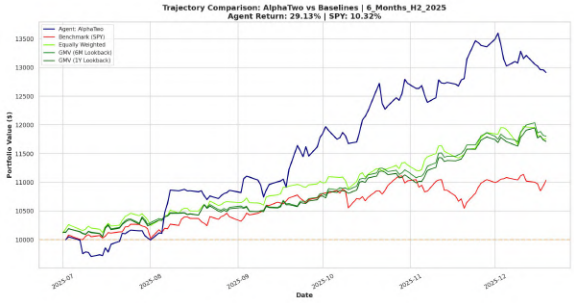
(a) H1 2024



(b) H2 2024

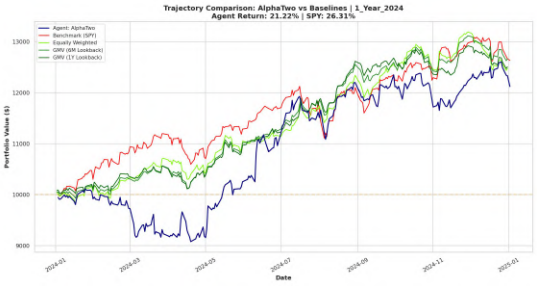


(c) H1 2025

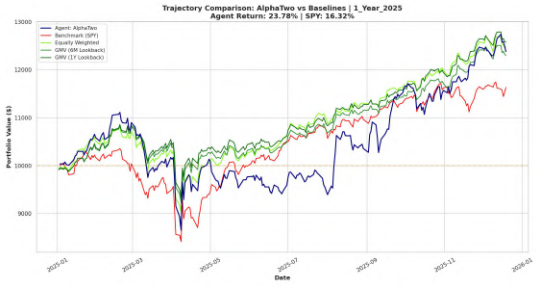


(d) H2 2025

Figure B.4: Equity curves of AlphaTwo's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.5: Equity curves of AlphaTwo's portfolio against benchmarks on a yearly basis.

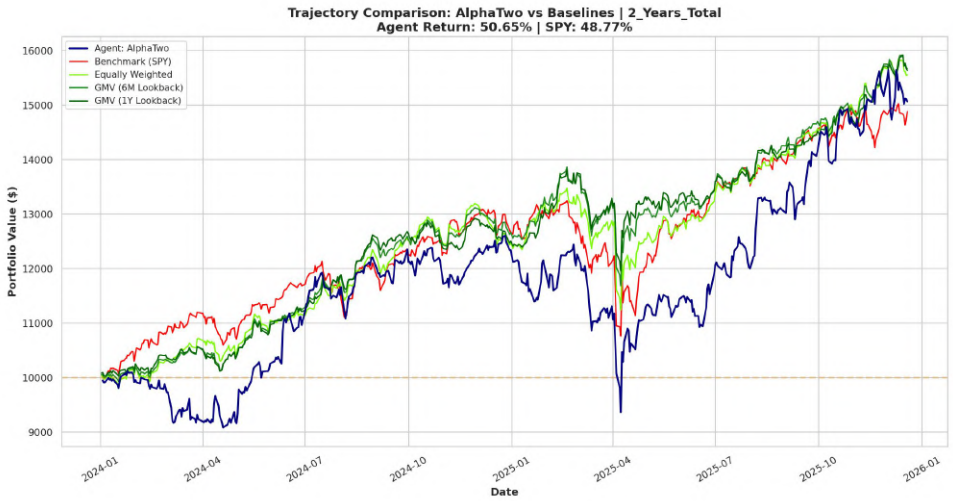
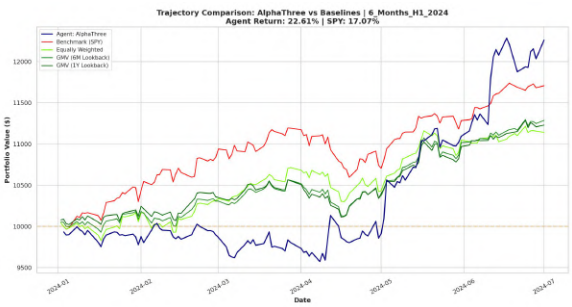
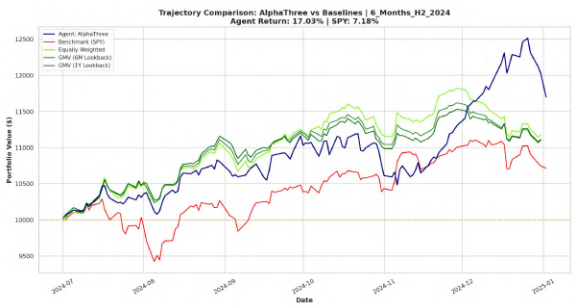


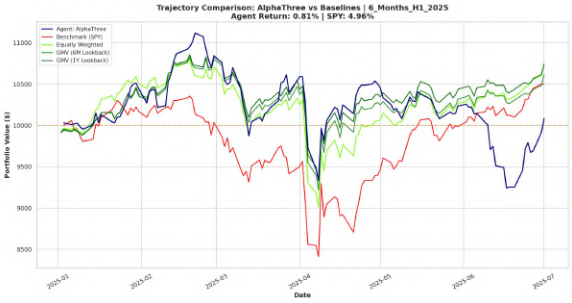
Figure B.6: Equity curve of AlphaTwo's portfolio against benchmarks over the full evaluation period (2024–2025).



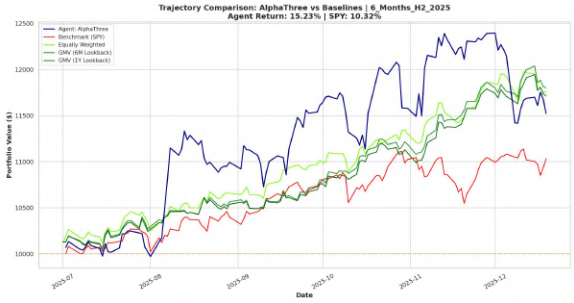
(a) H1 2024



(b) H2 2024

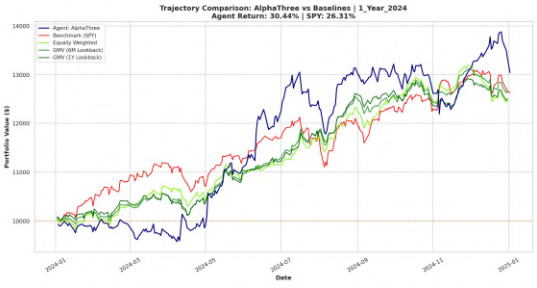


(c) H1 2025

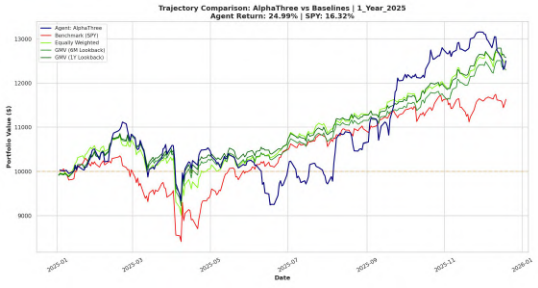


(d) H2 2025

Figure B.7: Equity curves of AlphaThree's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.8: Equity curves of AlphaThree's portfolio against benchmarks on a yearly basis.

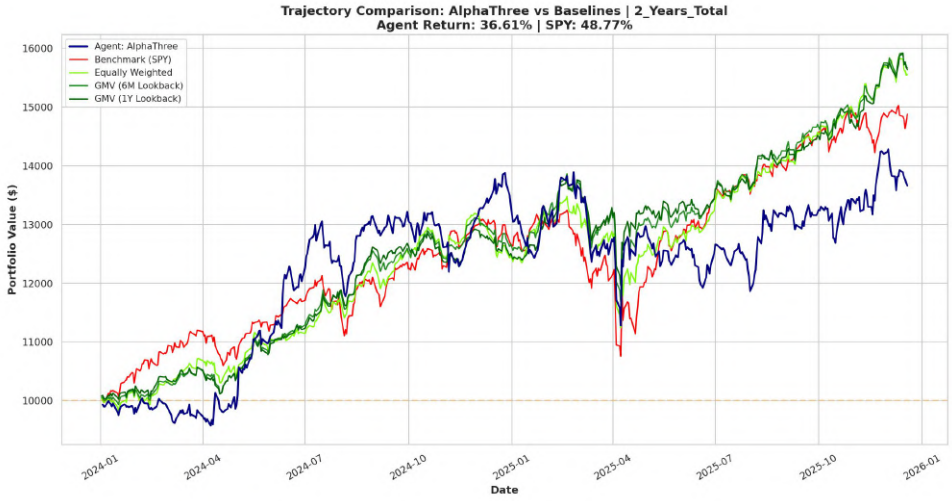
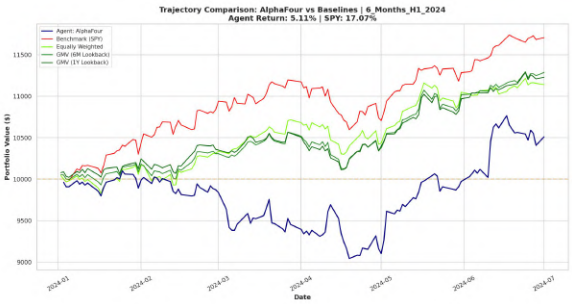
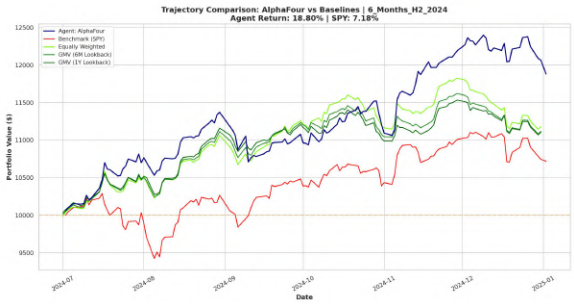


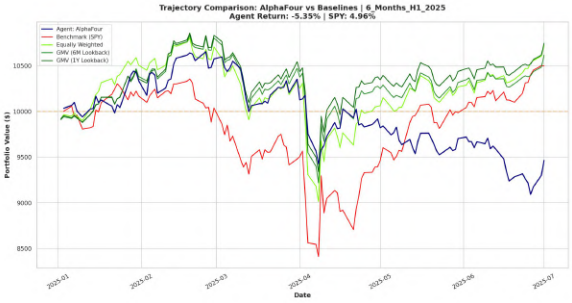
Figure B.9: Equity curve of AlphaThree's portfolio against benchmarks over the full evaluation period (2024–2025).



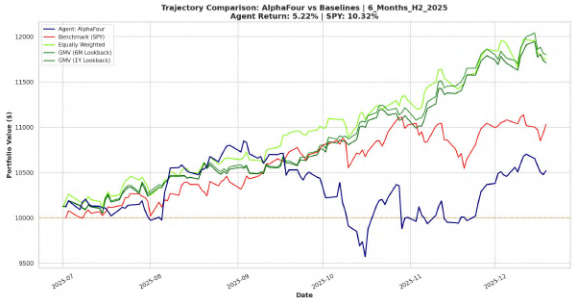
(a) H1 2024



(b) H2 2024

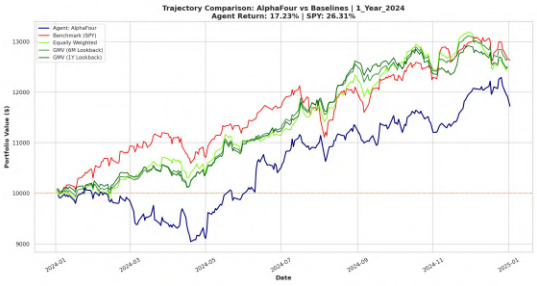


(c) H1 2025

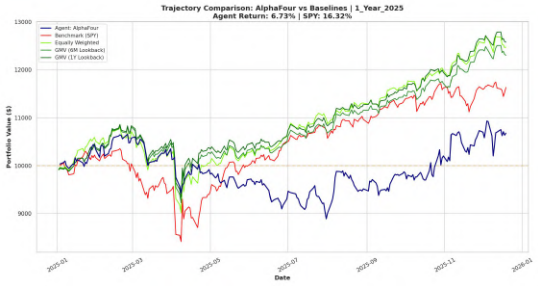


(d) H2 2025

Figure B.10: Equity curves of AlphaFour's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.11: Equity curves of AlphaFour's portfolio against benchmarks on a yearly basis.

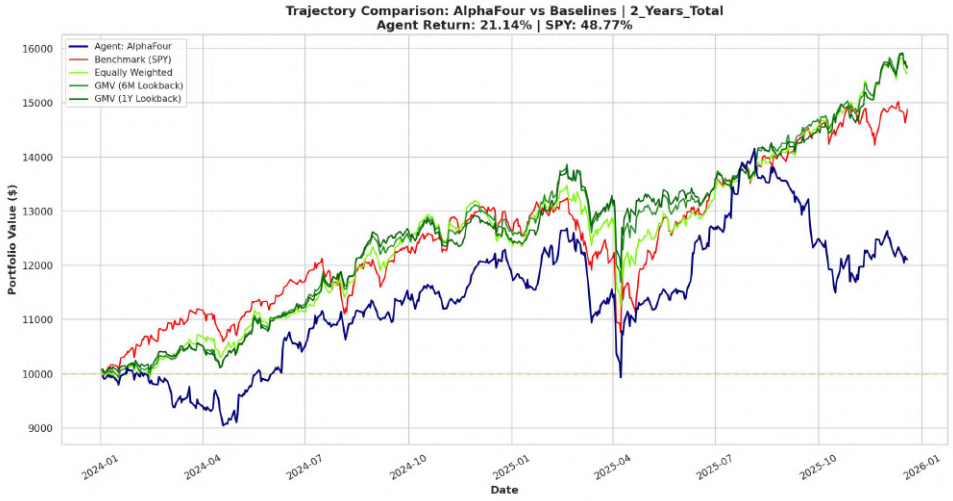


Figure B.12: Equity curve of AlphaFour's portfolio against benchmarks over the full evaluation period (2024–2025).

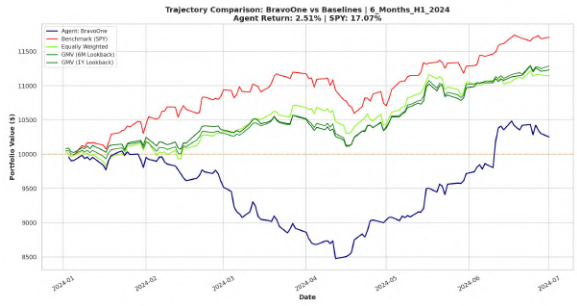
B.2 Bravo Family

BravoOne's deterministic policy—Figures from B.13 to B.15—performed very poorly, in a similar fashion to AlphaOne's. BravoOne underperformed its benchmarks in all occasions except for the first half of 2025. Furthermore, BravoOne obtained negative returns in one occasion, the second half of 2025. Overall, BravoOne returned a mere 3.38% over the full two-years run, a clearly unsatisfactory performance.

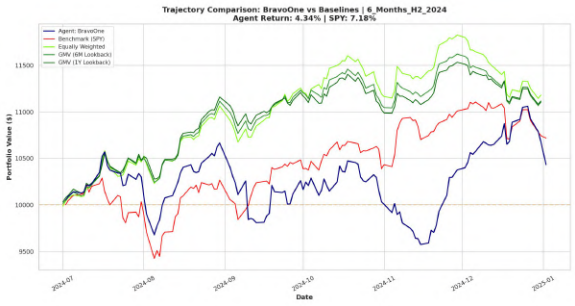
BravoTwo's deterministic policy—Figures from B.16 to B.18—performed significantly better than BravoOne's. Notably, BravoTwo was able to generate immense returns in the full two-years run, massively outperforming its benchmarks.

BravoThree's deterministic policy—Figures from B.19 to B.21—performed slightly poorer than BravoOne's. BravoThree underperformed its benchmarks in two out of four half-yearly backtesting windows. In the full two-years run, BravoThree's deterministic policy returned a mere 1.53%, a clearly unsatisfactory result.

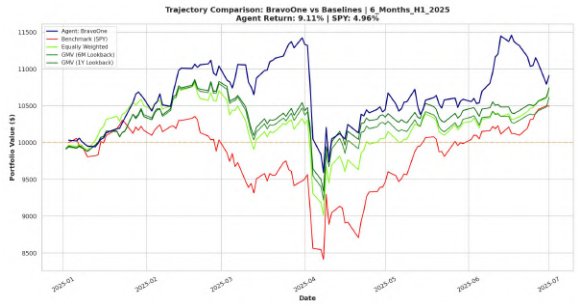
BravoFour's deterministic policy—Figures from B.22 to B.24—was the overall best performing policy among the Bravo family, outperforming even BravoTwo in the full run. However, as we'll discuss later on, both extremely positive results coming from BravoTwo and BravoFour are more a stroke of luck than the sign of a reliably learned policy.



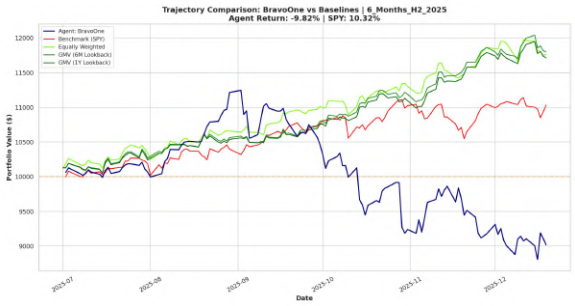
(a) H1 2024



(b) H2 2024

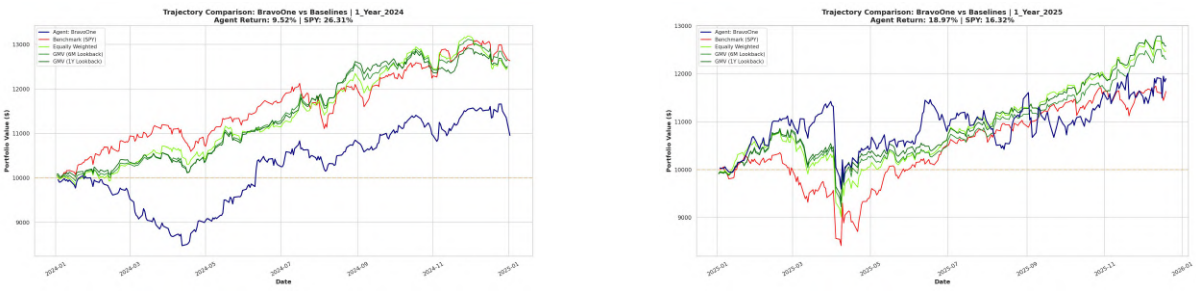


(c) H1 2025



(d) H2 2025

Figure B.13: Equity curves of BravoOne's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024 (b) 2025

Figure B.14: Equity curves of BravoOne's portfolio against benchmarks on a yearly basis.

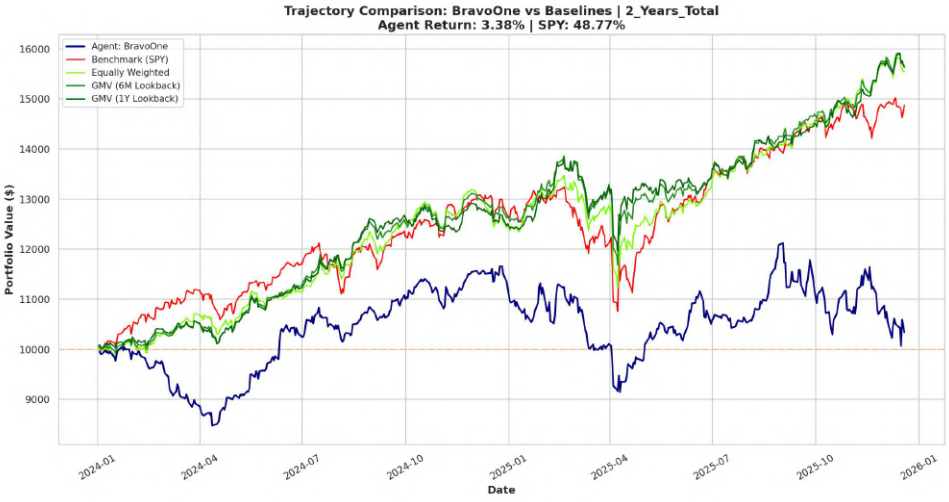
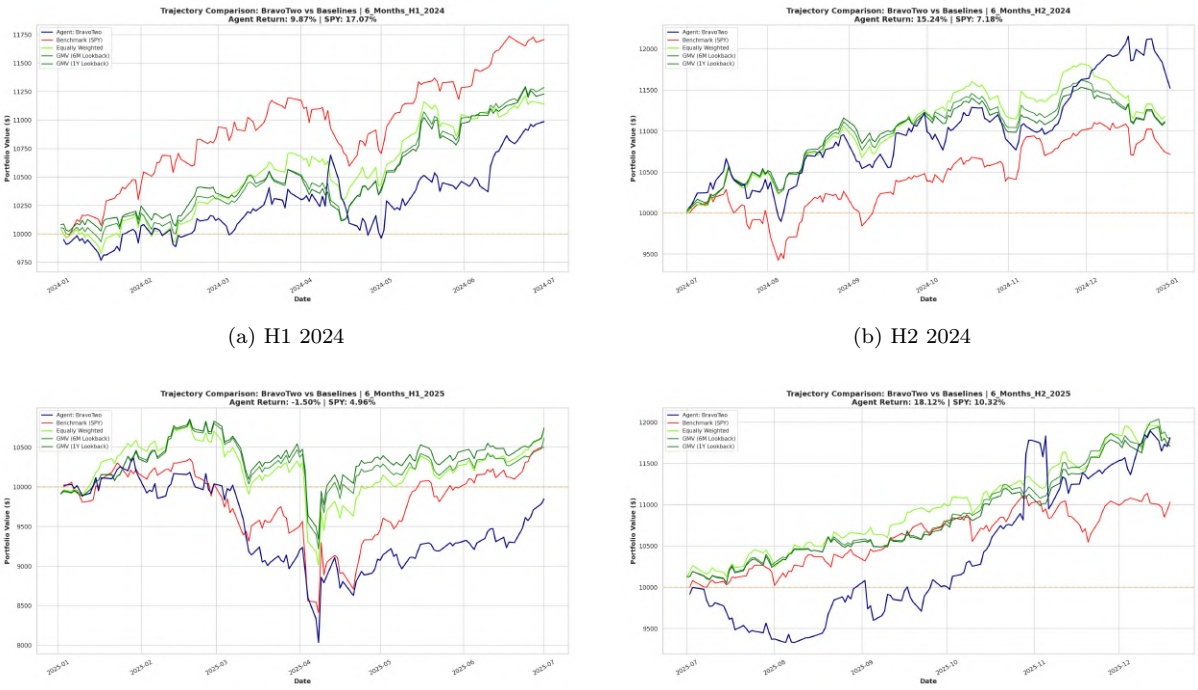


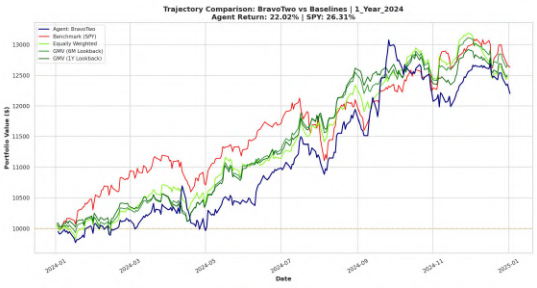
Figure B.15: Equity curve of BravoOne's portfolio against benchmarks over the full evaluation period (2024–2025).



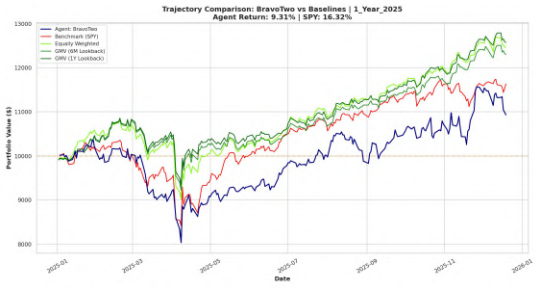
(a) H1 2024 (b) H2 2024

(c) H1 2025 (d) H2 2025

Figure B.16: Equity curves of BravoTwo's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.17: Equity curves of BravoTwo's portfolio against benchmarks on a yearly basis.

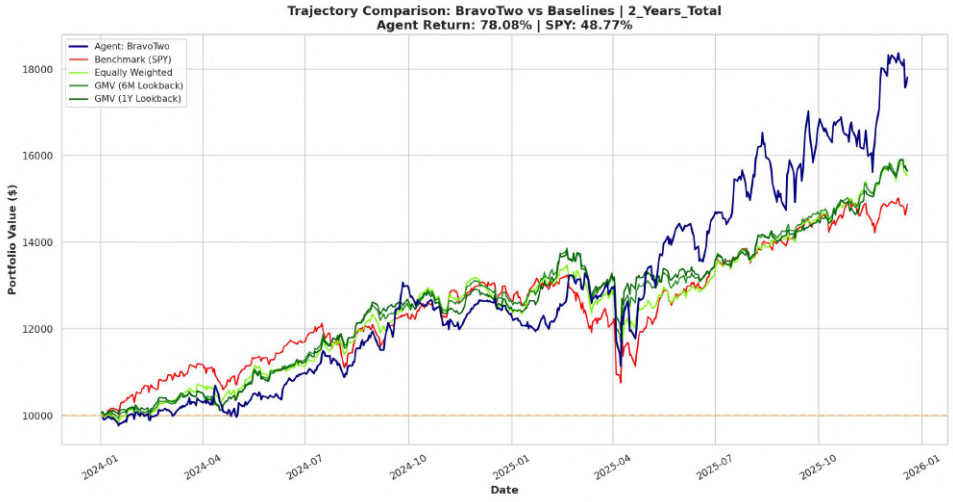
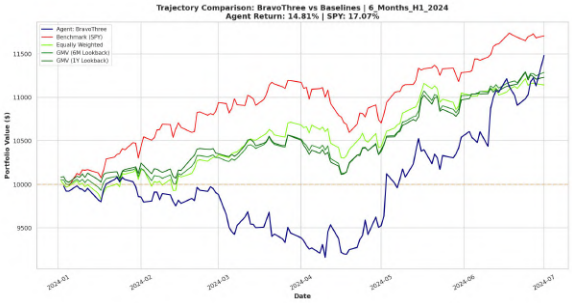
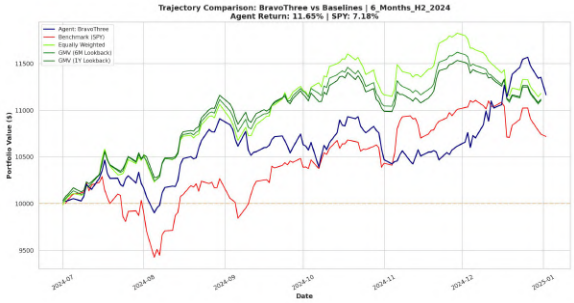


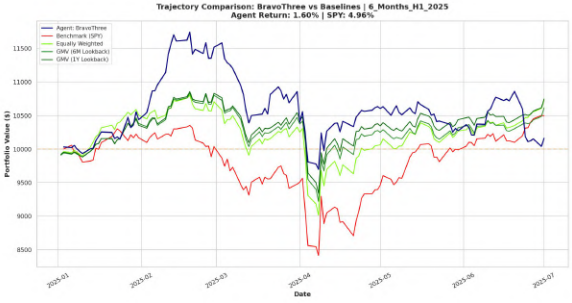
Figure B.18: Equity curve of BravoTwo's portfolio against benchmarks over the full evaluation period (2024–2025).



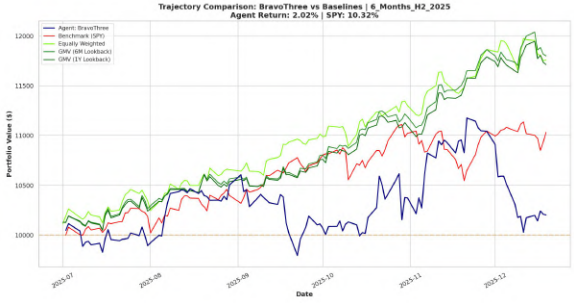
(a) H1 2024



(b) H2 2024

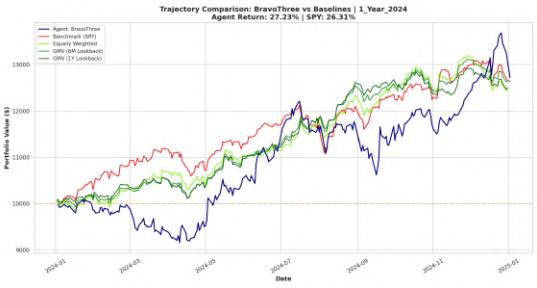


(c) H1 2025

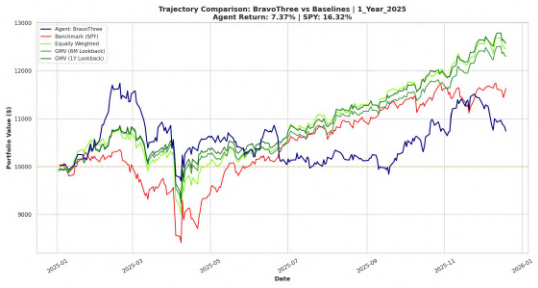


(d) H2 2025

Figure B.19: Equity curves of BravoThree's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.20: Equity curves of BravoThree's portfolio against benchmarks on a yearly basis.

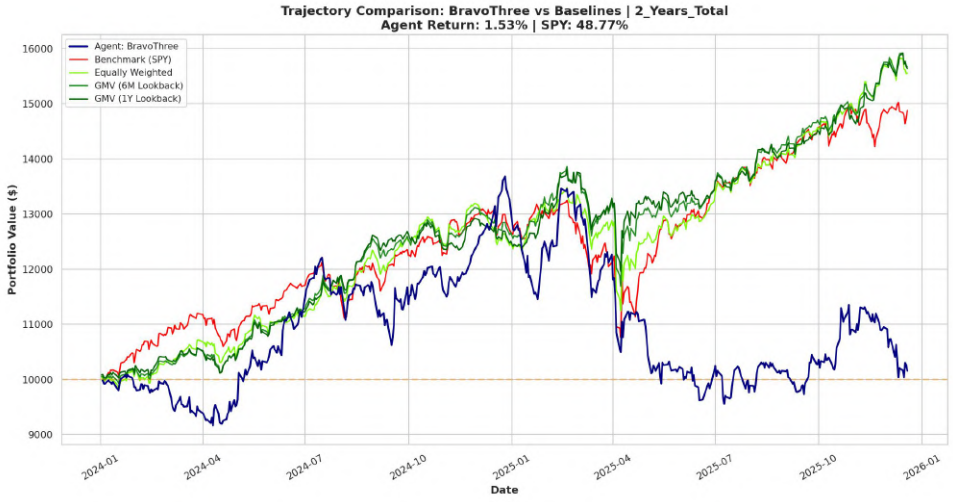
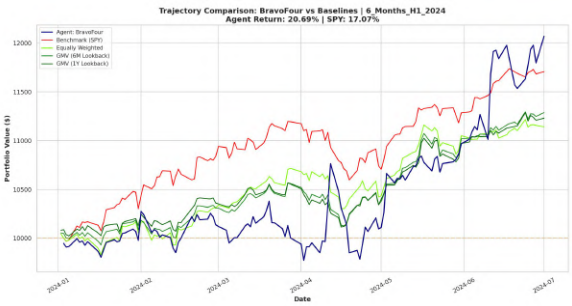
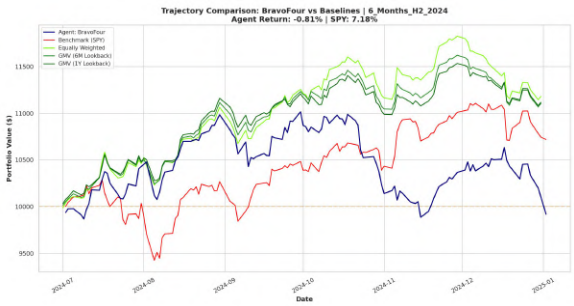


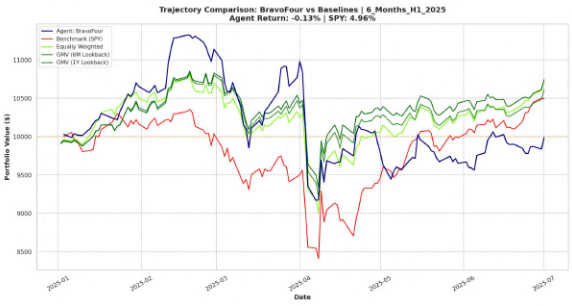
Figure B.21: Equity curve of BravoThree's portfolio against benchmarks over the full evaluation period (2024–2025).



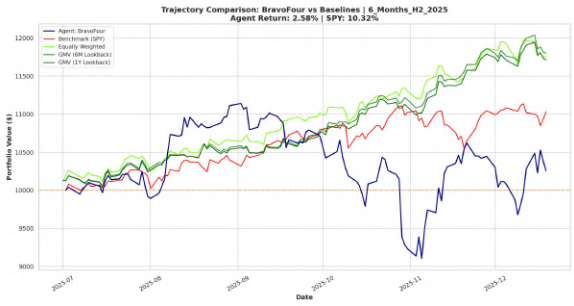
(a) H1 2024



(b) H2 2024

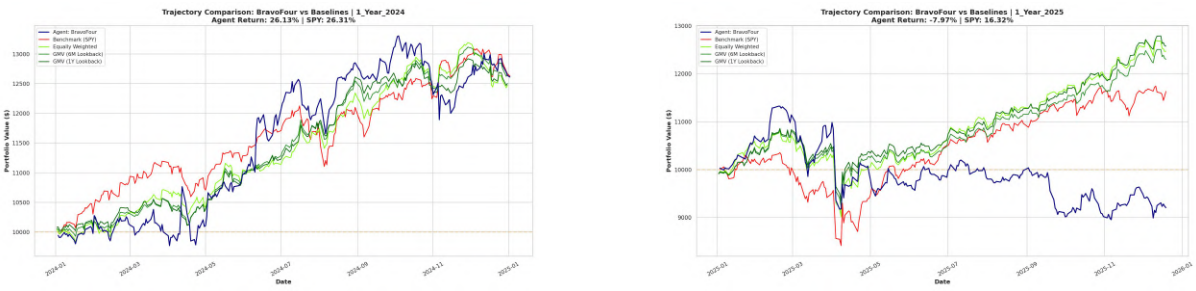


(c) H1 2025



(d) H2 2025

Figure B.22: Equity curves of BravoFour's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024 (b) 2025

Figure B.23: Equity curves of BravoFour's portfolio against benchmarks on a yearly basis.

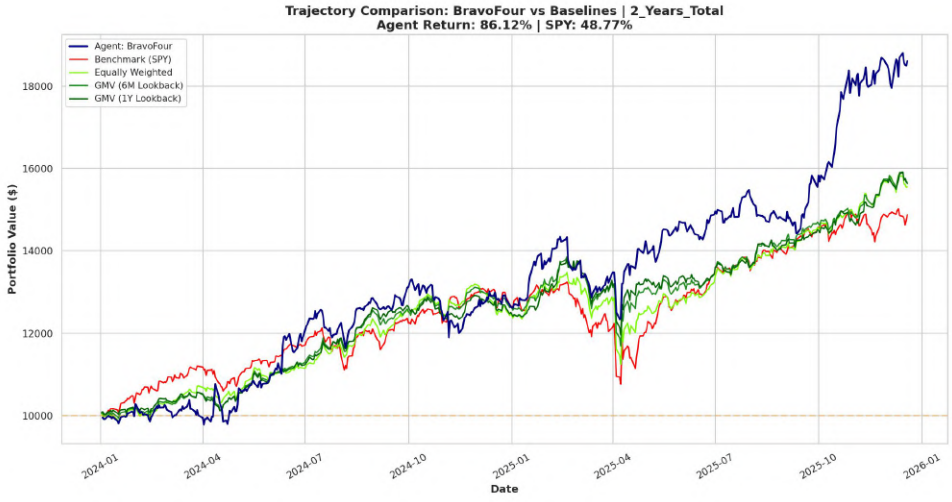


Figure B.24: Equity curve of BravoFour's portfolio against benchmarks over the full evaluation period (2024–2025).

B.3 Charlie Family

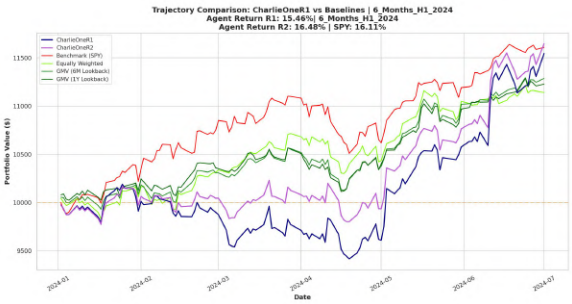
The deterministic policy learned by agent CharlieOne—Figures from B.25 to B.27—was able to outperform reliably its benchmarks. CharlieOne was able to outperform by a significant margin its benchmarks in three out of four half year backtesting windows. Furthermore, it outperformed its benchmarks in 2024, and performed similarly in 2025. Over the full two years run, CharlieOne was able to outperform all of its benchmarks. Furthermore, one may observe that the R1 trained agent outperformed the R2 trained agent in most windows, including the full two-years run. The results obtained by the first PPO wide agent are remarkable, especially considering that all its benchmarks are not affected by transaction costs.

The deterministic policy learned by CharlieTwo—Figures from B.28 to B.30—performed worse than CharlieOne's, outperforming its benchmarks only in the second half of 2025—which appears to be a particularly proficuous window for all Charlie agents. However, when looking at yearly and full run returns, CharlieTwo performed in line with its benchmarks. Nevertheless, this is still a remarkable achievement, considering that its benchmarks do not consider transaction costs. Once again, the R1 agent outperform its R2 twin in most occasions.

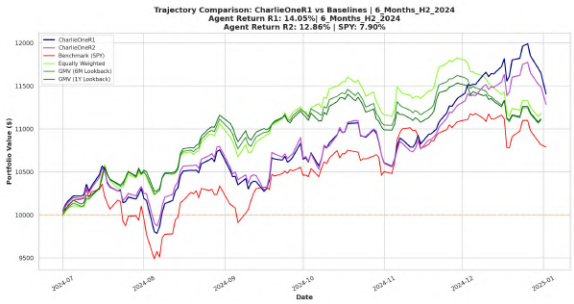
It seems that the added complexity stemming from a longer look-back window does not improve agent performance, instead leading to confusion and lower returns.

CharlieThree's deterministic policy—Figures from B.31 to B.33—performed worse than CharlieOne's and Two's. Still, agent performance is not bad overall—it is generating positive returns after all, but performed worse, or in line at best, than its benchmarks in multiple occasions. CharlieThree represents the only instance among wide PPO agents of the R2 agent outperforming its twin in most occasions. Nevertheless, R1 and R2 performances are quite similar between one another.

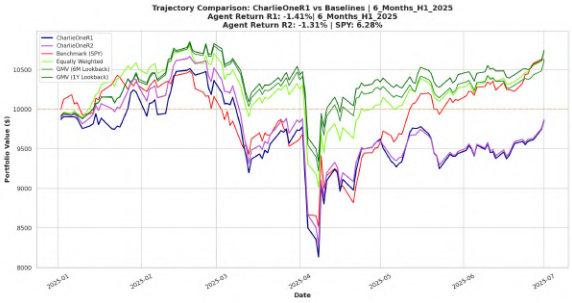
CharlieFour's policy—Figures from B.34 to B.36—is the second best performing among the Charlie family, when trained using the IRR reward function, and the worst performing under Sharpe reward. This difference in behavior, completely dependent on the reward function employed in training, presents evidence in favor of our hypothesis of the Sharpe ratio being unsuitable to be used as a reward function for view forming agents.



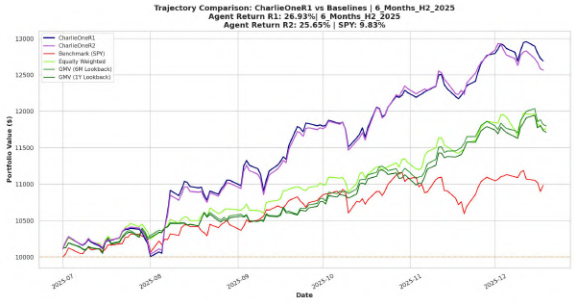
(a) H1 2024



(b) H2 2024

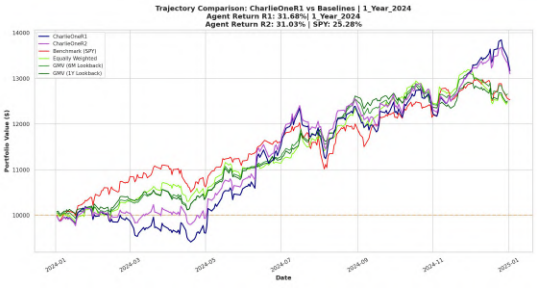


(c) H1 2025

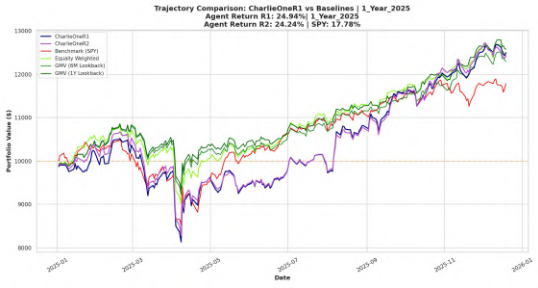


(d) H2 2025

Figure B.25: Equity curves of CharlieOne's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.26: Equity curves of CharlieOne's portfolio against benchmarks on a yearly basis.

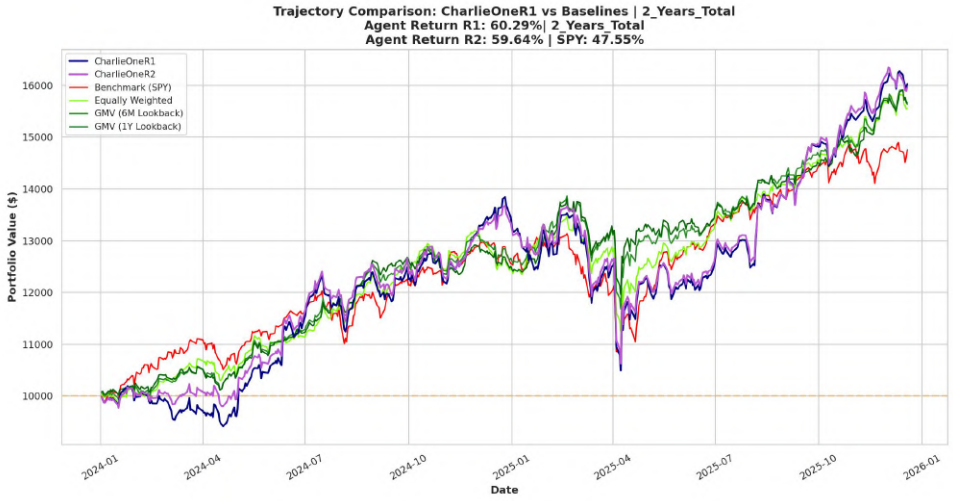
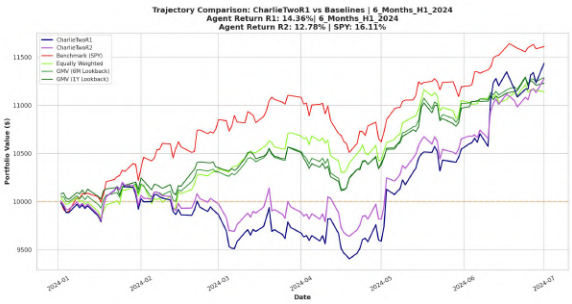
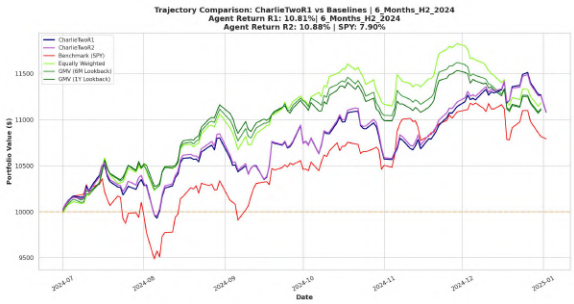


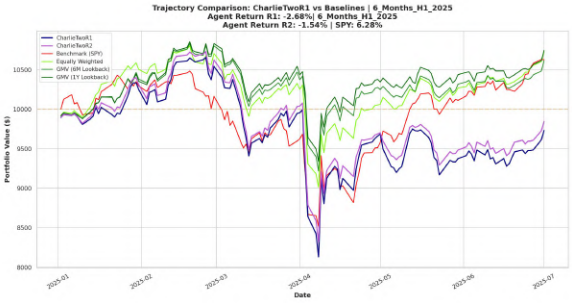
Figure B.27: Equity curve of CharlieOne's portfolio against benchmarks over the full evaluation period (2024–2025).



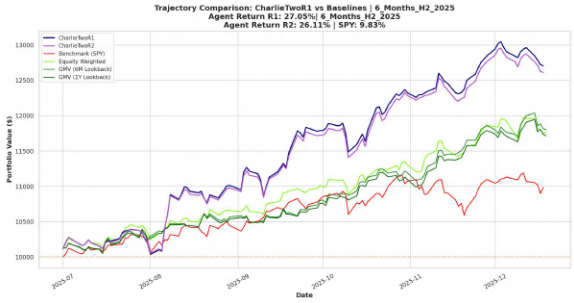
(a) H1 2024



(b) H2 2024

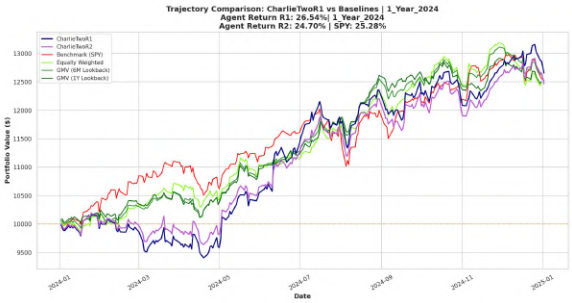


(c) H1 2025

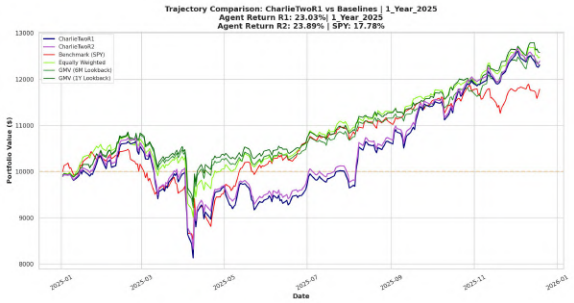


(d) H2 2025

Figure B.28: Equity curves of CharlieTwo's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.29: Equity curves of CharlieTwo's portfolio against benchmarks on a yearly basis.

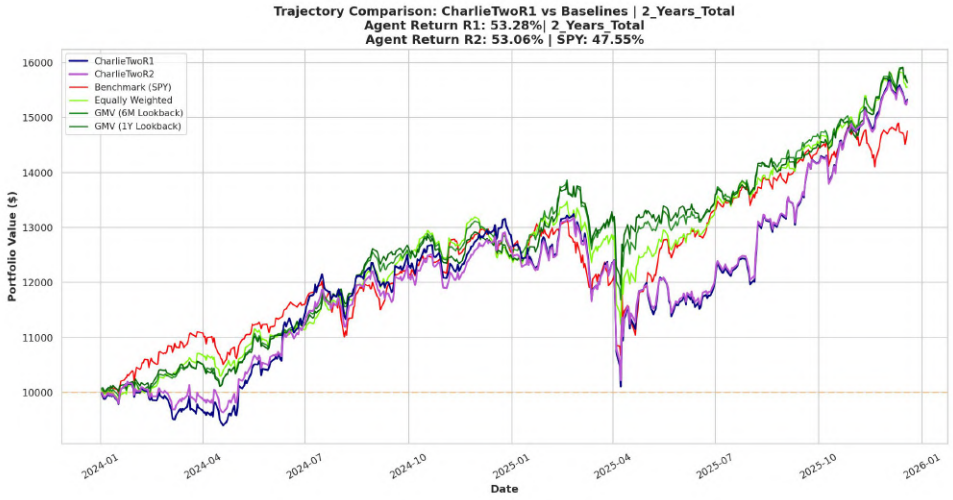
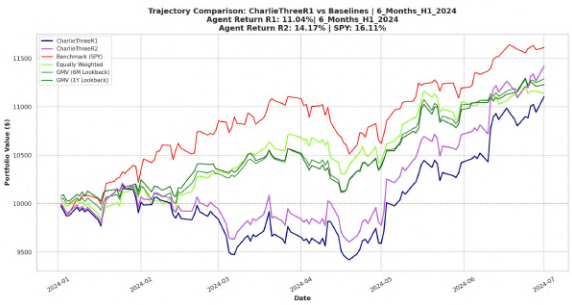
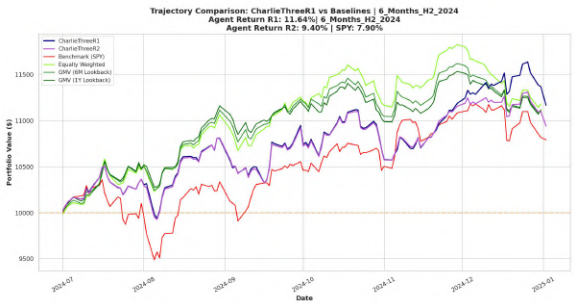


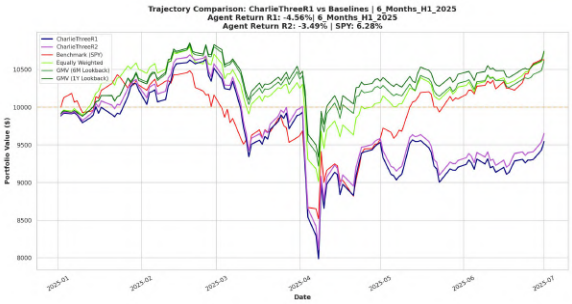
Figure B.30: Equity curve of CharlieTwo's portfolio against benchmarks over the full evaluation period (2024–2025).



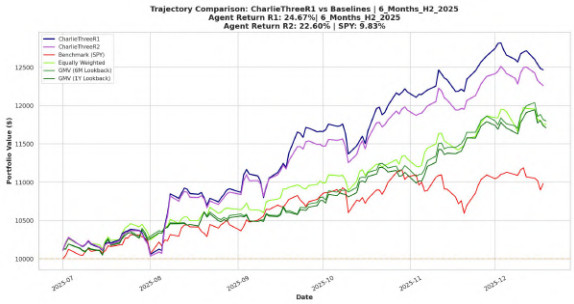
(a) H1 2024



(b) H2 2024

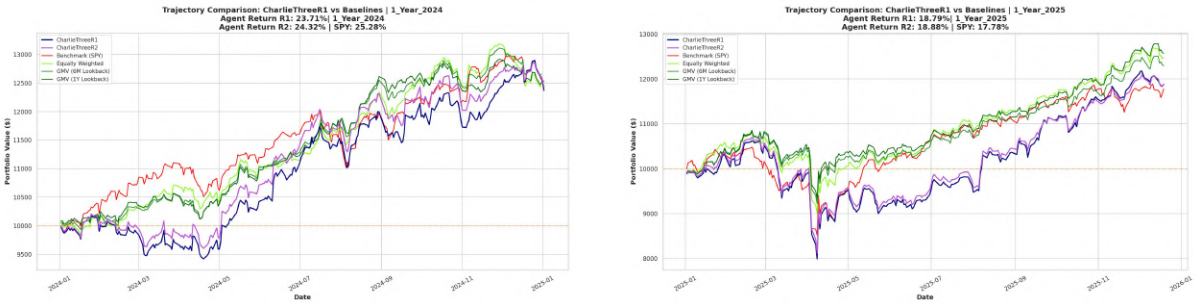


(c) H1 2025



(d) H2 2025

Figure B.31: Equity curves of CharlieThree's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024 (b) 2025

Figure B.32: Equity curves of CharlieThree's portfolio against benchmarks on a yearly basis.

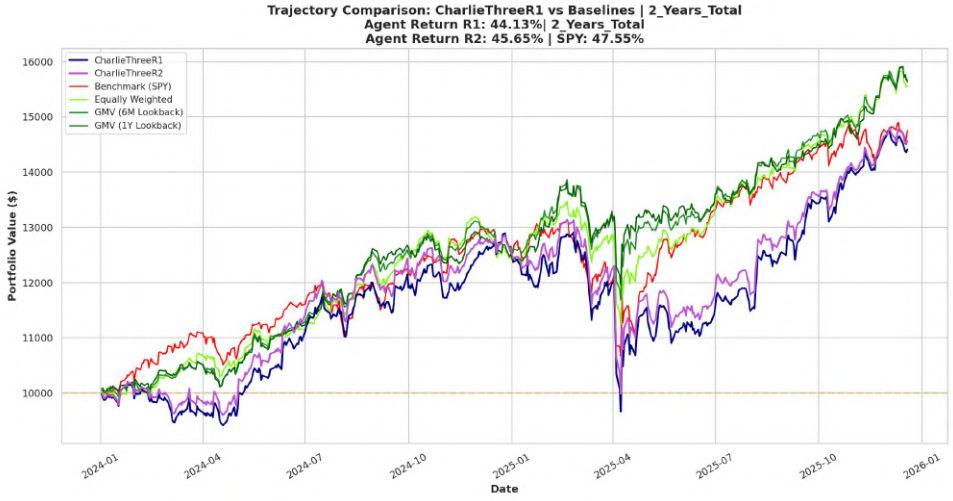
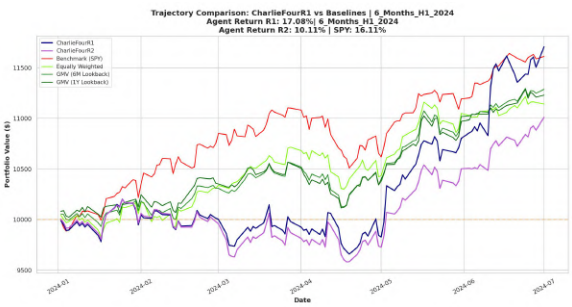
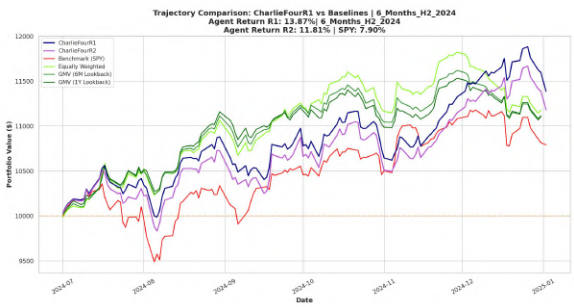


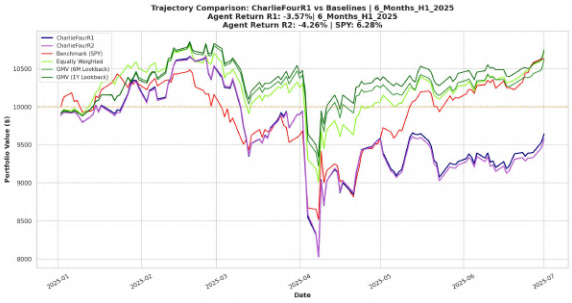
Figure B.33: Equity curve of CharlieThree's portfolio against benchmarks over the full evaluation period (2024–2025).



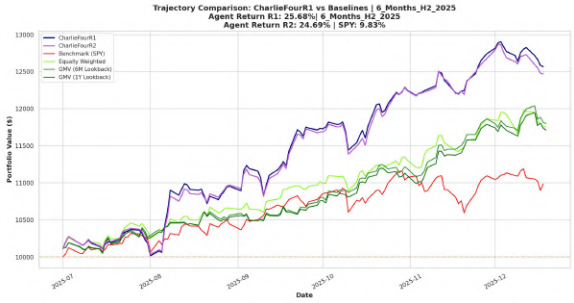
(a) H1 2024



(b) H2 2024

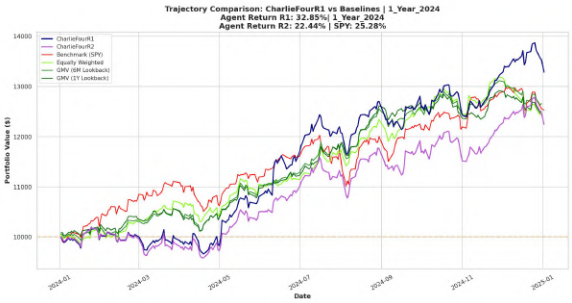


(c) H1 2025

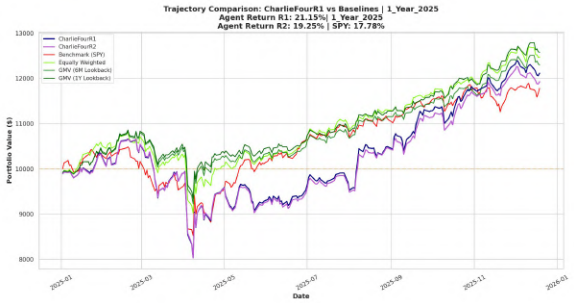


(d) H2 2025

Figure B.34: Equity curves of CharlieFour's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.35: Equity curves of CharlieFour's portfolio against benchmarks on a yearly basis.

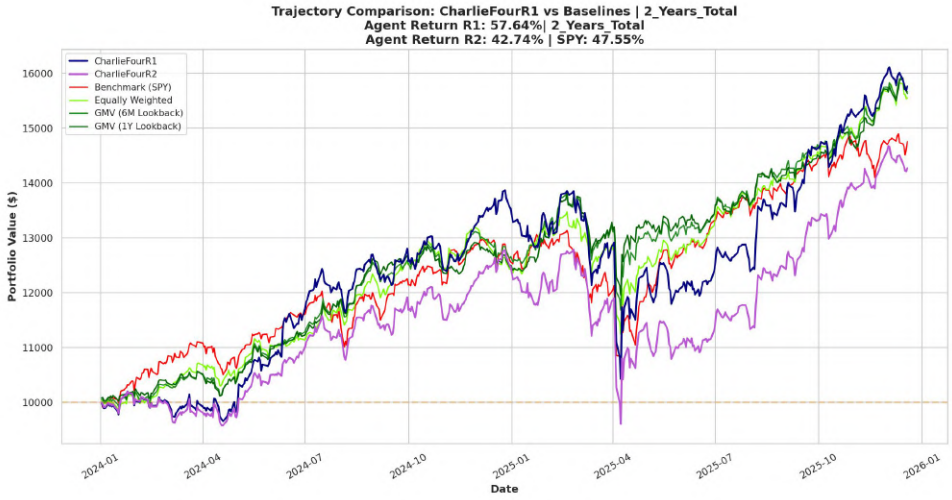


Figure B.36: Equity curve of CharlieFour's portfolio against benchmarks over the full evaluation period (2024–2025).

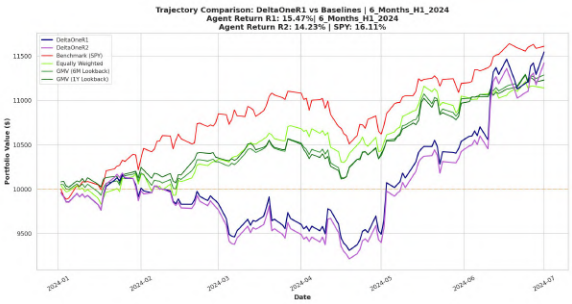
B.4 Delta Family

One can observe that DeltaOne's deterministic policy—Figures from B.37 to B.39—was the strongest among R1 Delta agents and the weakest among R2 agents. The R1 policy was the second best performing among the entire Delta family, being able to outperform its benchmarks in most occasions. Conversely, the R2 policy struggled to achieve the same performances of its R1 twin, performing worse, or at best in line, than its benchmarks.

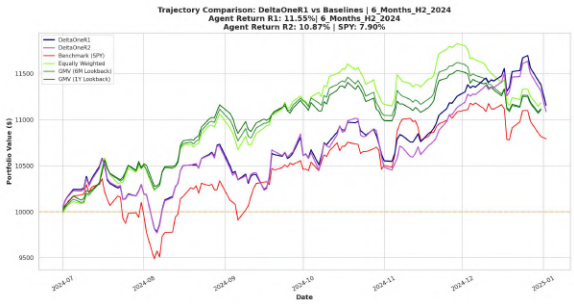
DeltaTwo's performances are unremarkable. Its deterministic strategy trained under Sharpe reward—Figures from B.40 to B.42—was able to outperform is benchmarks in a few occasions—notably in H1 2024, H2 2025, and in 2024—but it performed in line with its benchmarks in the long two-years window. The deterministic strategy emerging from the IRR reward function performed slightly worse than the R2 twin.

The R2 version of DeltaThree was, by far, the best performing agent from the Delta family. Despite under-performing its benchmarks in the first half of 2025, which has been proven to be a tough window for all wide agents, and performing in line with benchmarks in the second half of 2024, its deterministic strategy—Figures from B.43 to B.45—was able to achieve a return of 66% over the full two years window. Conversely, its twin's performances are unremarkable.

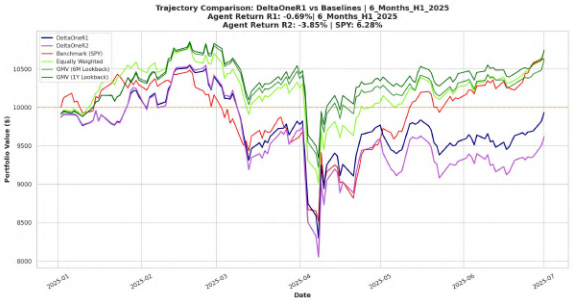
When trained under Sharpe reward, DeltaFour was the third best performing agent among the Delta family—Figures from B.46 to B.48. Conversely, it proven to be the weakest performing agent when trained using the IRR reward function.



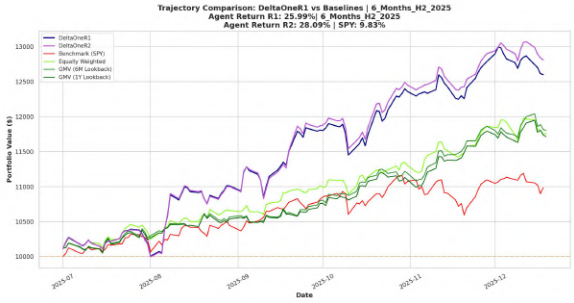
(a) H1 2024



(b) H2 2024

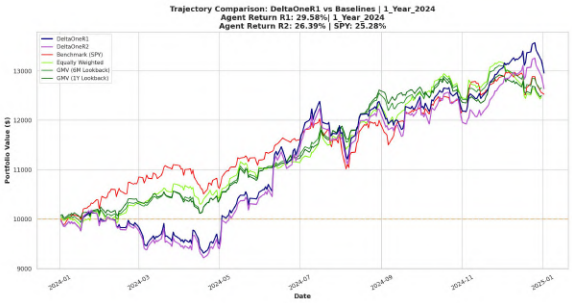


(c) H1 2025

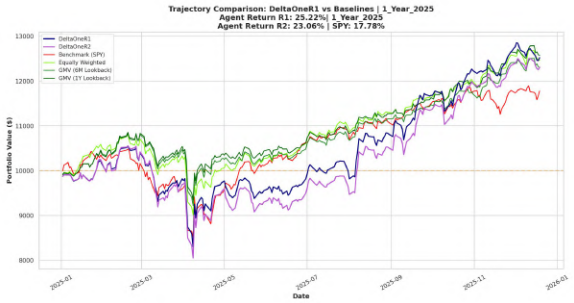


(d) H2 2025

Figure B.37: Equity curves of DeltaOne's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.38: Equity curves of DeltaOne's portfolio against benchmarks on a yearly basis.

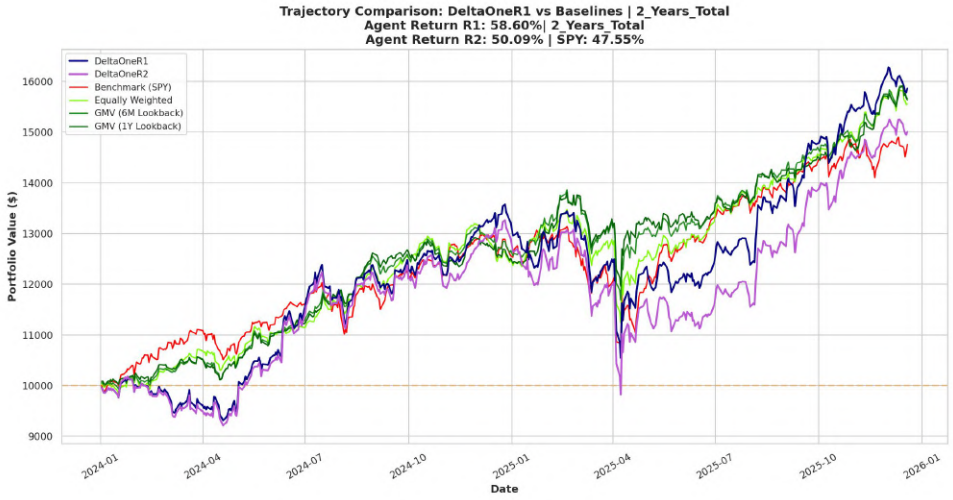
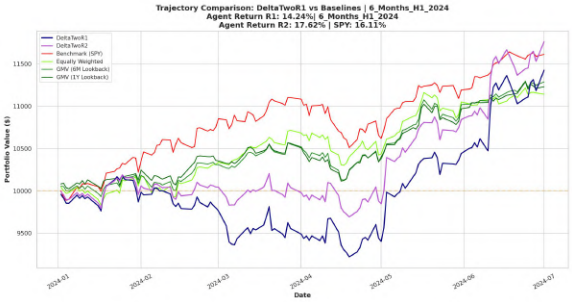
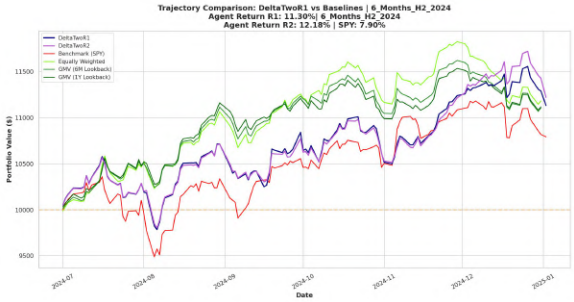


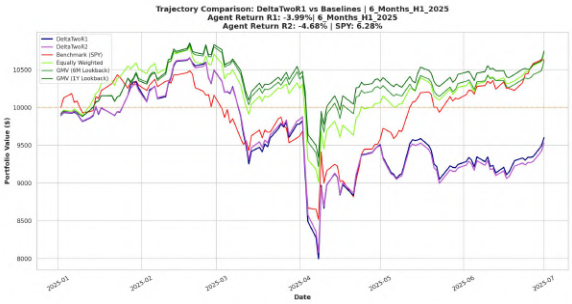
Figure B.39: Equity curve of DeltaOne's portfolio against benchmarks over the full evaluation period (2024–2025).



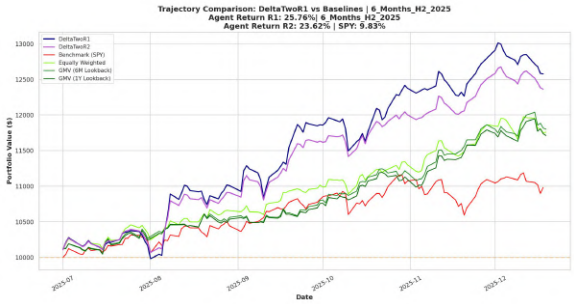
(a) H1 2024



(b) H2 2024

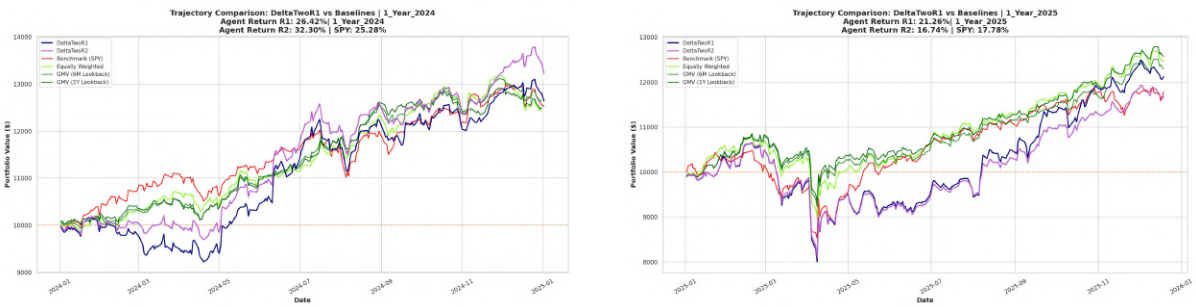


(c) H1 2025



(d) H2 2025

Figure B.40: Equity curves of DeltaTwo's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024 (b) 2025

Figure B.41: Equity curves of DeltaTwo's portfolio against benchmarks on a yearly basis.

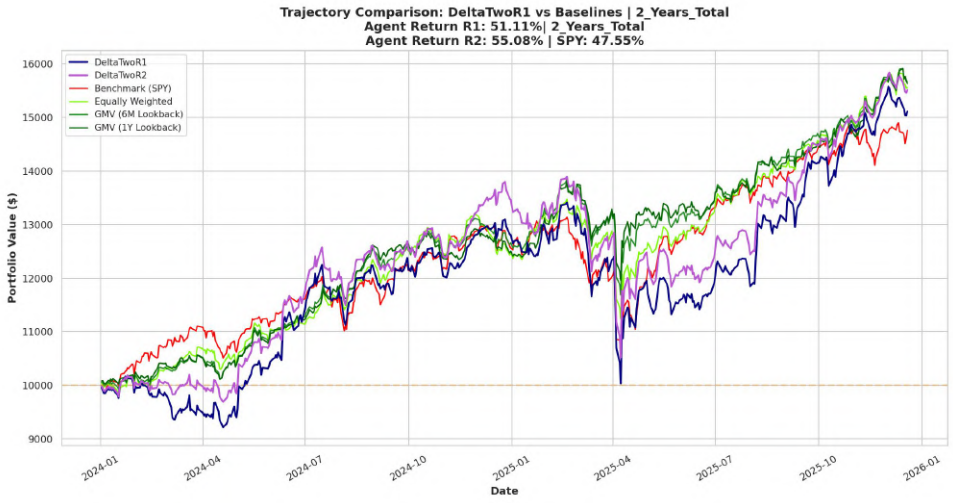
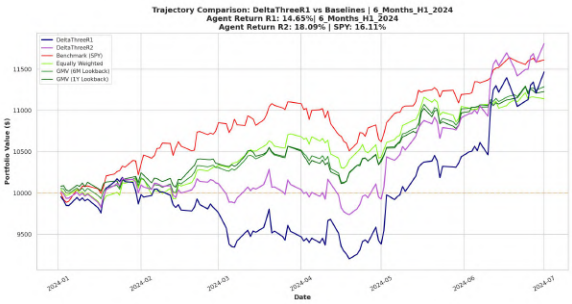
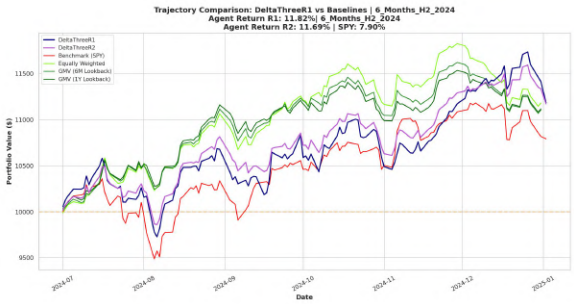


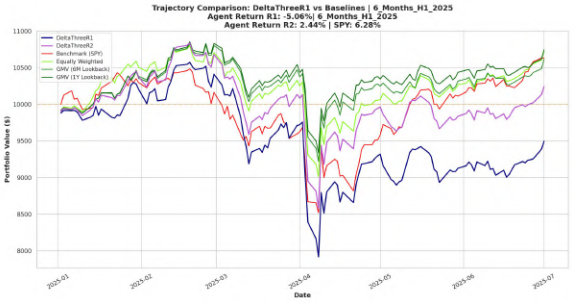
Figure B.42: Equity curve of DeltaTwo's portfolio against benchmarks over the full evaluation period (2024–2025).



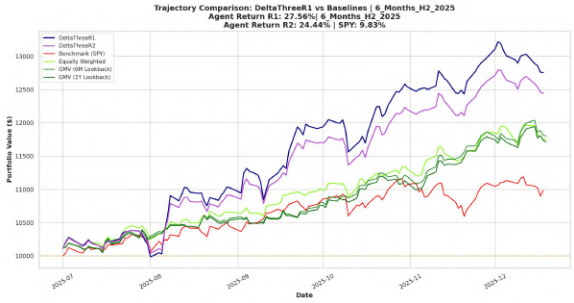
(a) H1 2024



(b) H2 2024

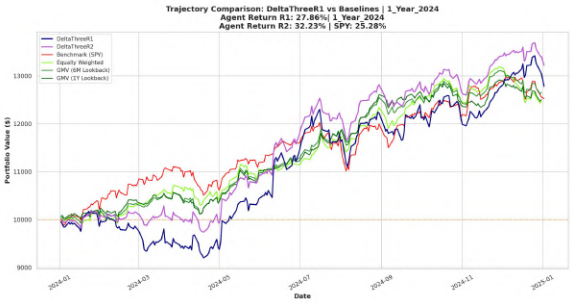


(c) H1 2025

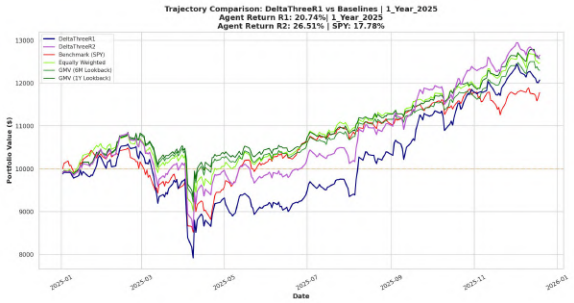


(d) H2 2025

Figure B.43: Equity curves of DeltaThree's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024



(b) 2025

Figure B.44: Equity curves of DeltaThree's portfolio against benchmarks on a yearly basis.

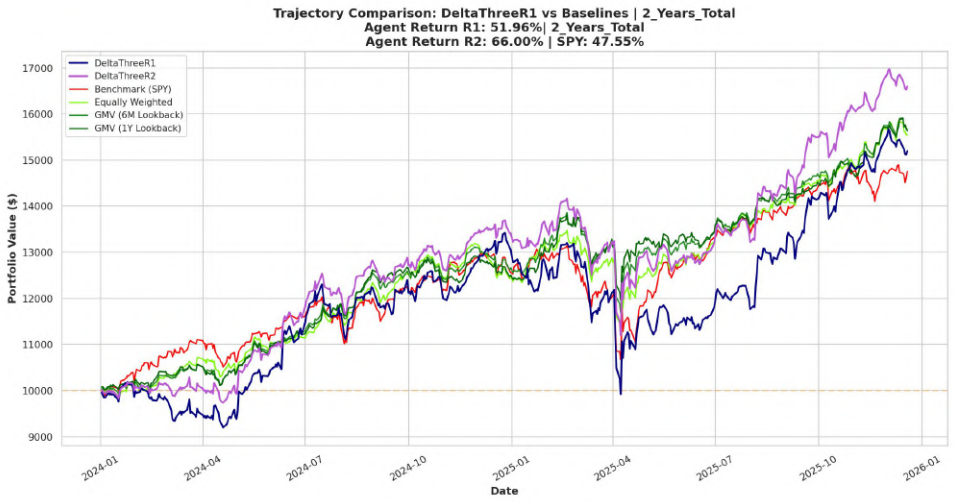
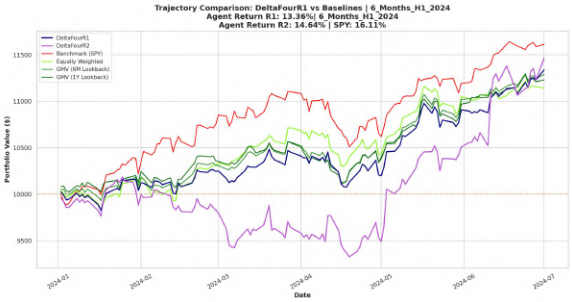
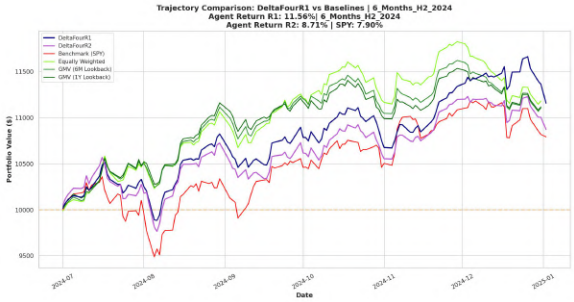


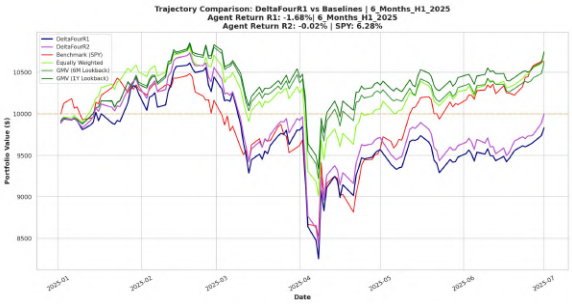
Figure B.45: Equity curve of DeltaThree's portfolio against benchmarks over the full evaluation period (2024–2025).



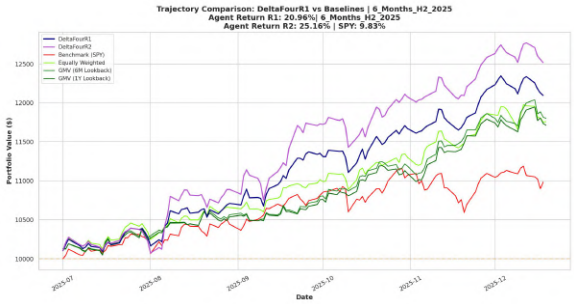
(a) H1 2024



(b) H2 2024

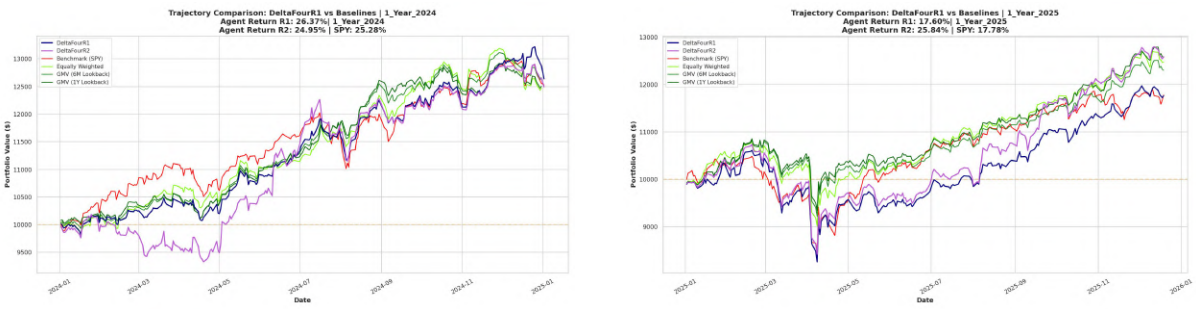


(c) H1 2025



(d) H2 2025

Figure B.46: Equity curves of DeltaFour's portfolio against benchmarks across semi-annual periods (2024–2025).



(a) 2024 (b) 2025

Figure B.47: Equity curves of DeltaFour's portfolio against benchmarks on a yearly basis.

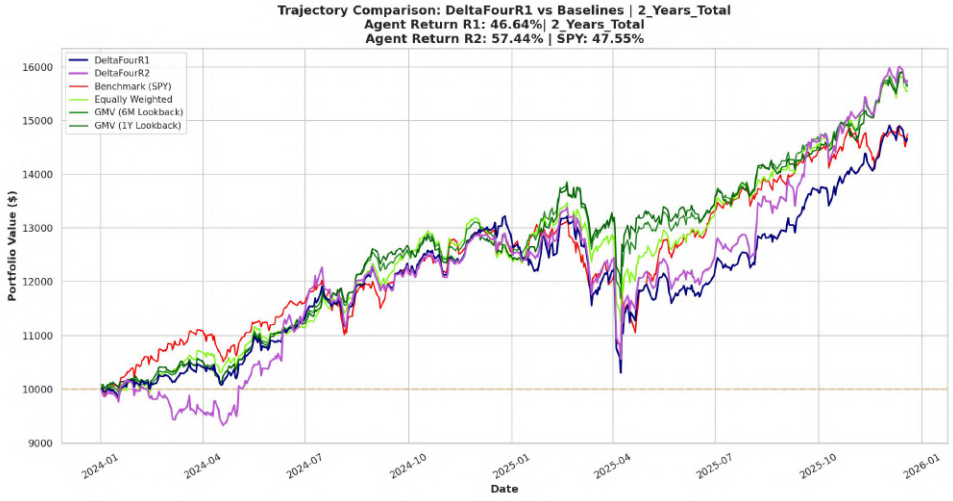


Figure B.48: Equity curve of DeltaFour's portfolio against benchmarks over the full evaluation period (2024–2025).

Appendix C

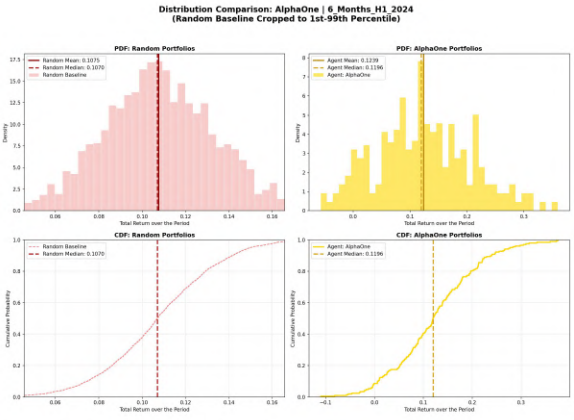
Stochastic Runs vs Randomness

C.1 Alpha Family

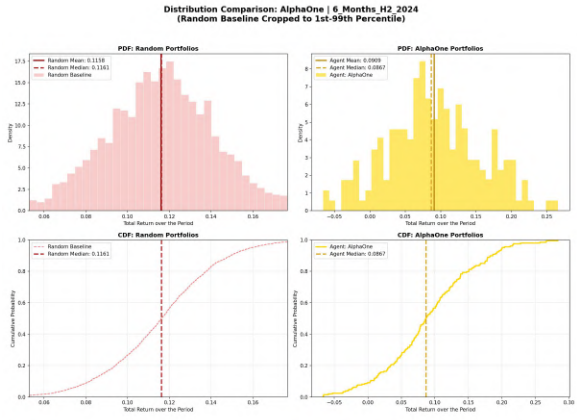
Tables 4.6 and 4.7, presented in Section 4.2.2, provide the reader with the results of the stochastic dominance tests—Kolmogorov-Smirnov (KS) and Mann-Whitney U tests (MW)—conducted on the CDFs of stochastic agent returns against the random baseline portfolios. Here, we discuss the test results and provide with the distribution plots for our runs.

From the reported tables one can observe that agents' returns were never stochastically dominant over the random baseline. Furthermore, apart for the H1 2024 backtesting window, the CDFs of stochastically simulated agents' returns were never found to be stochastically greater than the random baselines they got compared to.

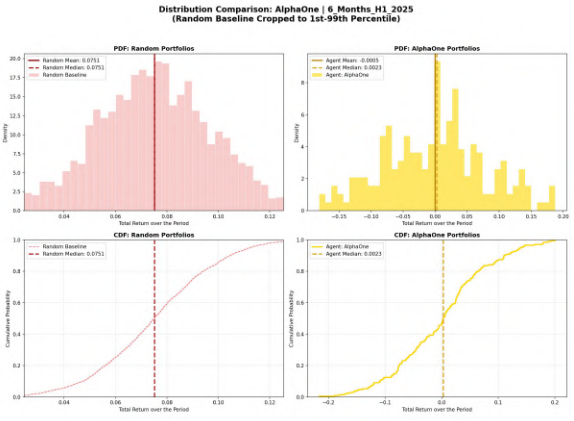
H1 2024 presents a peculiar circumstance, which we will encounter in more than one occasion throughout the analysis of our results. As mentioned, agents' returns were deemed stochastically greater, but not dominant, than random returns. This happens for one specific reason: FSD is a significantly more stringent condition than stochastic greatness. Under our hypothesis framework, presented in Section 4.2.2, FSD is violated the moment the agents' CFD crosses above the random baseline's. In contrast, the MW test is concerned with the aggregate probability mass. It allows for localized intersections (crossings) of the CDFs, provided that the overall probability of a random agent return outperforming a random baseline return remains significantly above 0.5.



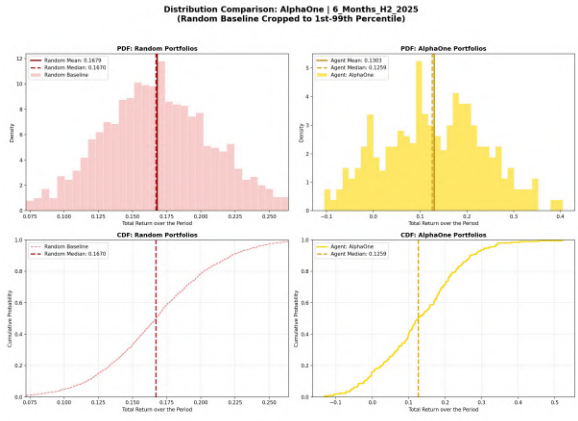
(a) H1 2024



(b) H2 2024

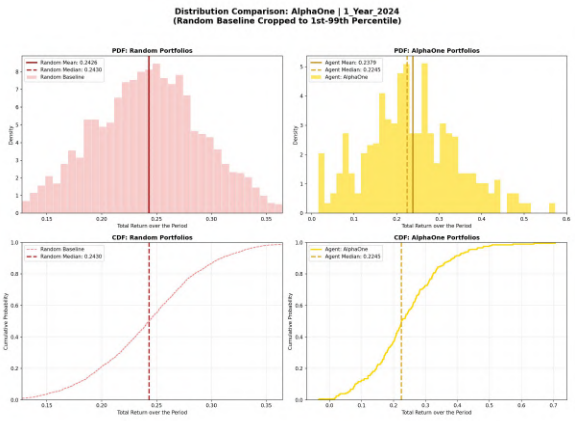


(c) H1 2025

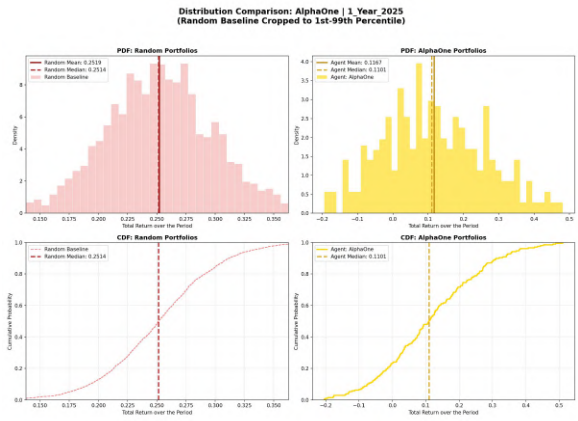


(d) H2 2025

Figure C.1: Returns PDFs and CDFs for AlphaOne's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.2: Returns PDFs and CDFs for AlphaOne's stochastic runs vs random baseline - full years.

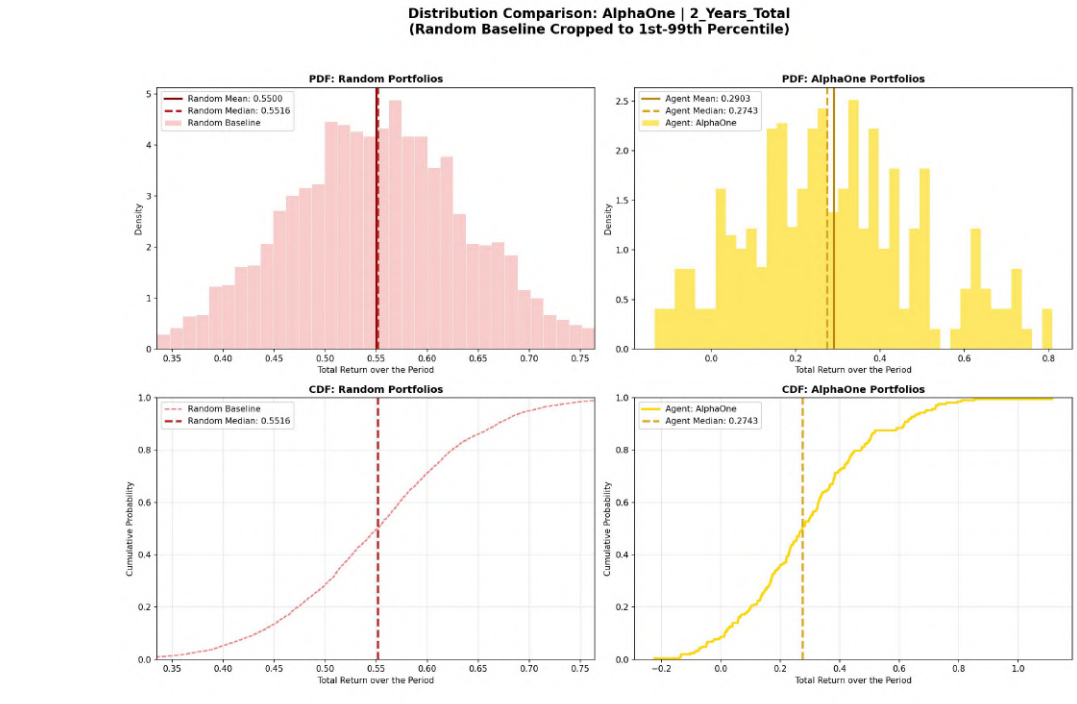
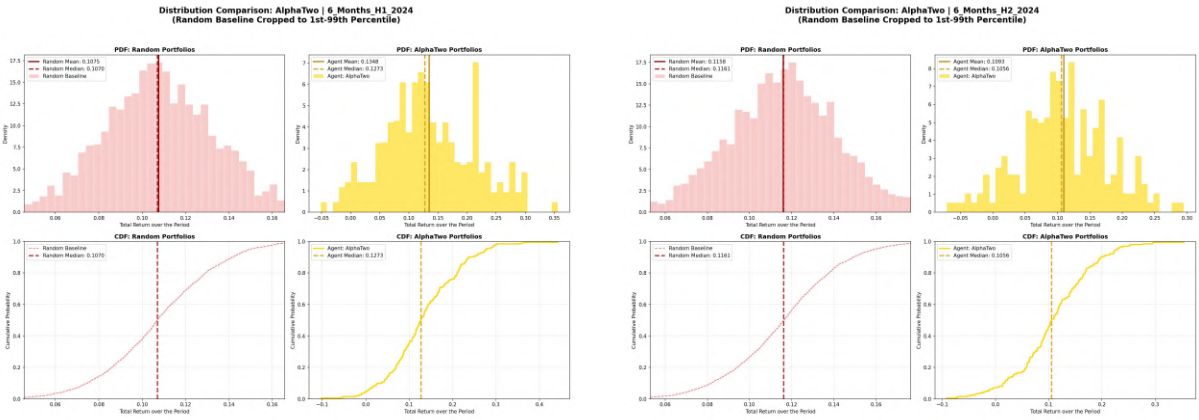
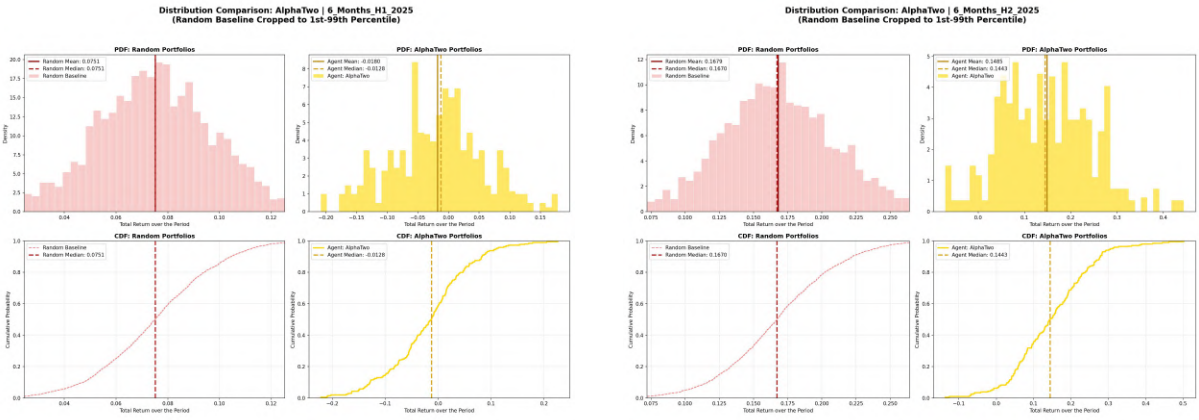


Figure C.3: Returns PDFs and CDFs for AlphaOne's stochastic runs vs random baseline - full run.



(a) H1 2024

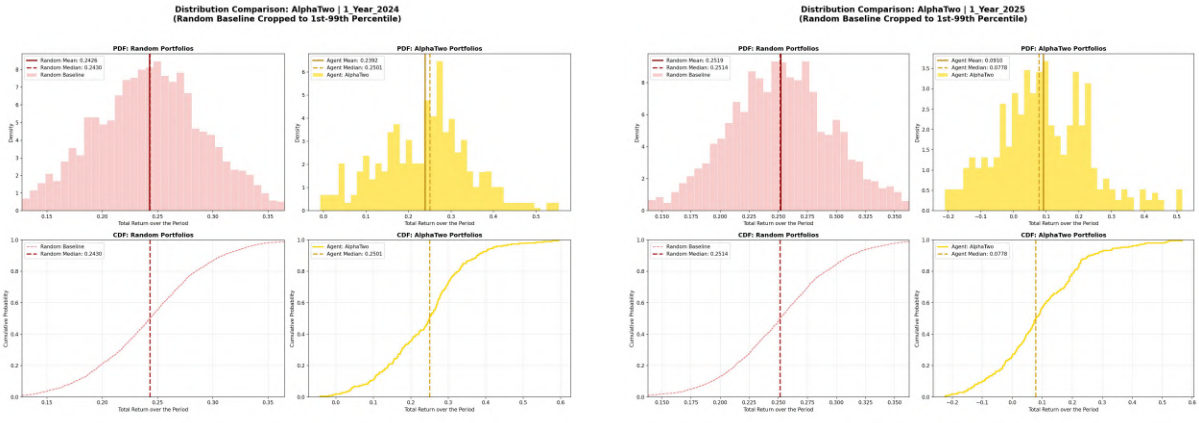
(b) H2 2024



(c) H1 2025

(d) H2 2025

Figure C.4: Returns PDFs and CDFs for AlphaTwo's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.5: Returns PDFs and CDFs for AlphaTwo's stochastic runs vs random baseline - full years.

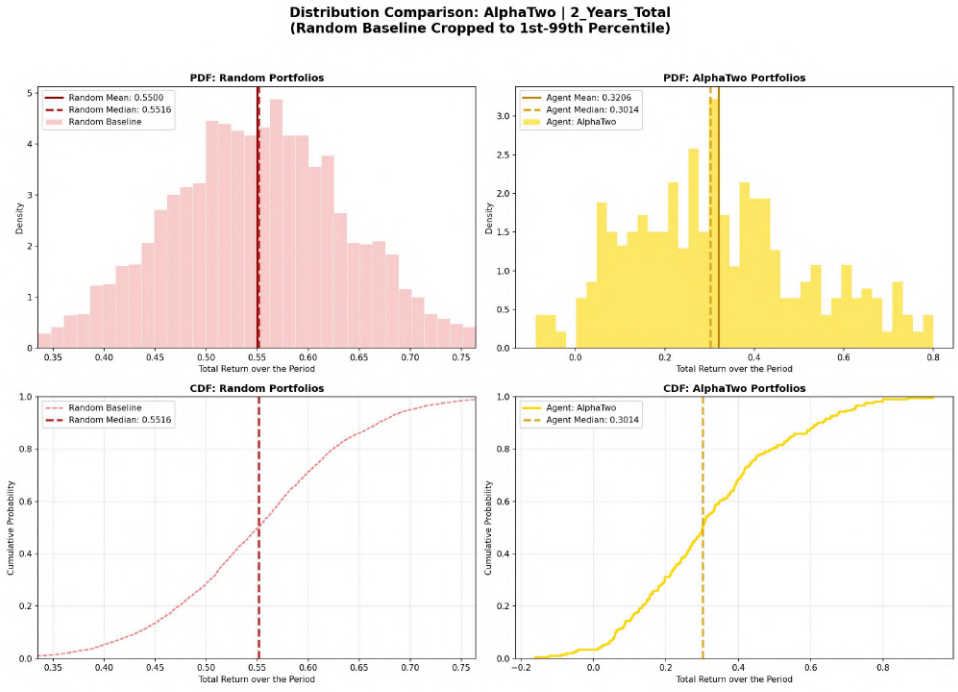
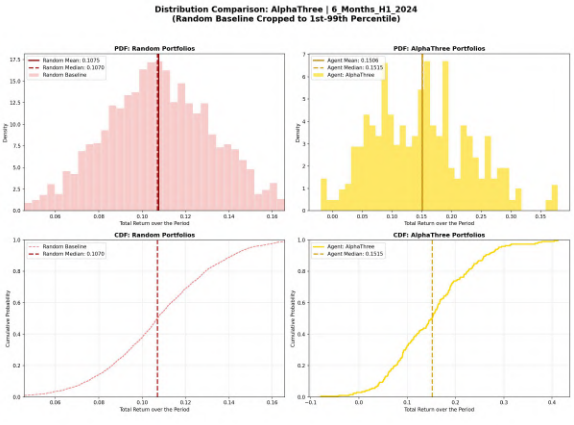
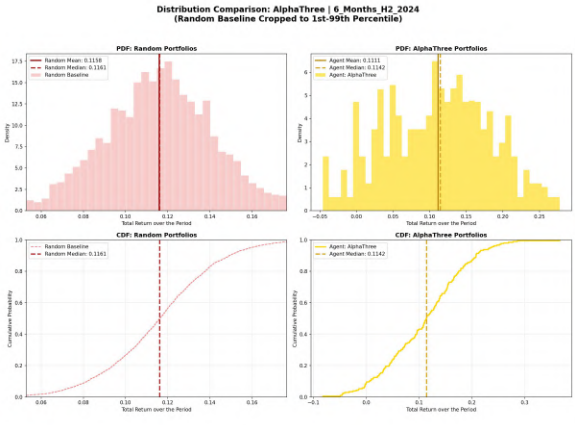


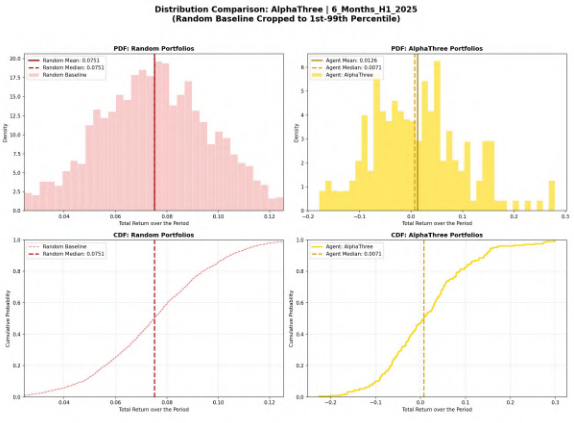
Figure C.6: Returns PDFs and CDFs for AlphaTwo's stochastic runs vs random baseline - full run.



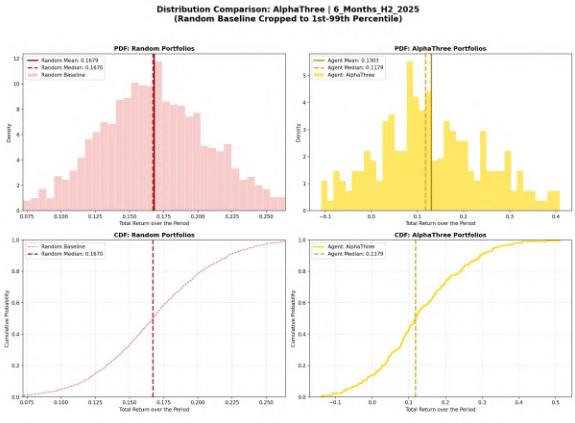
(a) H1 2024



(b) H2 2024

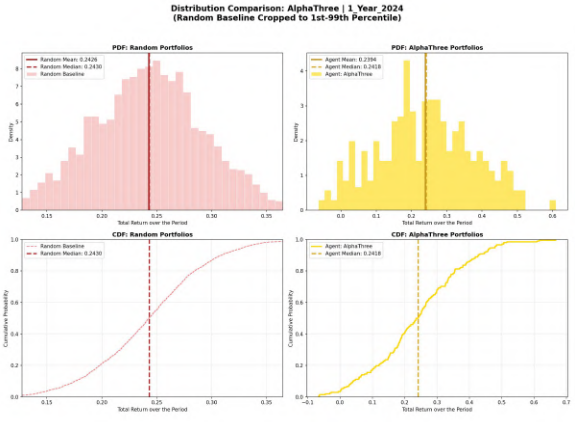


(c) H1 2025

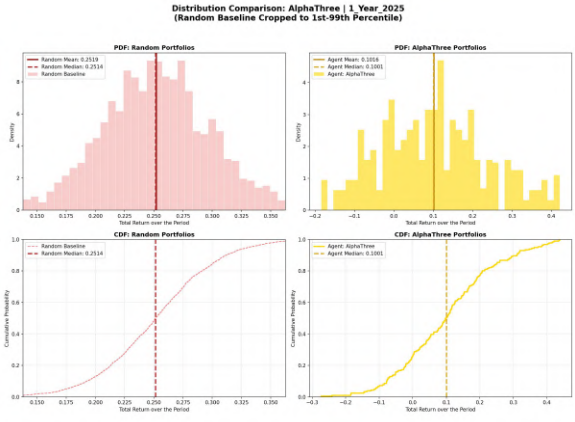


(d) H2 2025

Figure C.7: Returns PDFs and CDFs for AlphaThree's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.8: Returns PDFs and CDFs for AlphaThree's stochastic runs vs random baseline - full years.

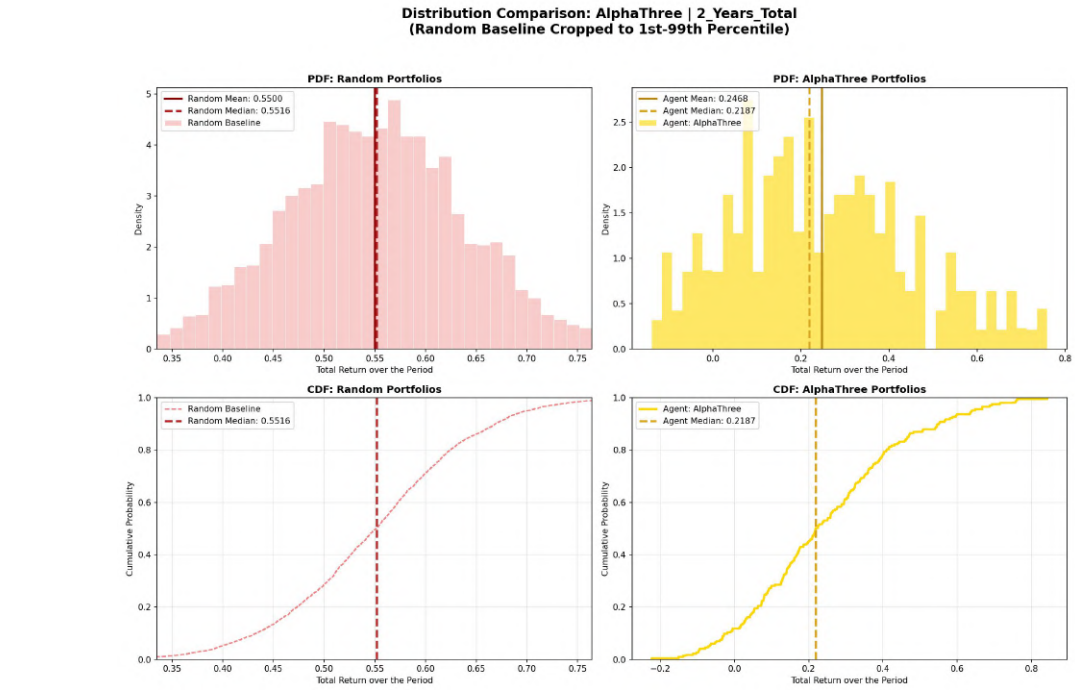
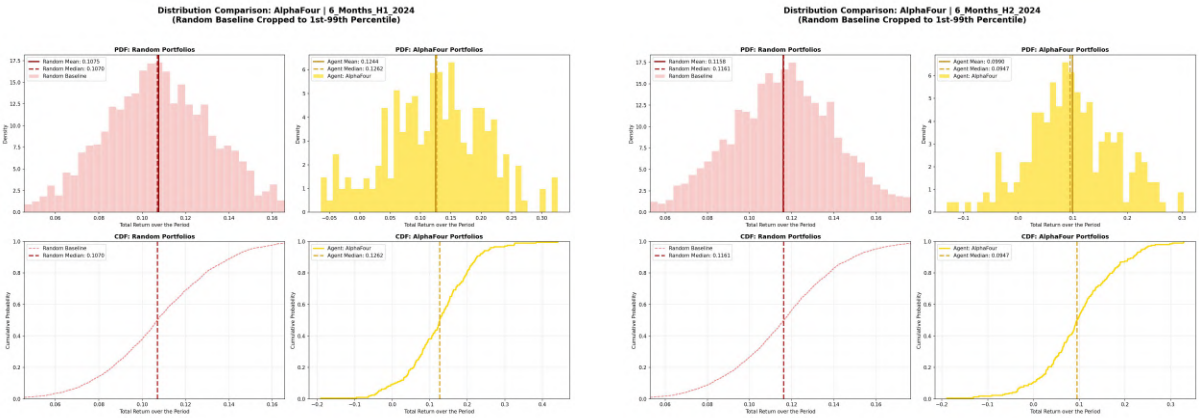
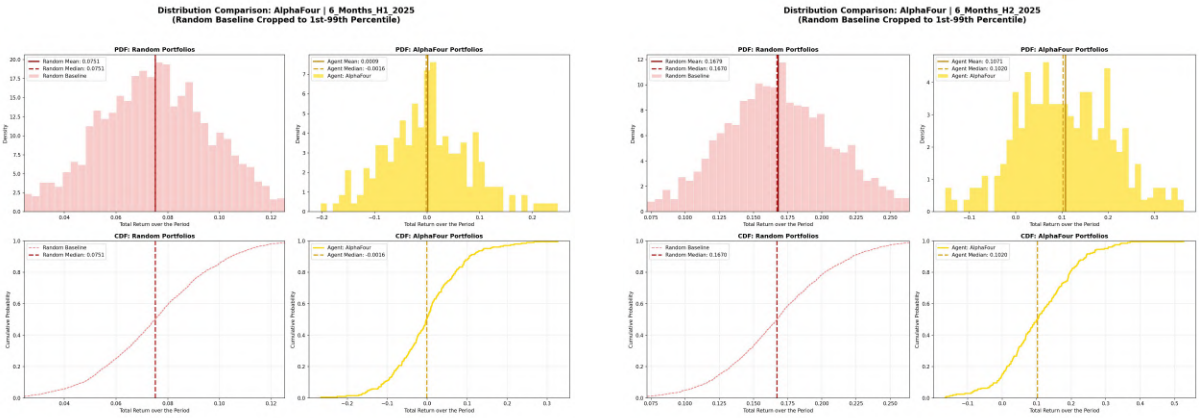


Figure C.9: Returns PDFs and CDFs for AlphaThree's stochastic runs vs random baseline - full run.



(a) H1 2024

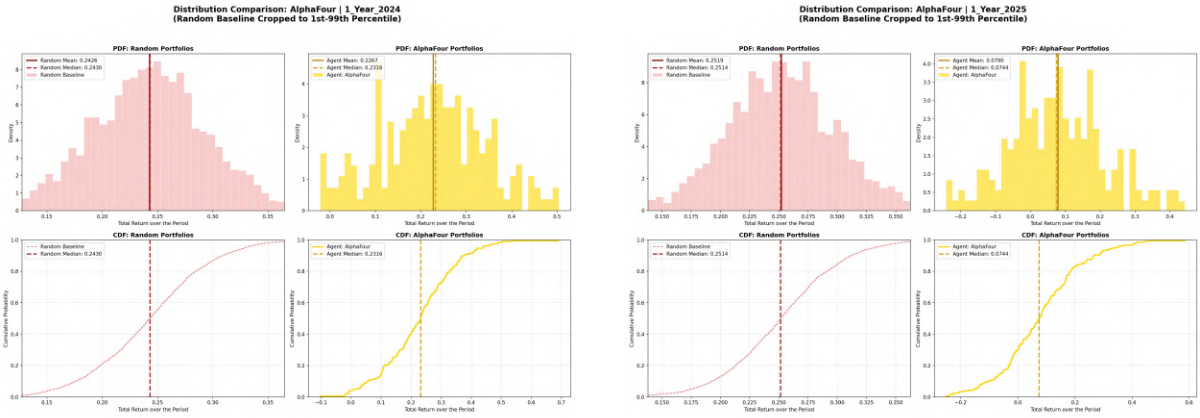
(b) H2 2024



(c) H1 2025

(d) H2 2025

Figure C.10: Returns PDFs and CDFs for AlphaFour's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.11: Returns PDFs and CDFs for AlphaFour's stochastic runs vs random baseline - full years.

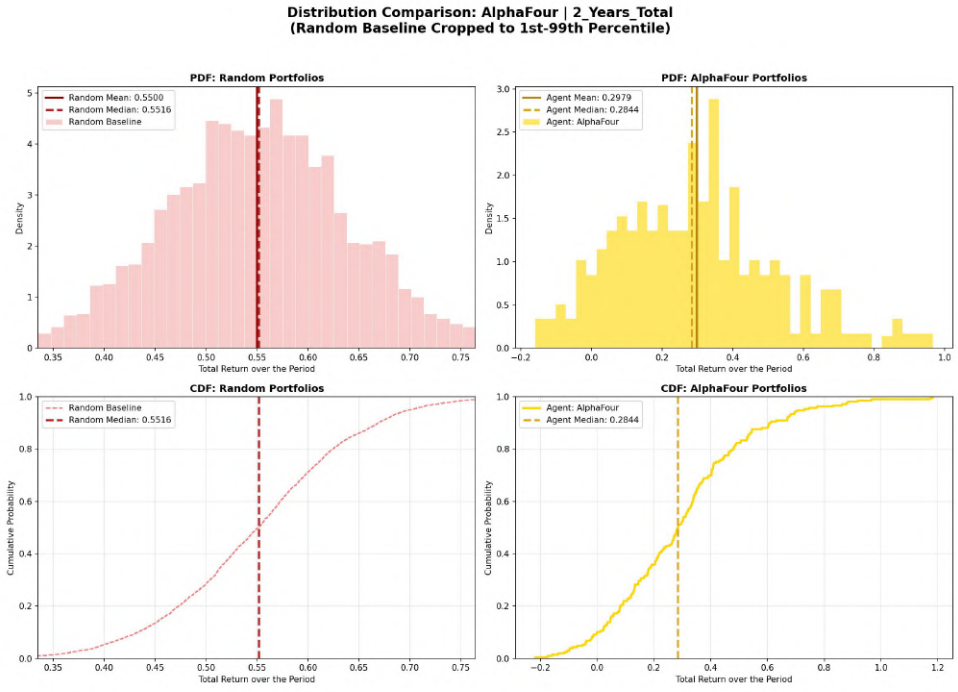


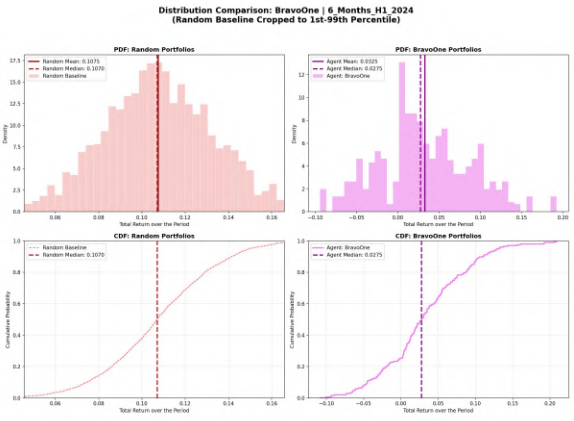
Figure C.12: Returns PDFs and CDFs for AlphaFour's stochastic runs vs random baseline - full run.

C.2 Bravo Family

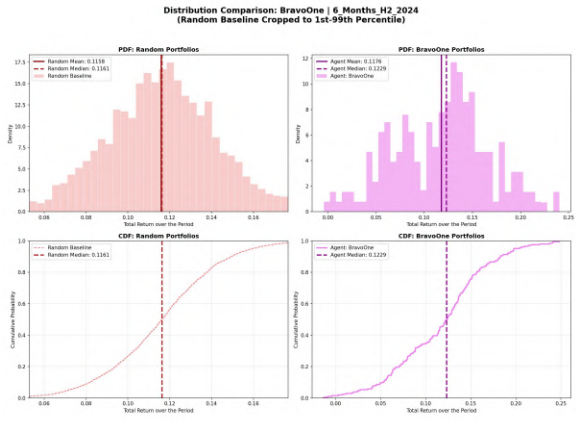
Tables 4.8 and 4.9 present the reader with the results of the stochastic dominance tests—Kolmogorov-Smirnov (KS) and Mann-Whitney U tests (MW)—conducted on the CDFs of stochastic agent returns against the random baseline portfolios. For an in-depth discussion of the results the reader is directed to Section 4.2.2.

In terms of stochastic dominance and stochastic greatness, the Bravo family performed slightly better than the Alpha family. Bravo agents' returns were deemed to be stochastically greater than random in multiple occasions, although unreliably. Furthermore, within the Bravo family we can observe a single instance of stochastic dominance, which was attained by BravoTwo's returns in H2 2024.

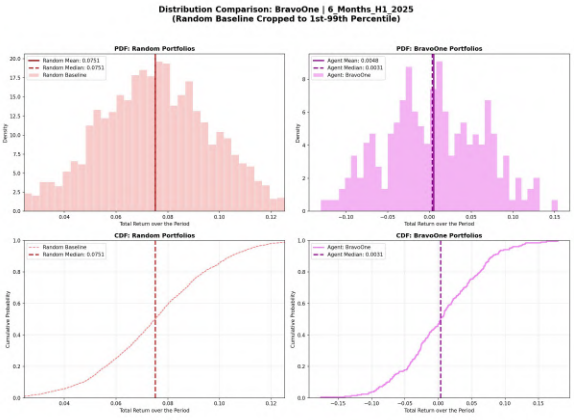
Among the aforementioned instances, the most remarkable are from BravoTwo's and Four's stochastic greatness in the full two-years run, although in both cases returns failed to be stochastically dominant, due to CDF crossings.



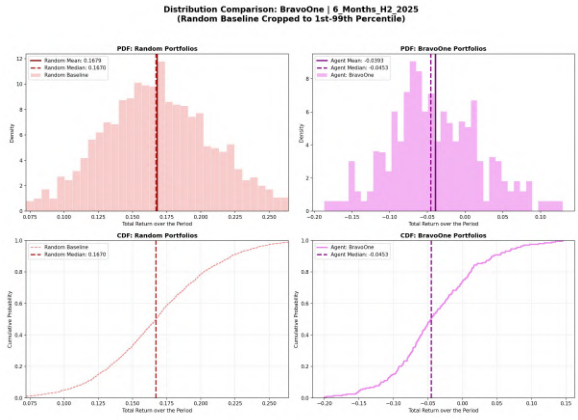
(a) H1 2024



(b) H2 2024

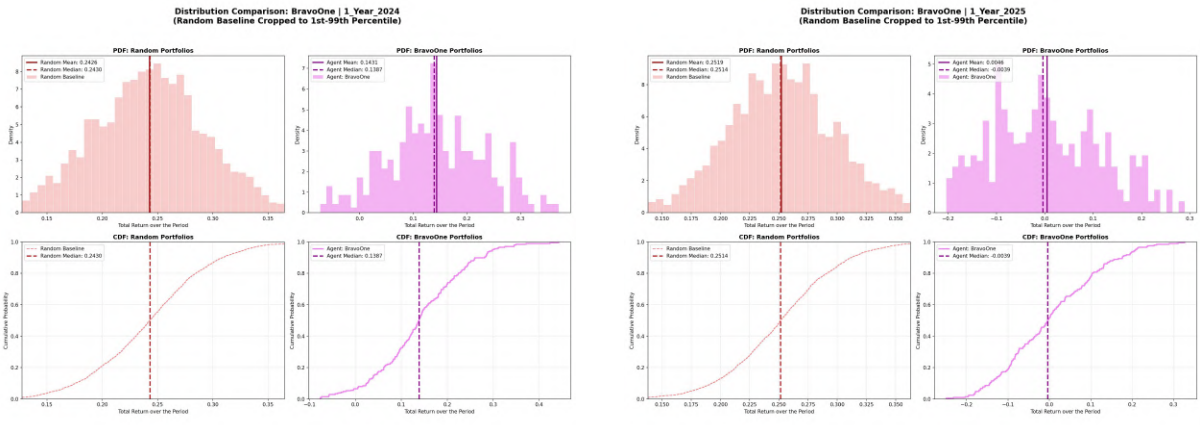


(c) H1 2025



(d) H2 2025

Figure C.13: Returns PDFs and CDFs for BravoOne's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.14: Returns PDFs and CDFs for BravoOne's stochastic runs vs random baseline - full years.

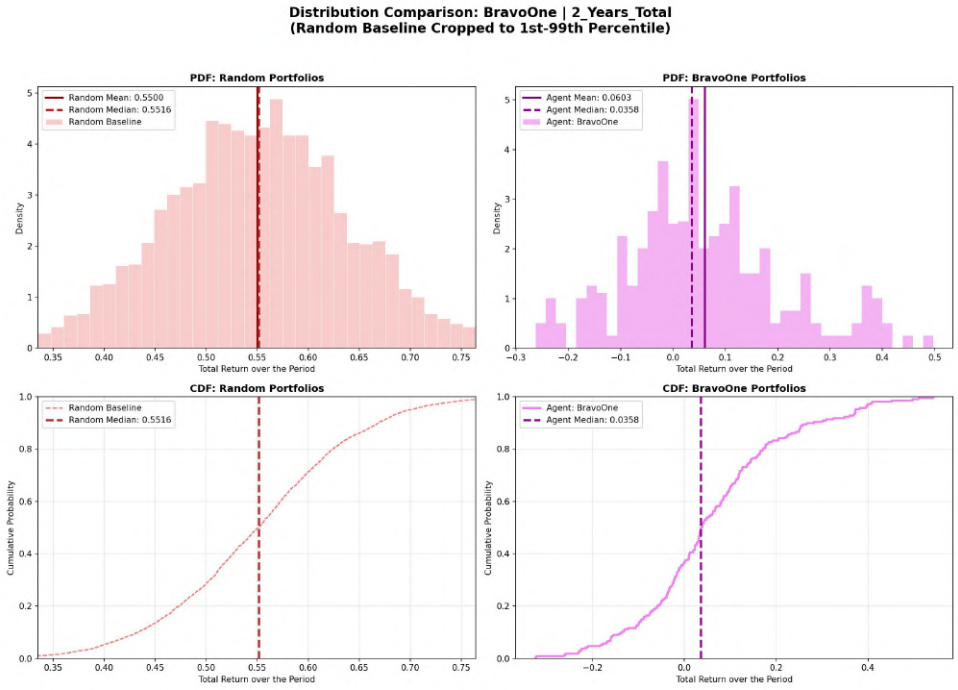
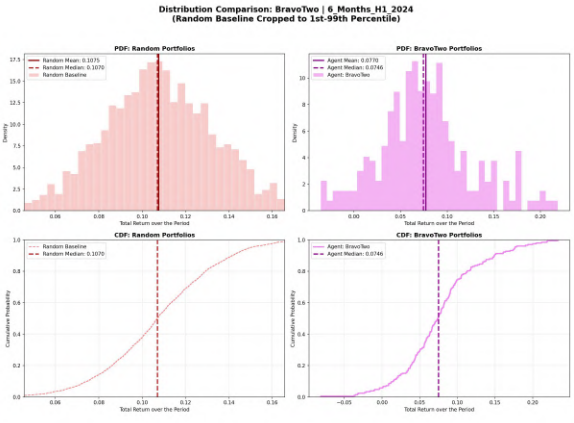
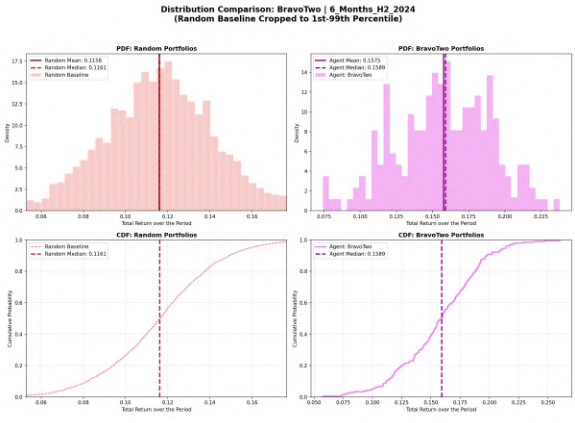


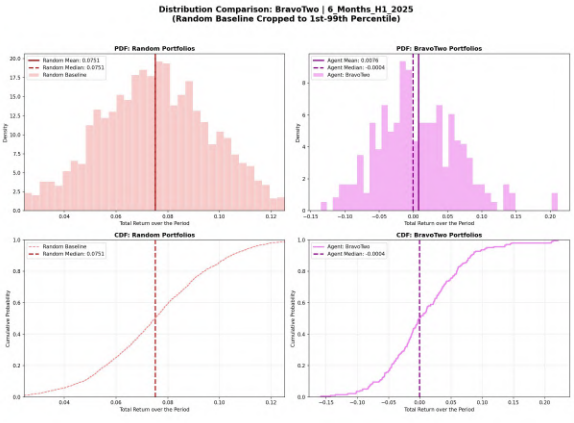
Figure C.15: Returns PDFs and CDFs for BravoOne's stochastic runs vs random baseline - full run.



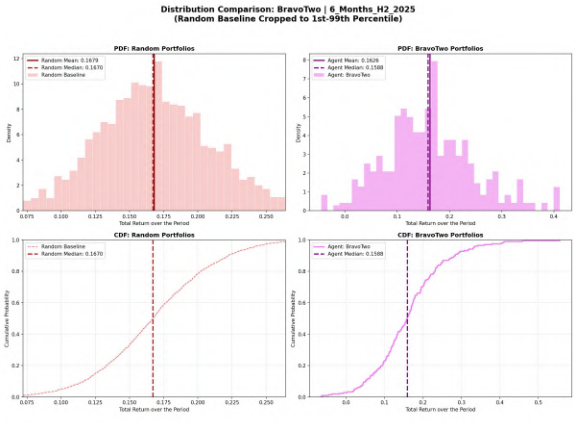
(a) H1 2024



(b) H2 2024

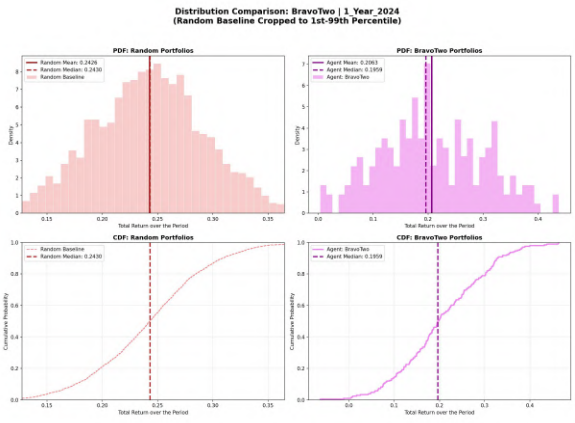


(c) H1 2025

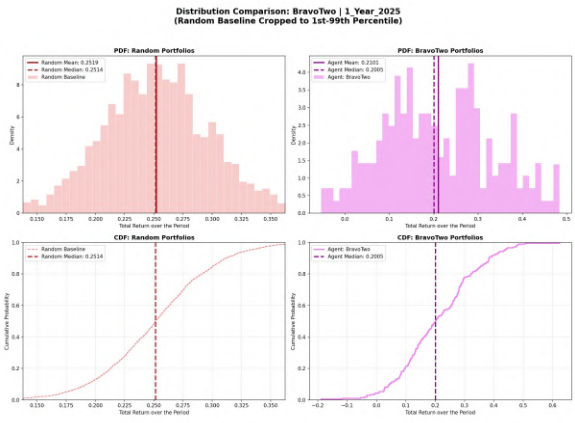


(d) H2 2025

Figure C.16: Returns PDFs and CDFs for BravoTwo's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.17: Returns PDFs and CDFs for BravoTwo's stochastic runs vs random baseline - full years.

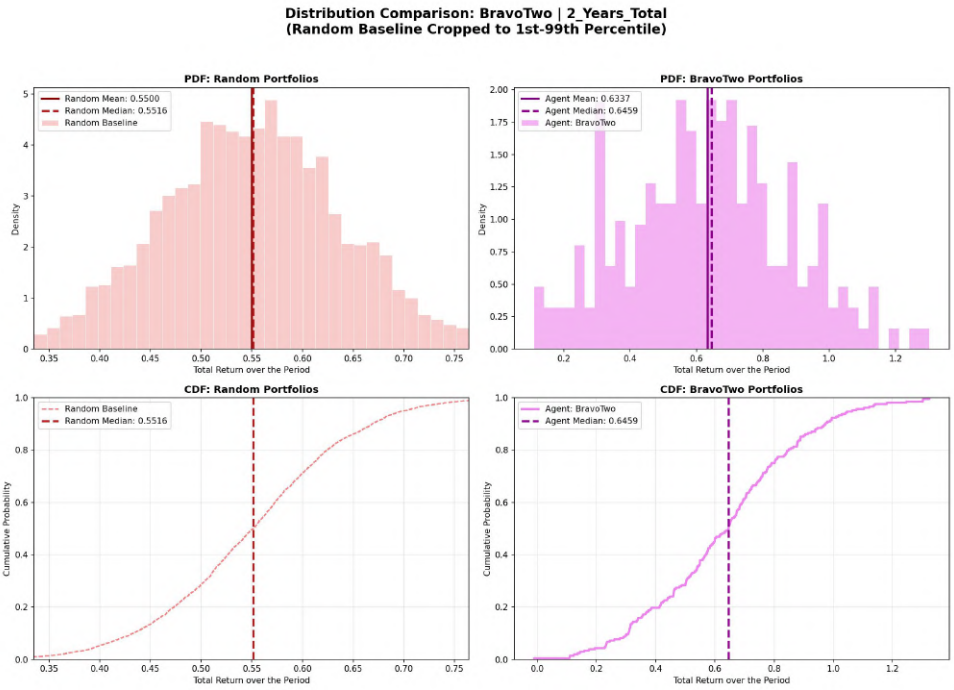
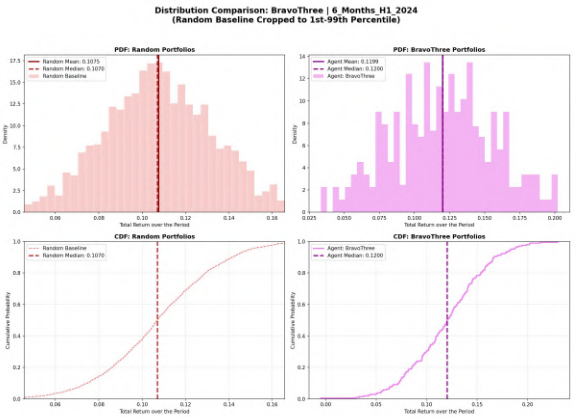
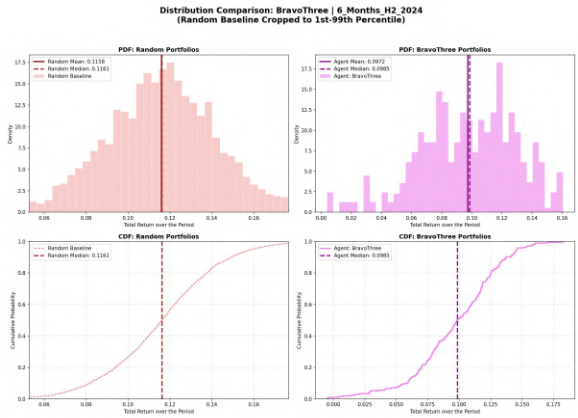


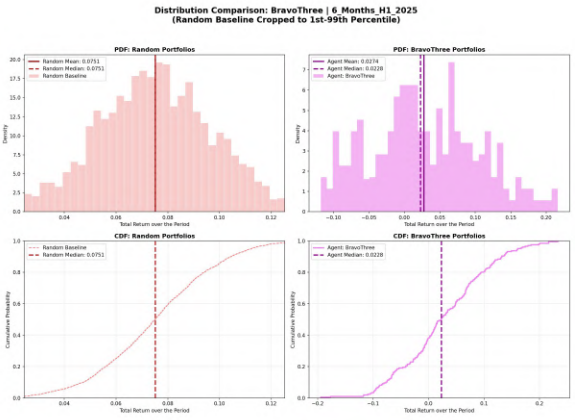
Figure C.18: Returns PDFs and CDFs for BravoTwo's stochastic runs vs random baseline - full run.



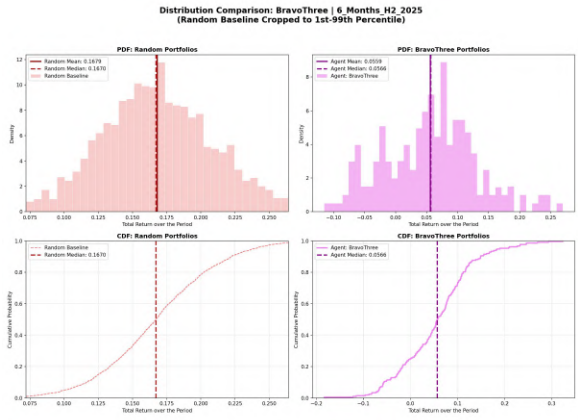
(a) H1 2024



(b) H2 2024

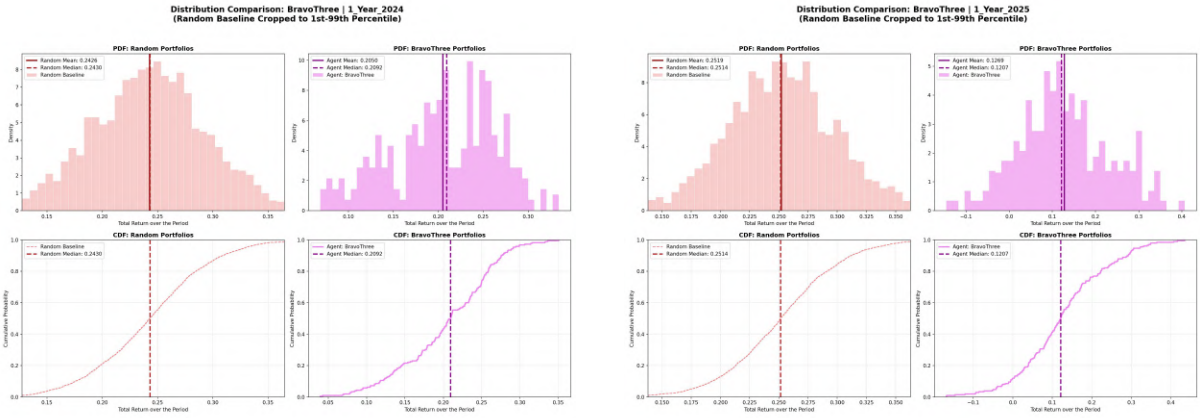


(c) H1 2025



(d) H2 2025

Figure C.19: Returns PDFs and CDFs for BravoThree's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.20: Returns PDFs and CDFs for BravoThree's stochastic runs vs random baseline - full years.

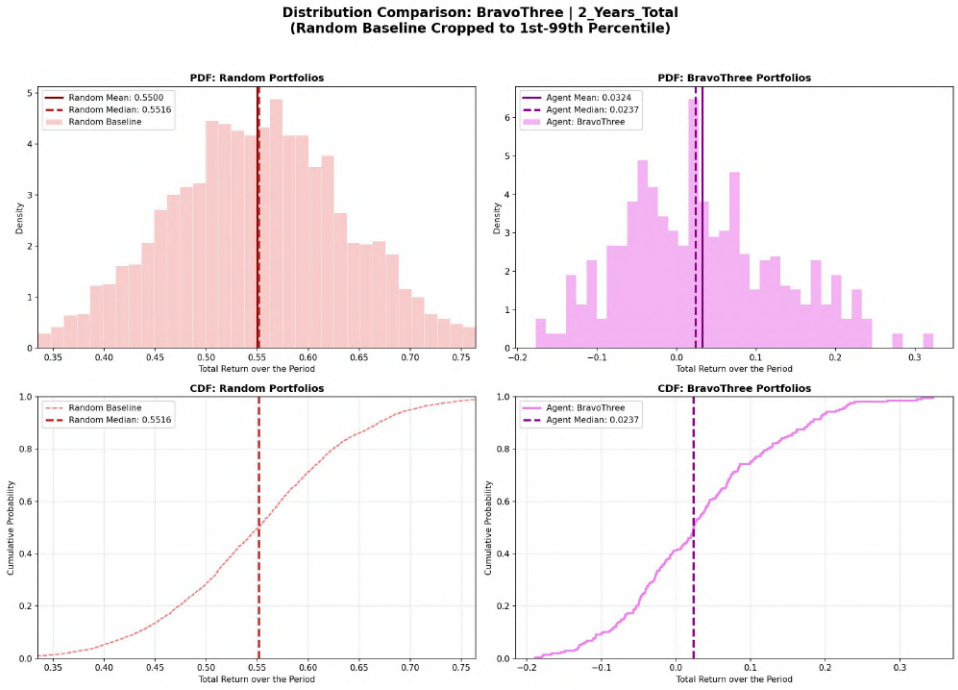
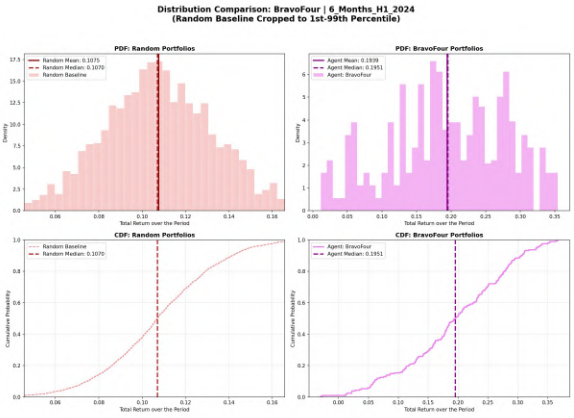
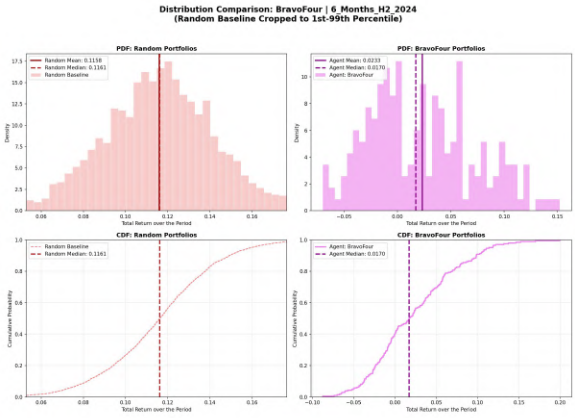


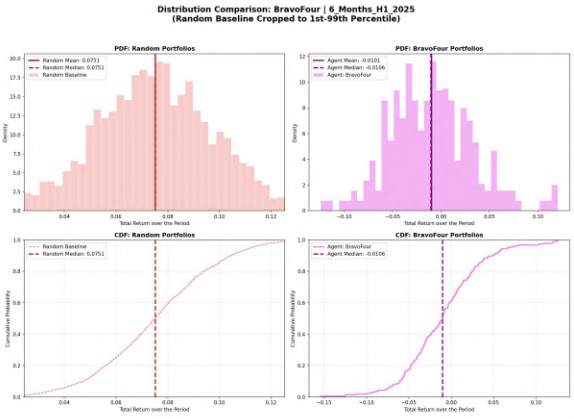
Figure C.21: Returns PDFs and CDFs for BravoThree's stochastic runs vs random baseline - full run.



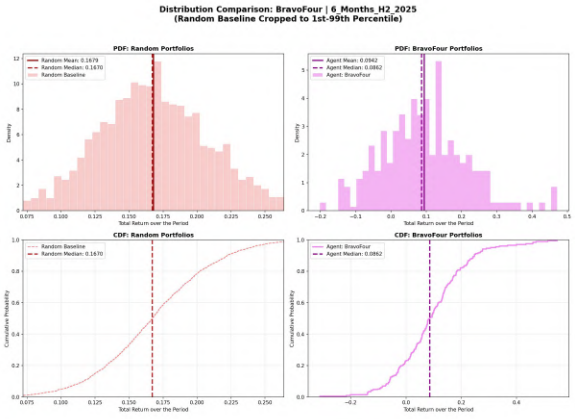
(a) H1 2024



(b) H2 2024

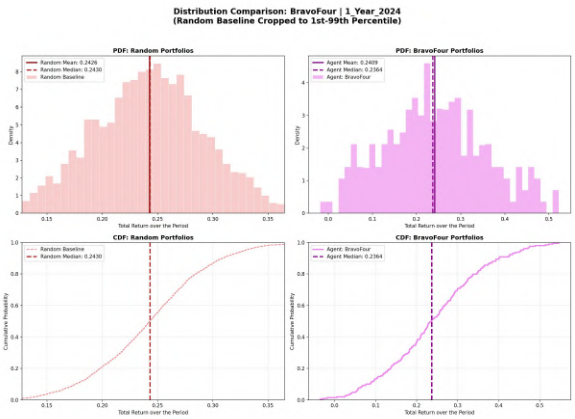


(c) H1 2025

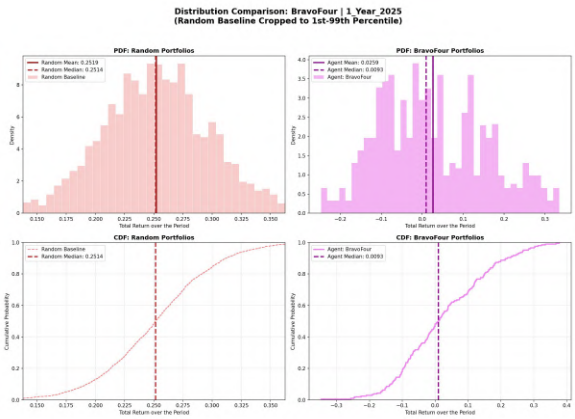


(d) H2 2025

Figure C.22: Returns PDFs and CDFs for BravoFour's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.23: Returns PDFs and CDFs for BravoFour's stochastic runs vs random baseline - full years.

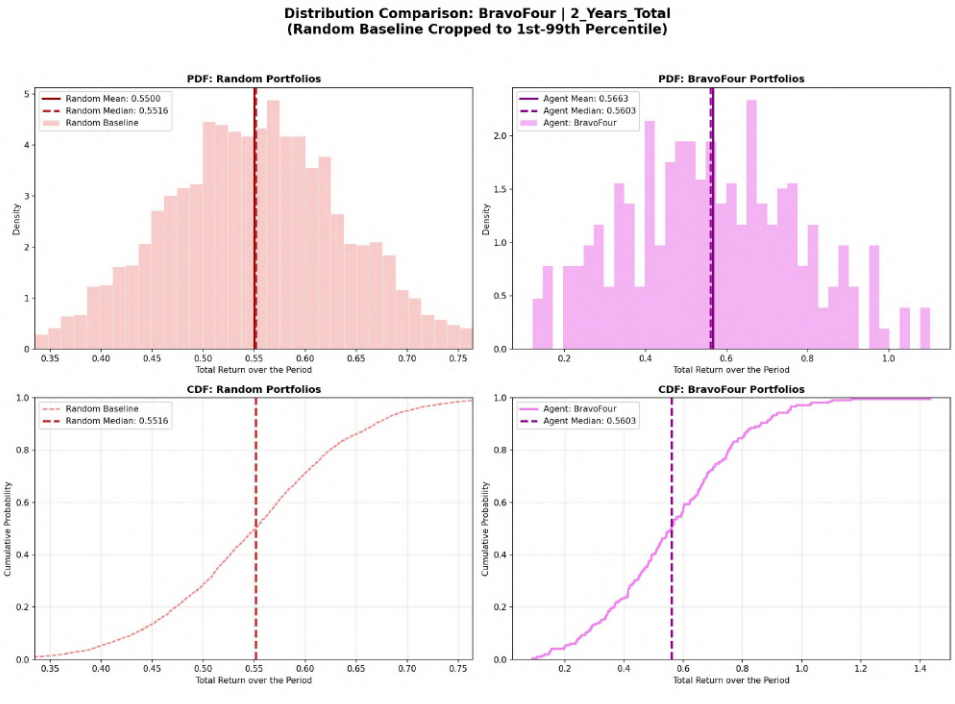


Figure C.24: Returns PDFs and CDFs for BravoFour's stochastic runs vs random baseline - full run.

C.3 Charlie Family

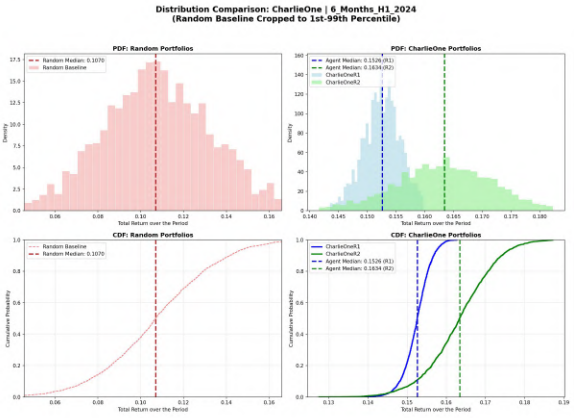
Tables 4.10 and 4.11 present the reader with results from the stochastic dominance test conducted for each Charlie agent returns against random portfolio returns, when agents are simulated following a stochastic policy. The plots for empirical PDFs and CDFs of agents' stochastic returns against random returns, for both reward functions, are presented in the following subsections.

One may observe that the results for CharlieOne, Two, and Three were invariant with respect to the training function used. The interesting instance concerns CharlieFour, which is the weakest agent in terms of stochastic superiority and FSD when trained using the Sharpe reward, while tying for the first place when trained using the IRR reward function. As always, our analysis focus mainly on R1 agents.

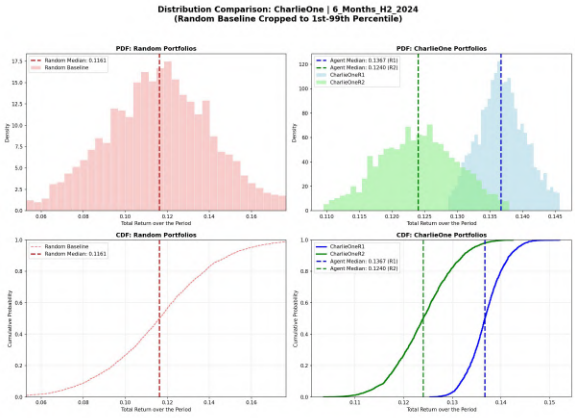
From the presented tables, we can observe that CharlieOne had the best performances among R1 agents, its returns being stochastically greater than the random baseline returns in five out of seven windows, sporadically achieving FSD. CharlieFour performed similarly to CharlieOne, when considering only R1 agents, attaining FSD one more time than One. However, we deem CharlieOne's performances superior to Four's due to it having higher median return on average. CharlieTwo's performances are slightly worse than One's and Four's, but stay close to satisfactory overall, its returns being deemed stochastically greater than the baseline in three out of seven windows.

CharlieThree was the weakest member of the pack, with its returns being stochastically greater than the baseline only twice, and attaining FSD only once.

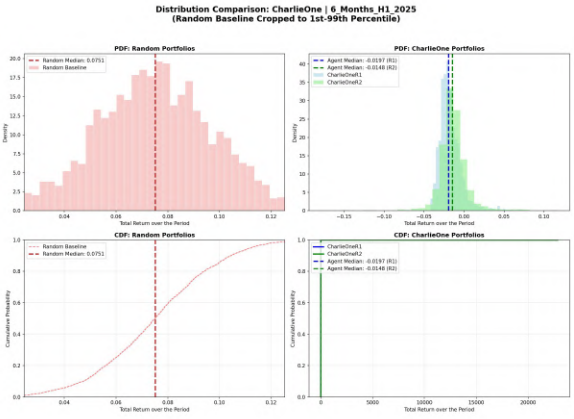
Since CharlieFourR1 performed vastly greater than its R2 twin, we can confidently claim that the IRR reward function has been proven to be vastly superior than the Sharpe reward with respect to PPO trained agents.



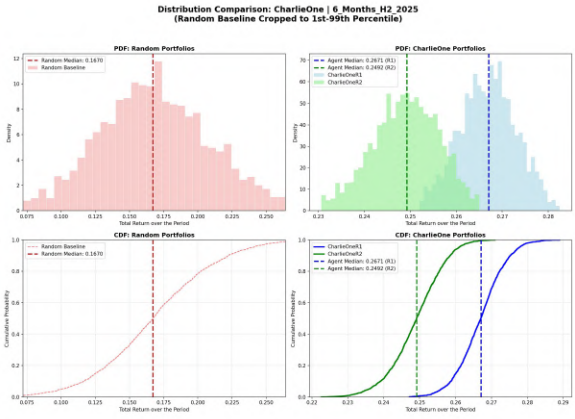
(a) H1 2024



(b) H2 2024

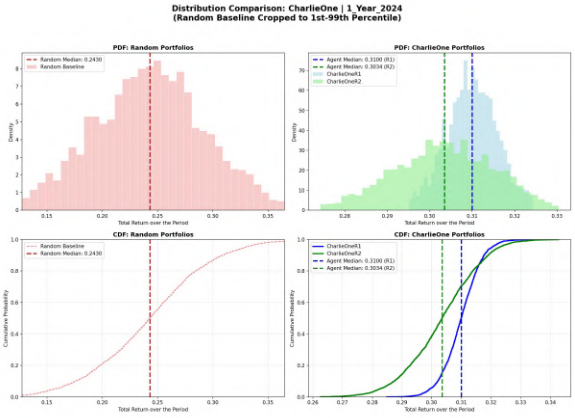


(c) H1 2025

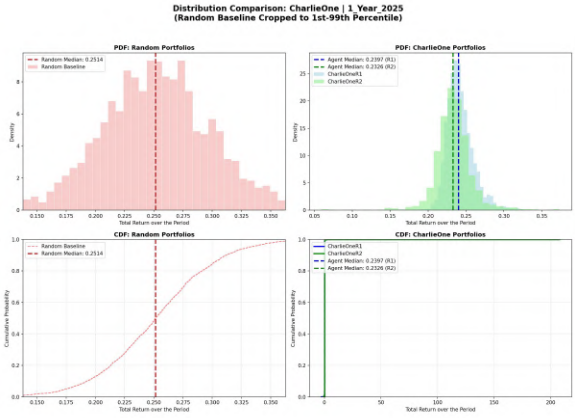


(d) H2 2025

Figure C.25: Returns PDFs and CDFs for CharlieOne's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.26: Returns PDFs and CDFs for CharlieOne's stochastic runs vs random baseline - full years.

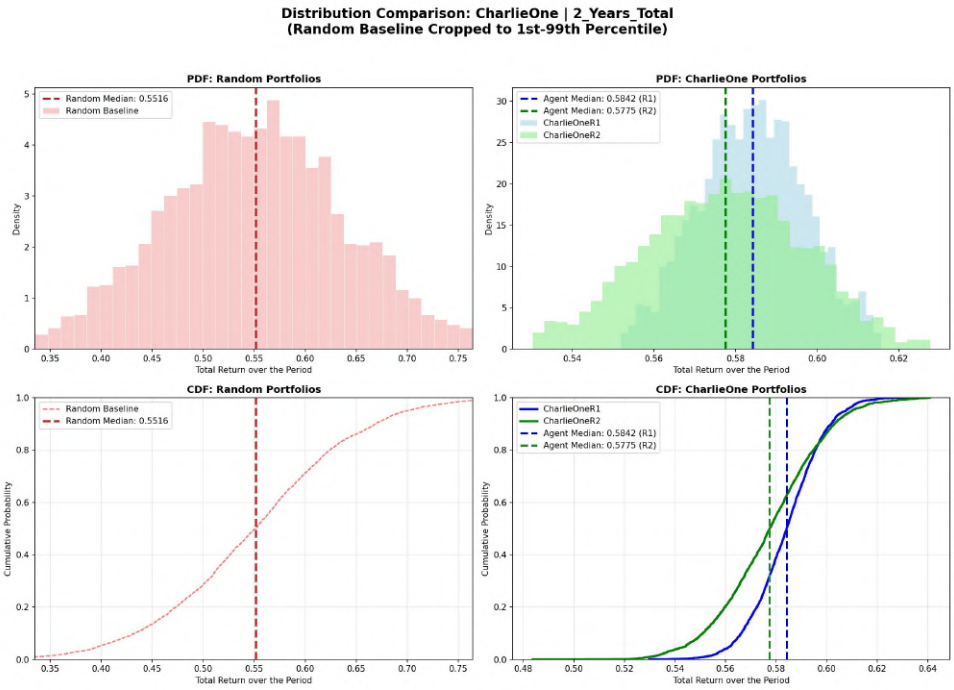
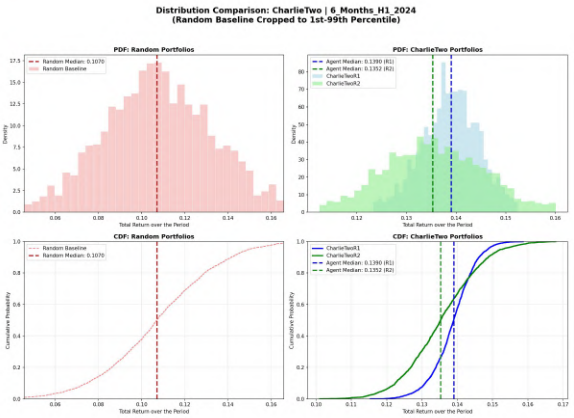
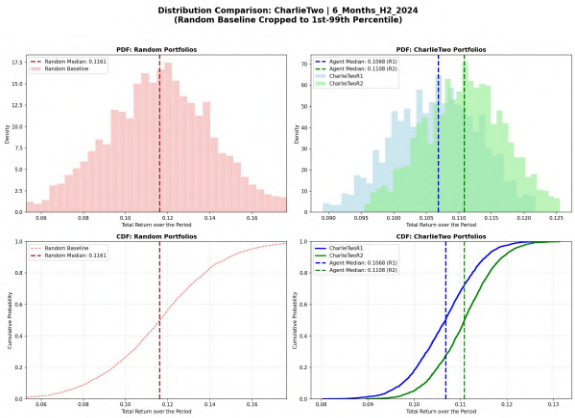


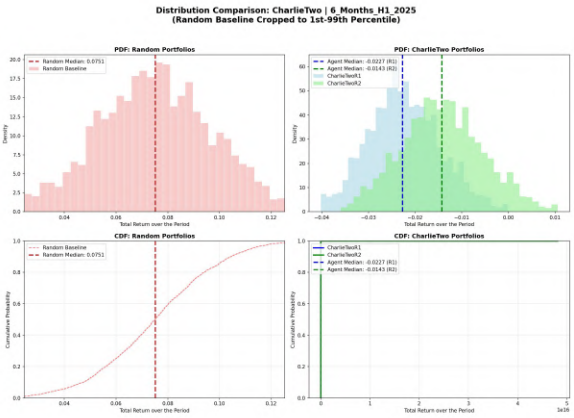
Figure C.27: Returns PDFs and CDFs for CharlieOne's stochastic runs vs random baseline - full run.



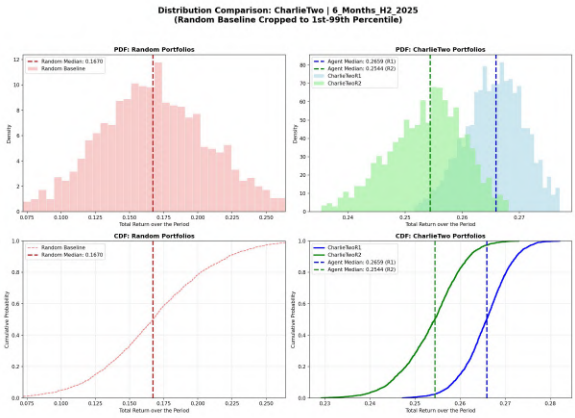
(a) H1 2024



(b) H2 2024

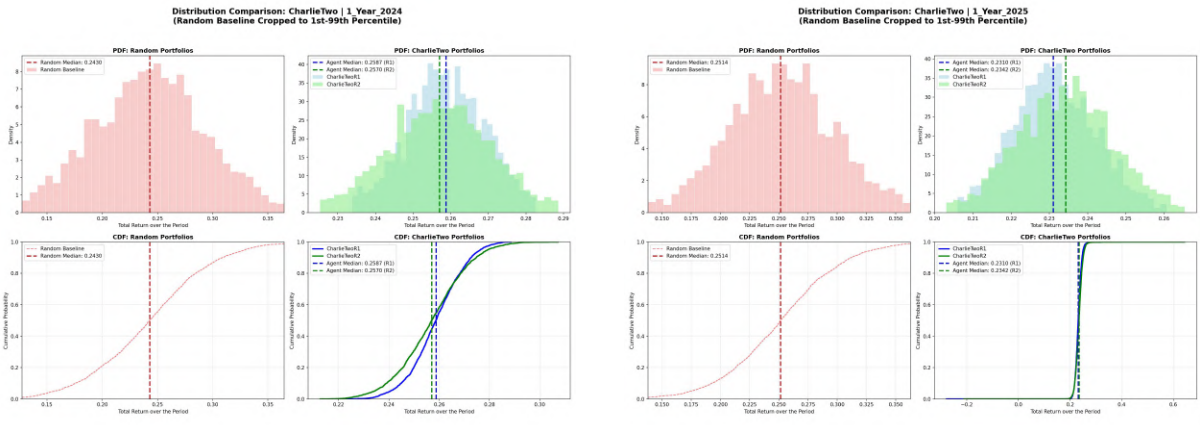


(c) H1 2025



(d) H2 2025

Figure C.28: Returns PDFs and CDFs for CharlieTwo's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.29: Returns PDFs and CDFs for CharlieTwo's stochastic runs vs random baseline - full years.

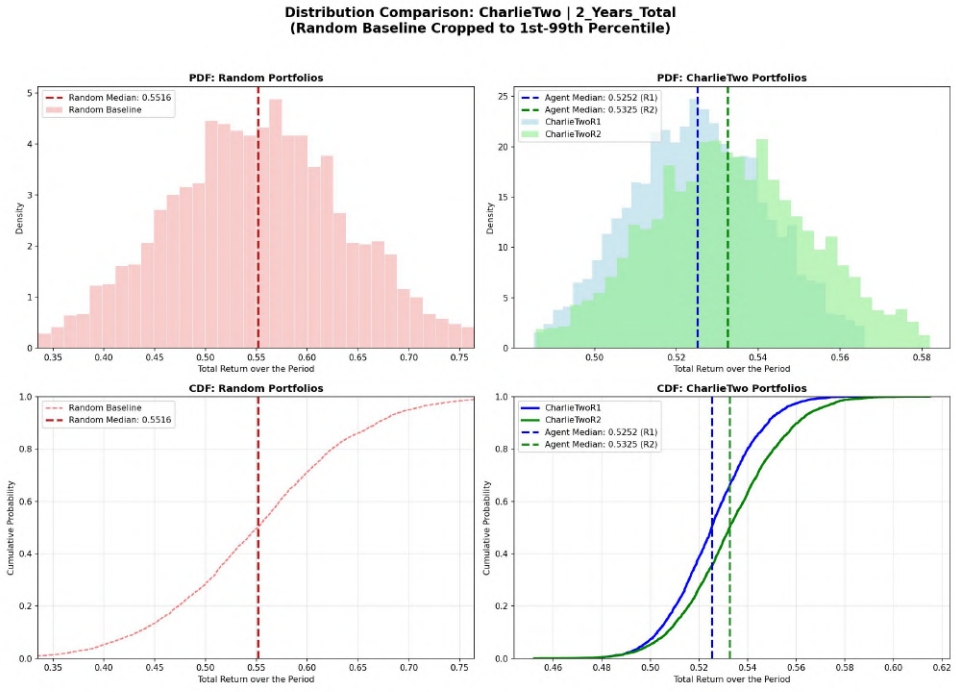
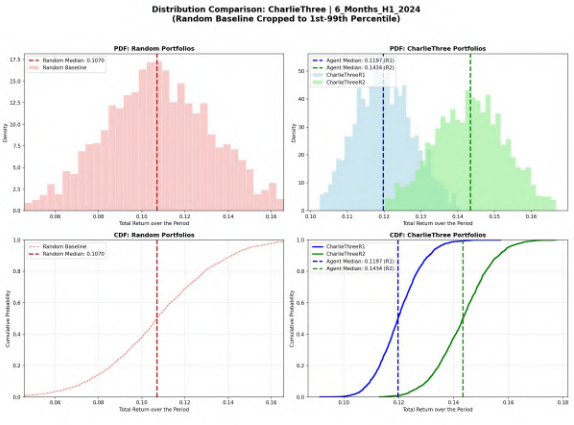
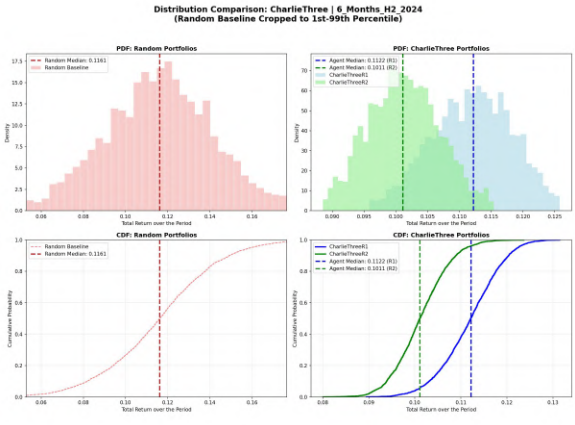


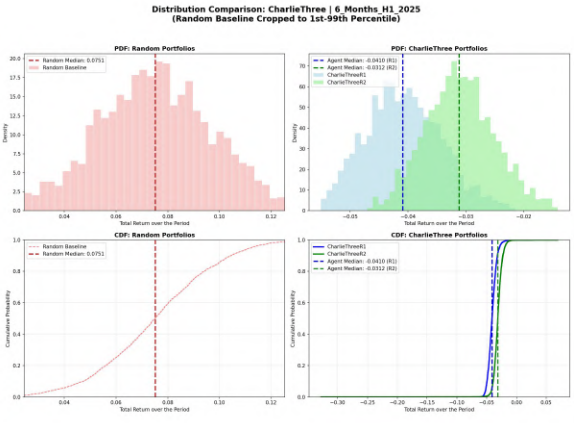
Figure C.30: Returns PDFs and CDFs for CharlieTwo's stochastic runs vs random baseline - full run.



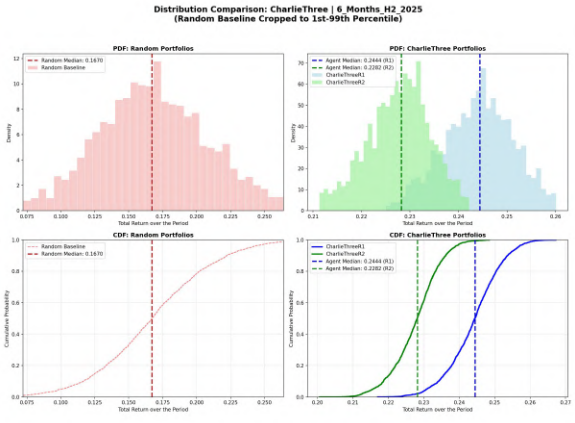
(a) H1 2024



(b) H2 2024

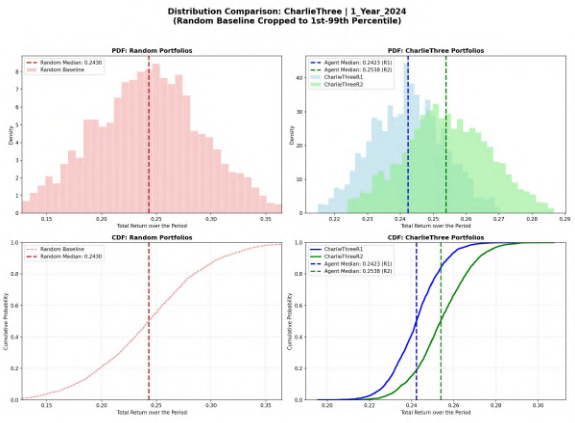


(c) H1 2025

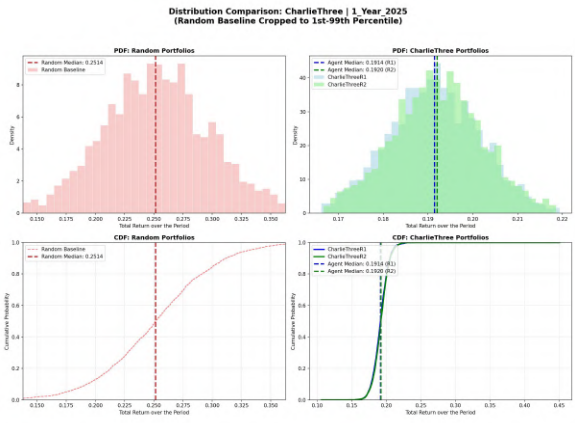


(d) H2 2025

Figure C.31: Returns PDFs and CDFs for CharlieThree's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.32: Returns PDFs and CDFs for CharlieThree's stochastic runs vs random baseline - full years.

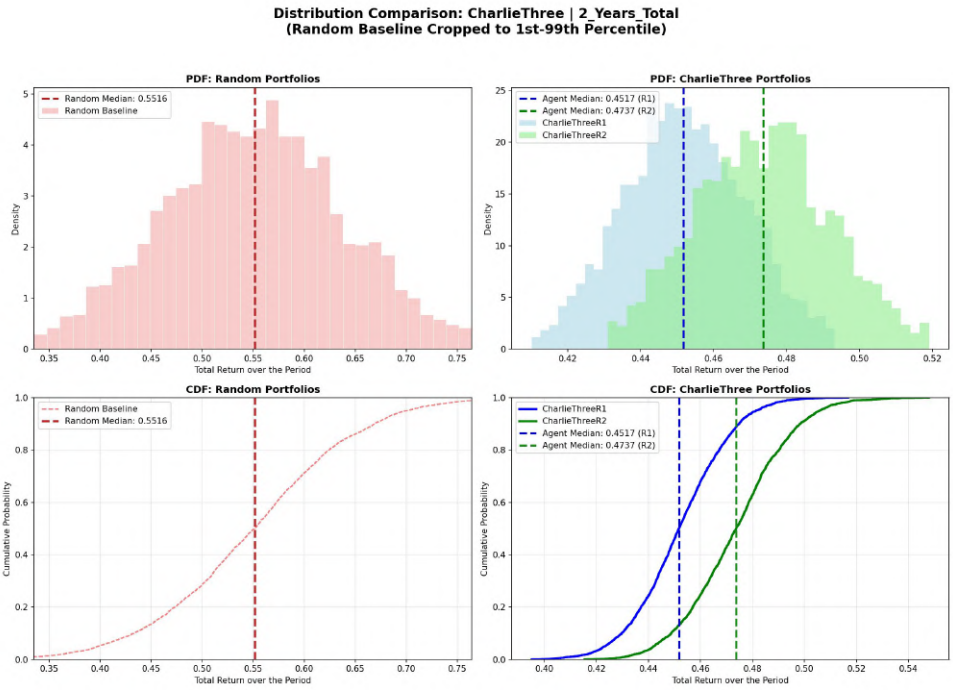
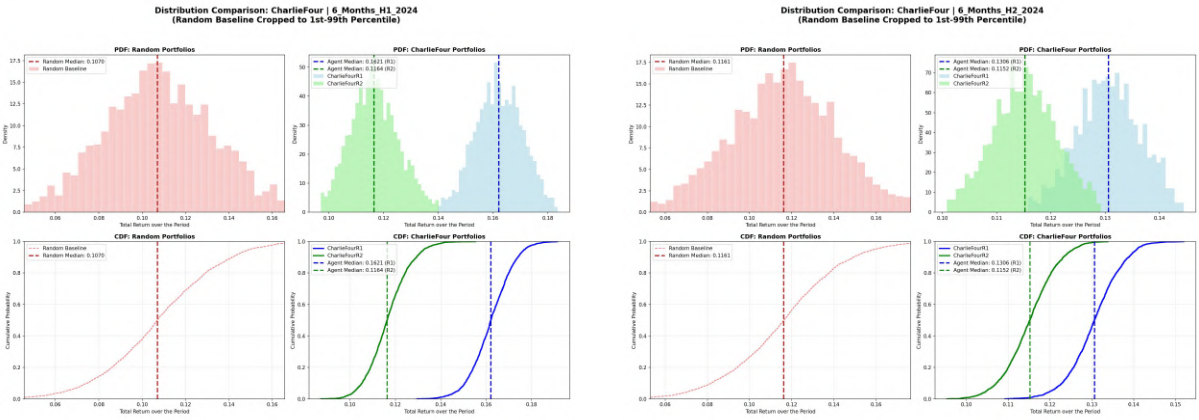
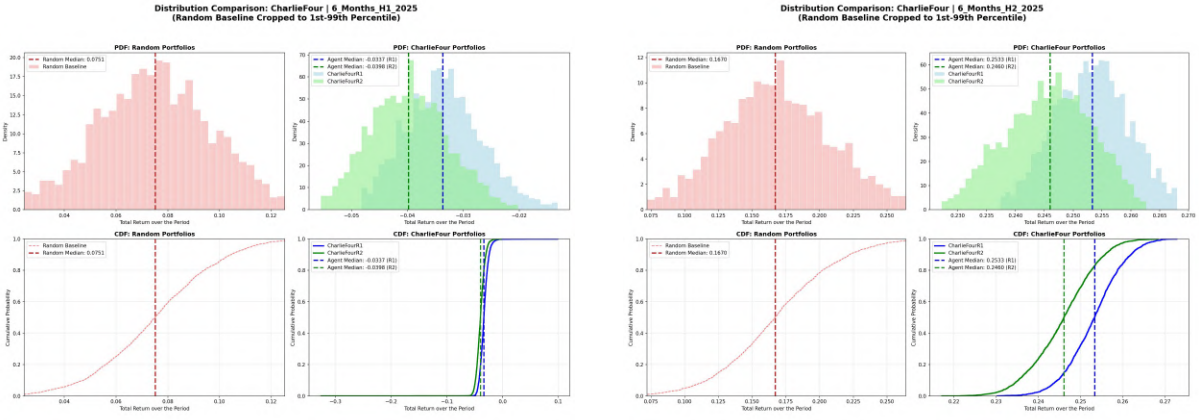


Figure C.33: Returns PDFs and CDFs for CharlieThree's stochastic runs vs random baseline - full run.



(a) H1 2024

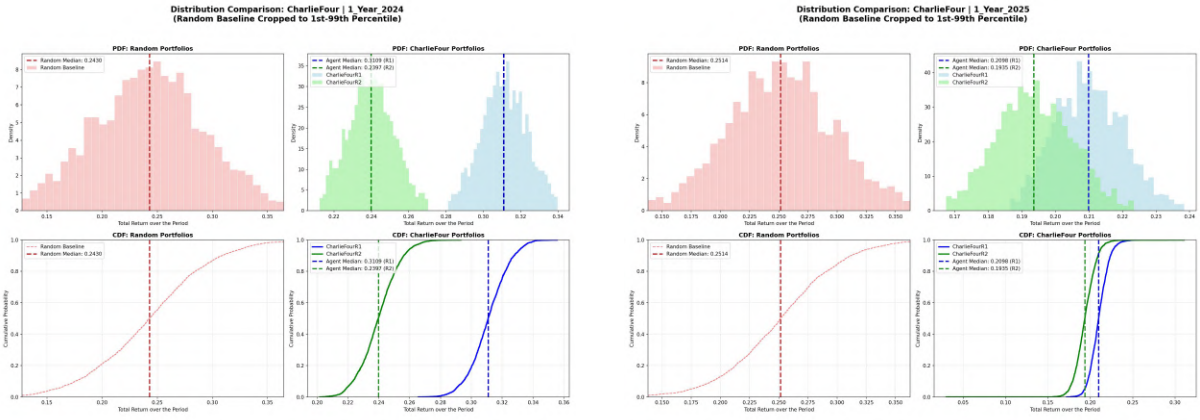
(b) H2 2024



(c) H1 2025

(d) H2 2025

Figure C.34: Returns PDFs and CDFs for CharlieFour's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.35: Returns PDFs and CDFs for CharlieFour's stochastic runs vs random baseline - full years.

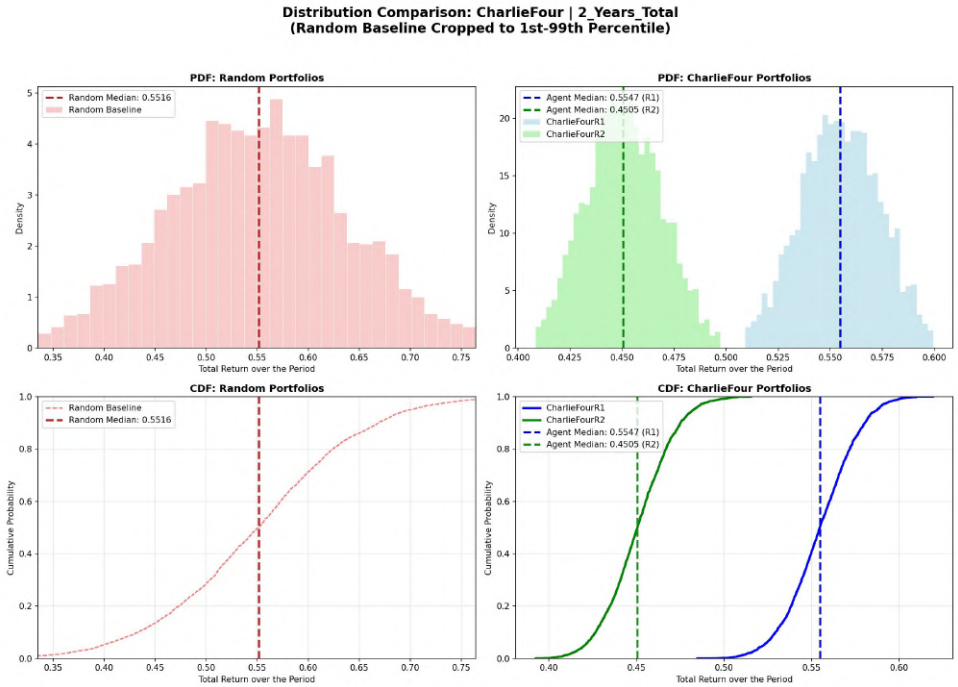


Figure C.36: Returns PDFs and CDFs for CharlieFour's stochastic runs vs random baseline - full run.

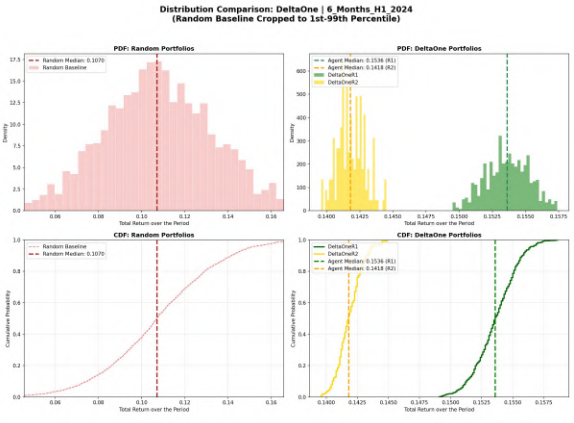
C.4 Delta Family

Tables 4.12 and 4.13 present the reader with the results of the stochastic dominance tests—Kolmogorov-Smirnov (KS) and Mann-Whitney U tests (MW)—conducted on the CDFs of stochastic agent returns against the random baseline portfolios. For an in-depth discussion of the results the reader is directed to Section 4.2.2.

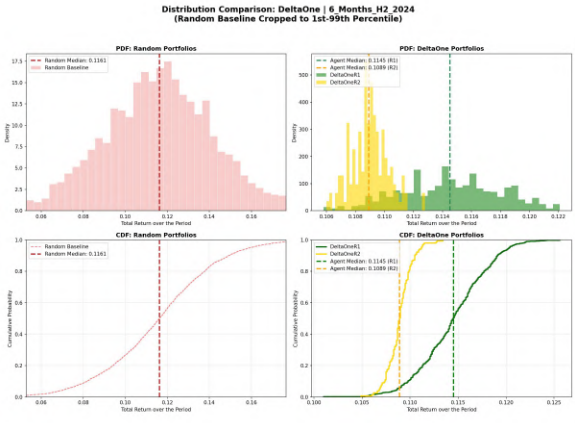
Among the agents trained with the IRR reward function (R1), the sole to obtain satisfactory results in terms of stochastic greatness over the random baseline was DeltaOne, its returns being stochastically greater than random in four out of seven backtesting windows, attaining FSD in two of those occasions. All the other R1 agents had generally unsatisfactory performances, which worsened with agent complexity.

The best performing agent among the Sharpe trained ones (R2) was DeltaThree. In five out of seven backtesting windows, DeltaThree returns were deemed stochastically greater than the baseline, achieving also stochastic dominance in two occasions. DeltaTwo and Four also performed well, their returns being stochastically grater in four out of seven windows, being also stochastically dominant over the baseline in two and one occasions respectively. The consistence for DeltaTwo's, Three's, and Four's returns in being stochastically greater than baseline returns is remarkable and a major improvement over Bravo's performances.

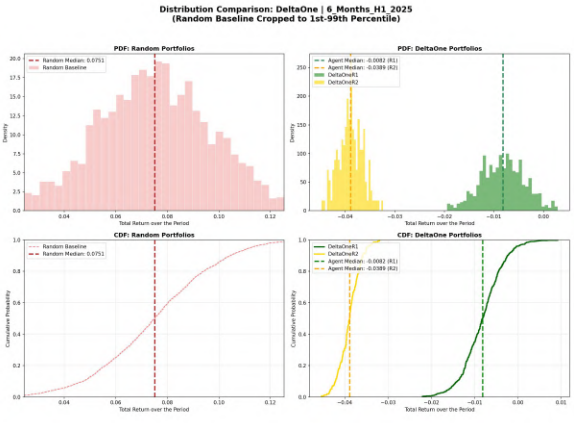
In conclusion, among the entire Delta family, considering both R1 and R2 agents, four agents had satisfactory performances in stochastic testing: DeltaOneR1, TwoR2, ThreeR2, and FourR2.



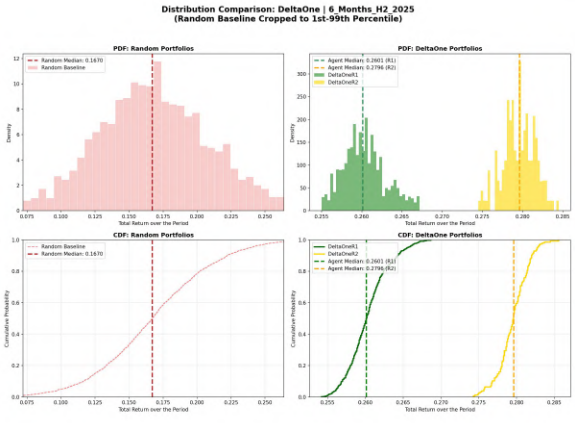
(a) H1 2024



(b) H2 2024

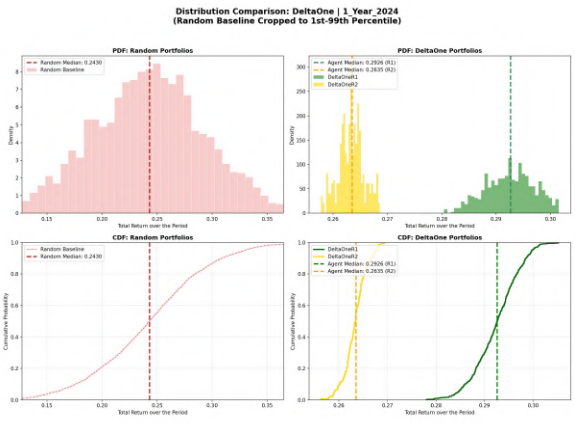


(c) H1 2025

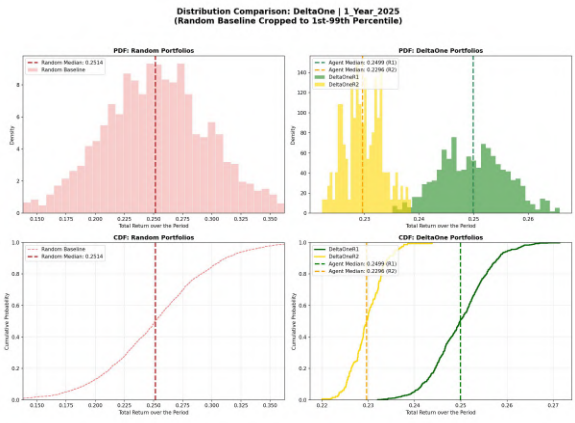


(d) H2 2025

Figure C.37: Returns PDFs and CDFs for DeltaOne's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.38: Returns PDFs and CDFs for DeltaOne's stochastic runs vs random baseline - full years.

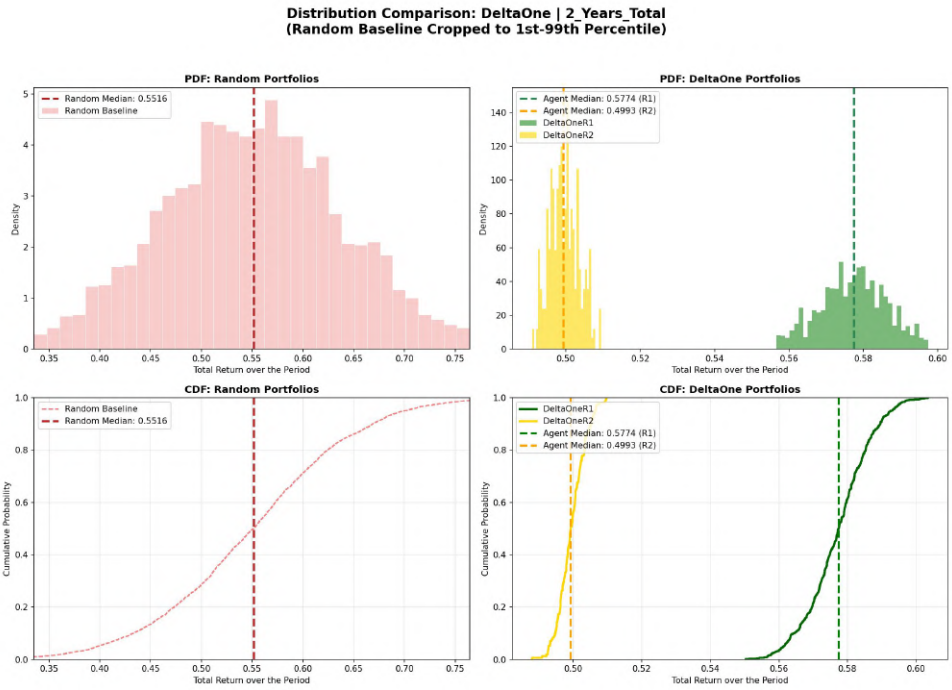
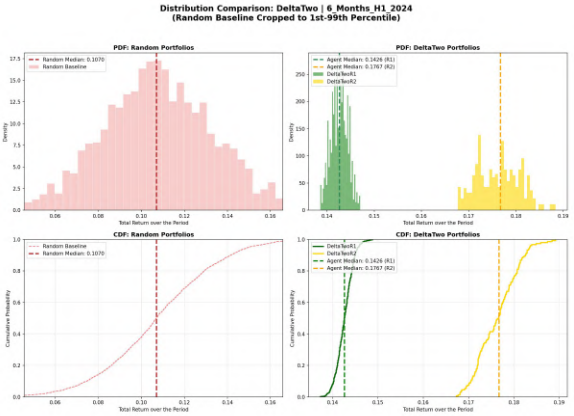
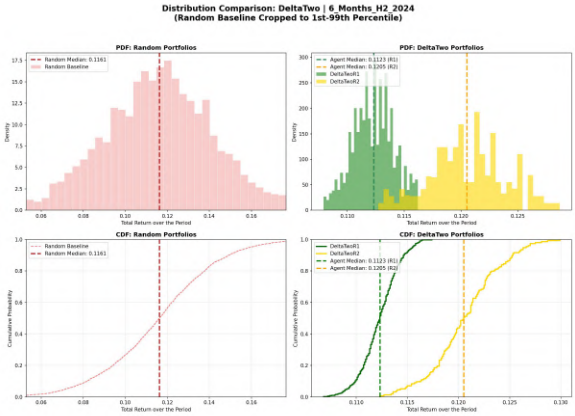


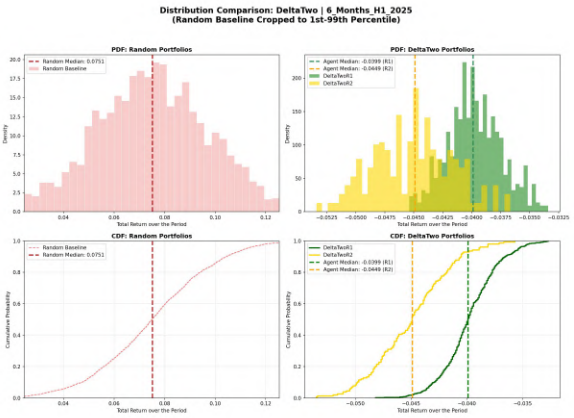
Figure C.39: Returns PDFs and CDFs for DeltaOne's stochastic runs vs random baseline - full run.



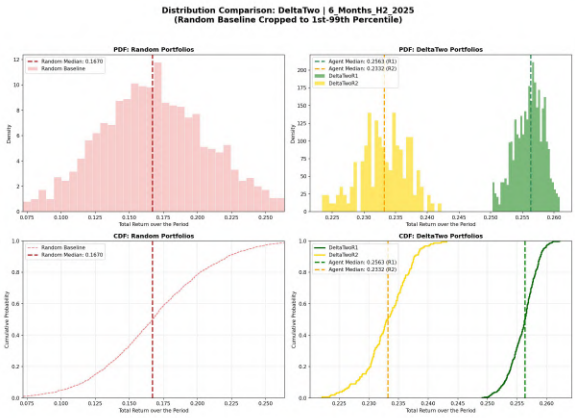
(a) H1 2024



(b) H2 2024

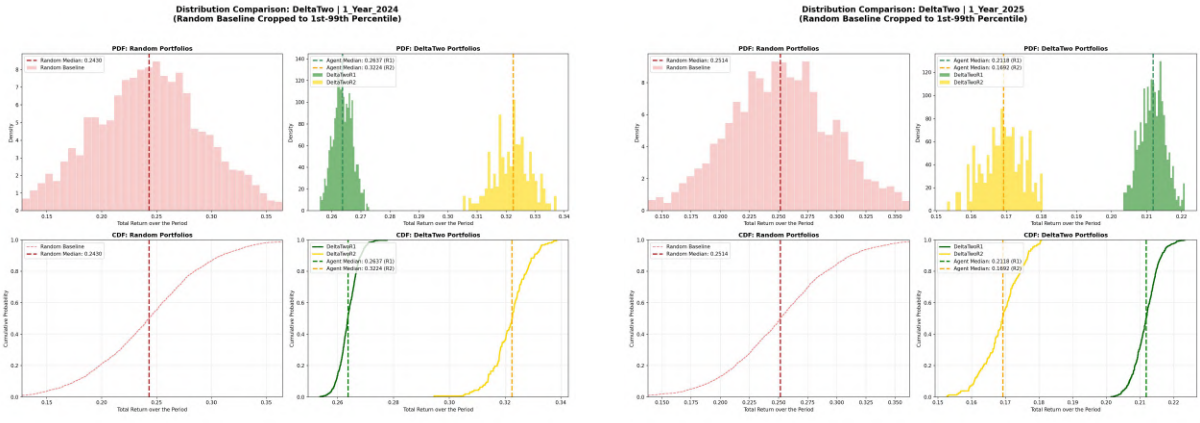


(c) H1 2025



(d) H2 2025

Figure C.40: Returns PDFs and CDFs for DeltaTwo's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.41: Returns PDFs and CDFs for DeltaTwo's stochastic runs vs random baseline - full years.

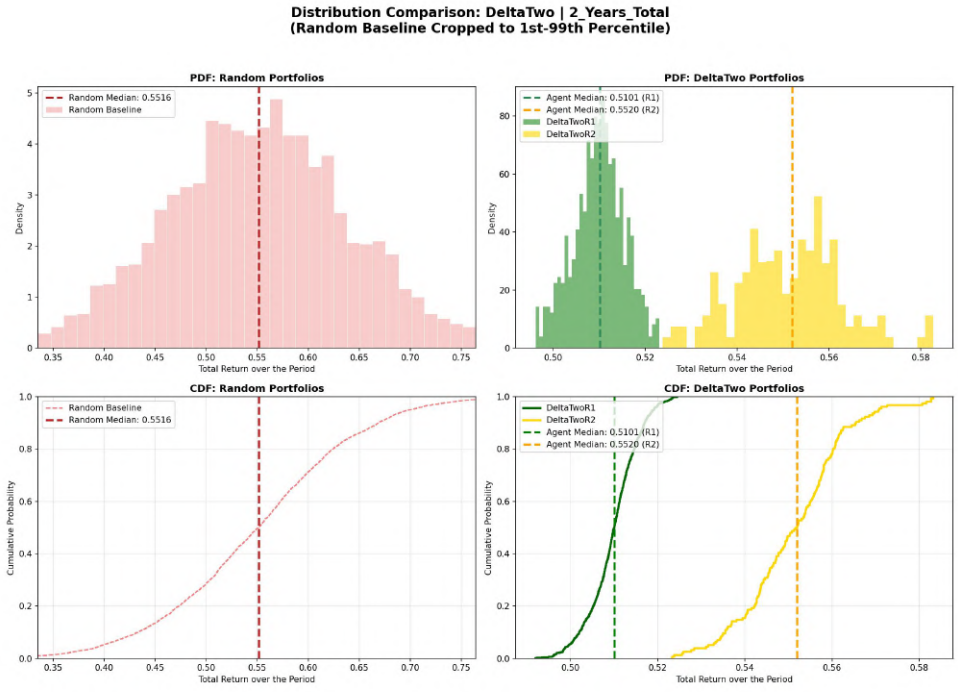
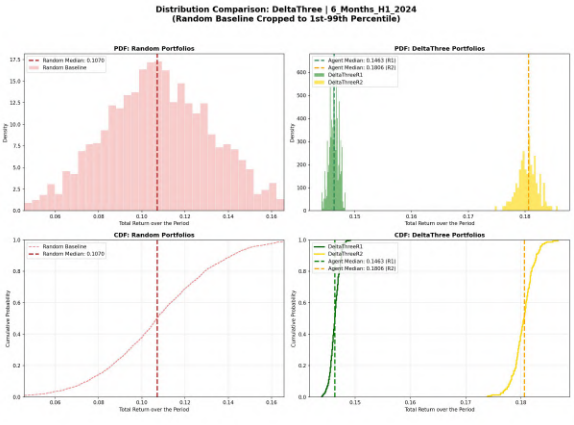
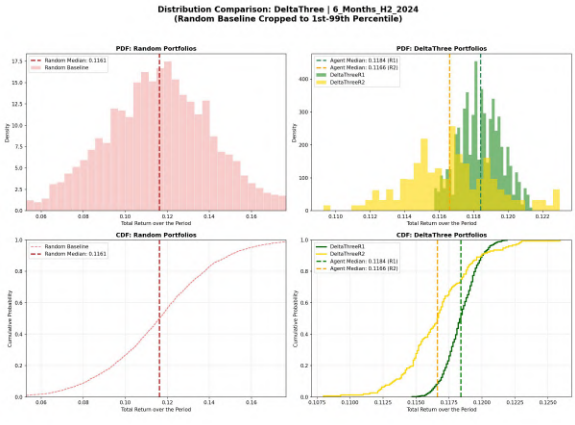


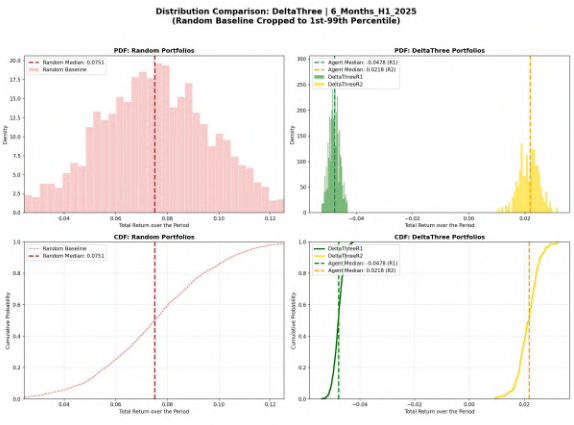
Figure C.42: Returns PDFs and CDFs for DeltaTwo's stochastic runs vs random baseline - full run.



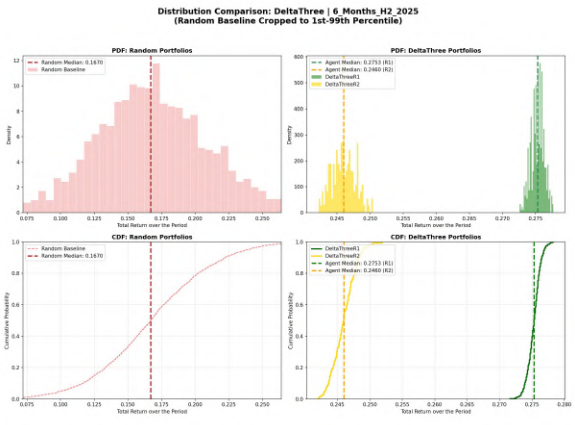
(a) H1 2024



(b) H2 2024

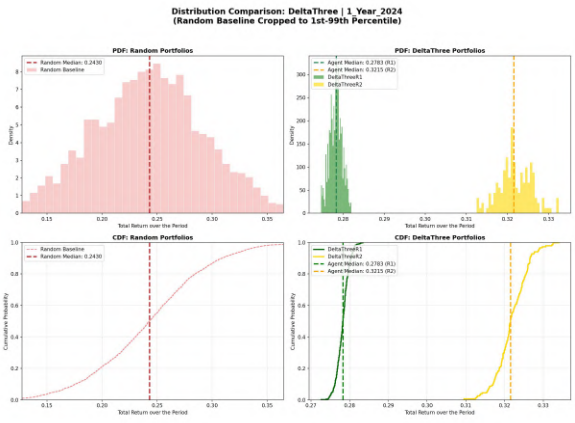


(c) H1 2025

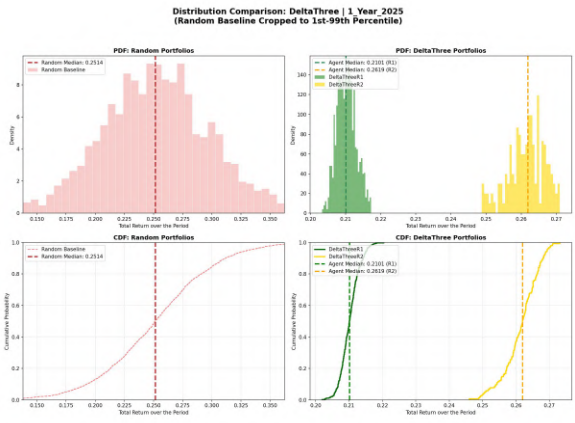


(d) H2 2025

Figure C.43: Returns PDFs and CDFs for DeltaThree's stochastic runs vs random baseline - half years.



(a) 2024



(b) 2025

Figure C.44: Returns PDFs and CDFs for DeltaThree's stochastic runs vs random baseline - full years.

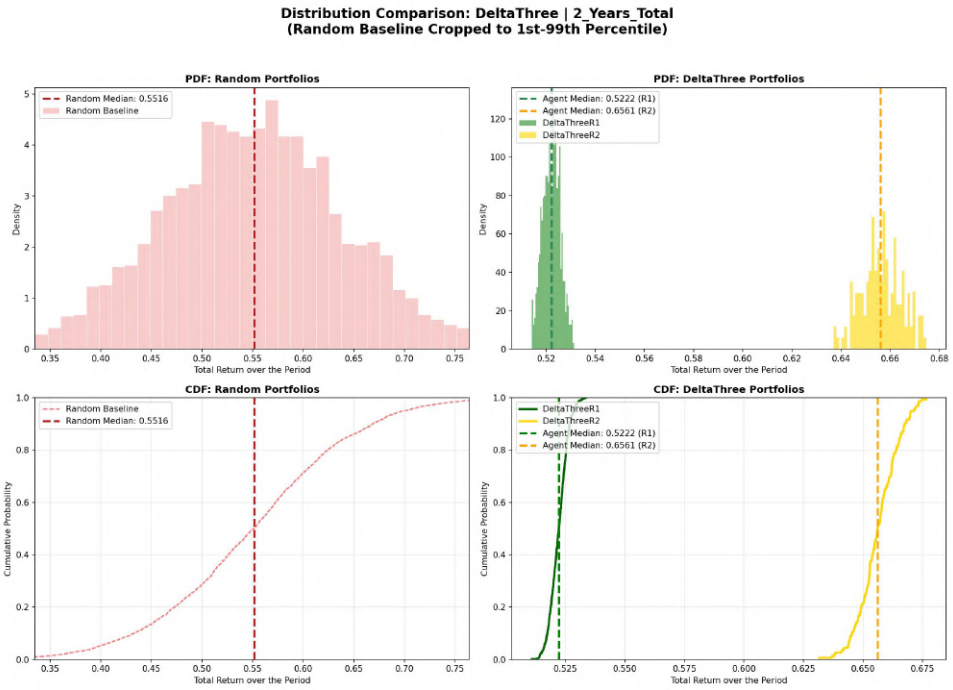
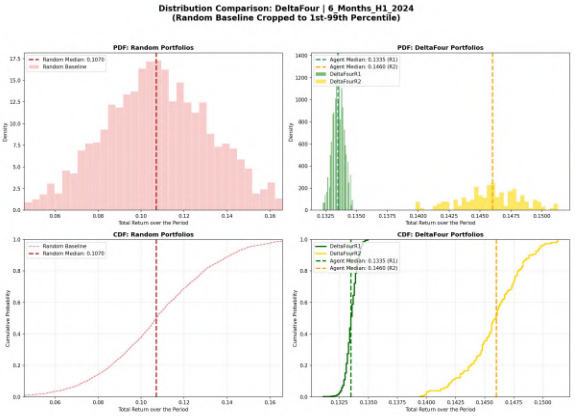
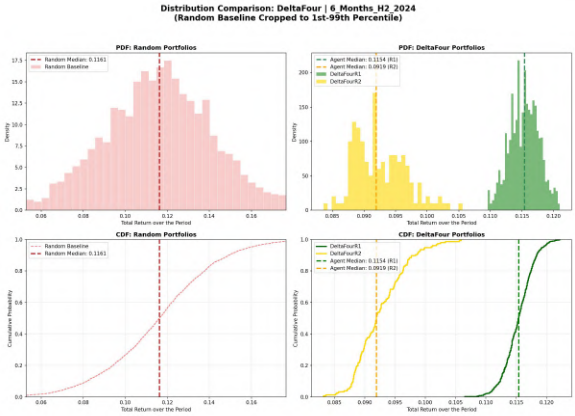


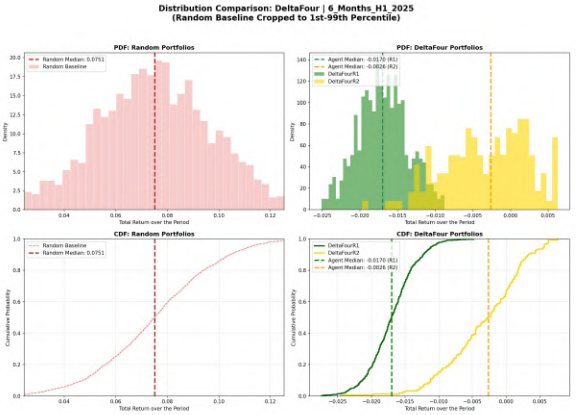
Figure C.45: Returns PDFs and CDFs for DeltaThree's stochastic runs vs random baseline - full run.



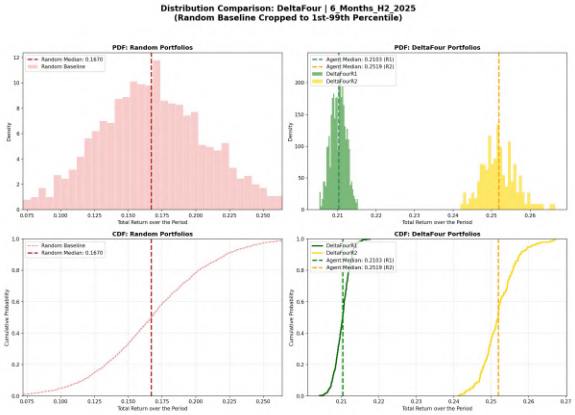
(a) H1 2024



(b) H2 2024

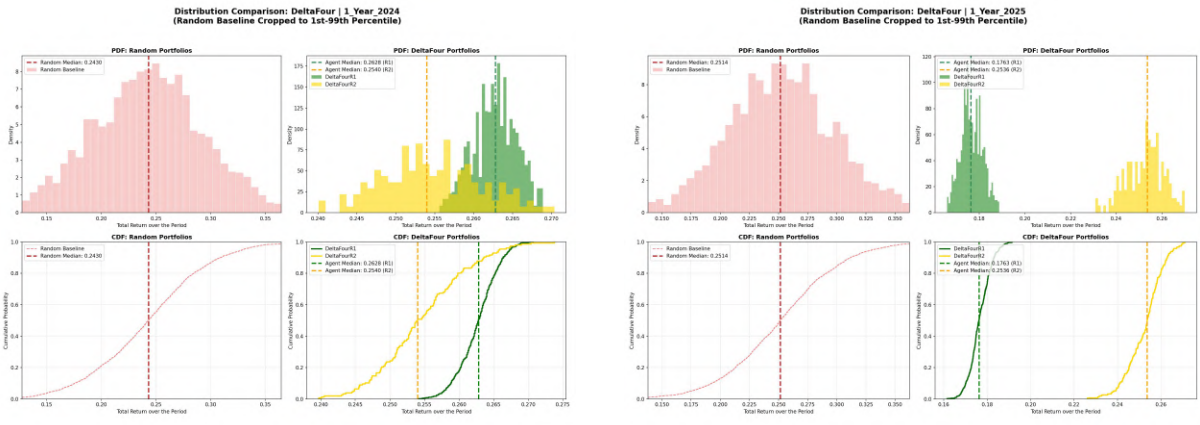


(c) H1 2025



(d) H2 2025

Figure C.46: Returns PDFs and CDFs for DeltaFour's stochastic runs vs random baseline - half years.



(a) 2024 (b) 2025

Figure C.47: Returns PDFs and CDFs for DeltaFour's stochastic runs vs random baseline - full years.

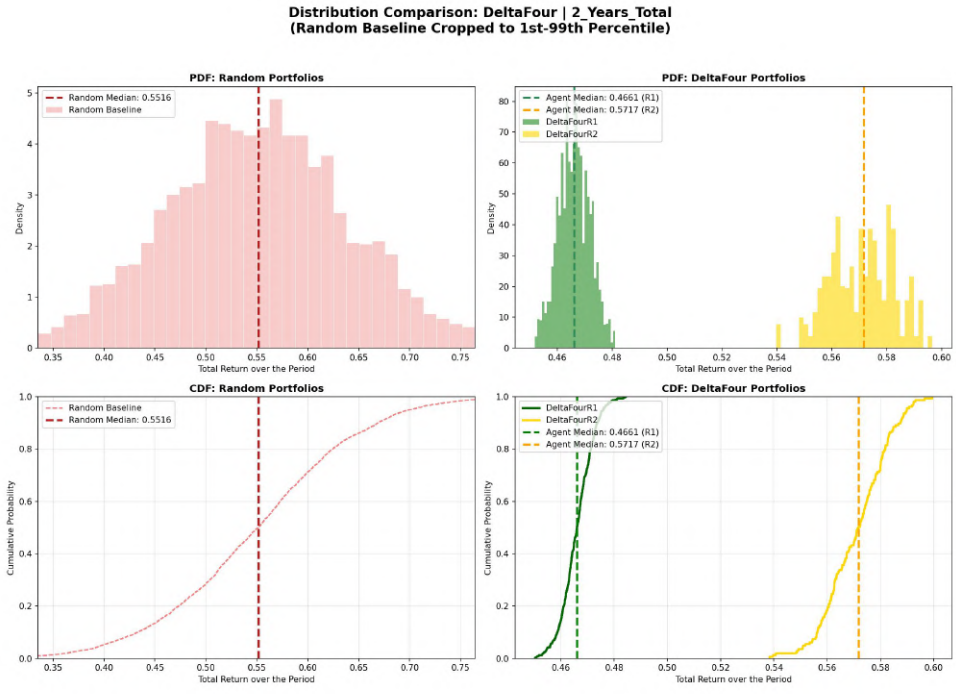


Figure C.48: Returns PDFs and CDFs for DeltaFour's stochastic runs vs random baseline - full run.

Appendix D

Acronyms and Symbols

Acronym	Meaning	Acronym	Meaning
AMH	Adaptive Market Hypothesis	MC	Monte Carlo methods
ANN	Artificial Neural Network	MDP	Markov Decision Process
BL	Black-Litterman model	ML	Machine Learning
CAPM	Capital Asset Pricing Model	MLP	Multilayer Perceptron
CDF	Cumulative Distribution Function	MW	Mann-Whitney U test
CNN	Convoluted Neural Network	PDF	Probability Density Function
CPU	Central Processing Unit	PG	Policy Gradient
DP	Dynamic Programming	PPO	Proximal Policy Optimization
DQN	Deep-Q Network	PSO	Portfolio Selection Oriented
DRL	Deep Reinforcement Learning	R1	Reward One
EBA	Error Backpropagation Algorithm	R2	Reward Two
ECDF	Empirical CDF	RAM	Random Access Memory
EMH	Efficient Market Hypothesis	RL	Reinforcement Learning
FSD	First-order Stochastic Dominance	RNN	Recurrent Neural Network
GAN	Generative Adversarial Network	RSI	Relative Strength Index
GMV	Global Minimum Variance	SAC	Soft Actor-Critic
GPI	Generalized Policy Iteration	SGD	Stochastic Gradient Descent
IRR	Isolated Return Reward	SMA	Simple Moving Average
KS	Kolmogorov-Smirnov test	SPY	Standard & Poor 500 index
LSTM	Long Short-Term Memory	TD	Temporal Difference learning
MAPS	Multi-Agent RL-based Portfolio management Systems	VG	View Generation

Table D.1: Table of Acronyms

Symbol	Meaning	Symbol	Meaning
S_t	State at timestep t	$\mathcal{S}$	State space
A_t	Action at timestep t	$\mathcal{A}(s)$	Action space
R_t	Reward at timestep t	$\mathcal{R}$	Reward support
G_t	Returns at timestep t	γ	discount factor
π	Policy	π^*	Optimal policy
$v_\pi(s)$	State-value function	$q_\pi(s,a)$	Action-value function
$\mathbf{w}$	Weight vector	$\nabla f(\mathbf{w})$	Gradient of f with respect to $\mathbf{w}$
η_i	Look-back window of sub-agent i	$\mathbf{H}$	Number of hidden layers in a feedforward MLP
h_i	Number of neurons in layer i	$\mathbf{W}$	Total number of weights in a feedforward MLP
α	Reward scaling parameter	c	Transaction cost in bps
w_*	Agent's weights vector	w_{eq}	Prior's weights vector

Table D.2: Table of Symbols

Bibliography

Aboussalah, A. M., Xu, Z., and Lee, C.-G. (2022). What is the value of the cross-sectional approach to deep reinforcement learning? *Quantitative Finance*, 22(6):1091 – 1111. Cited by: 23.

Amari, S.-I. (1972). Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on computers*, 100(11):1197–1206.

Aritonang, P. K., Wiryono, S. K., and Faturohman, T. (2024). Economic theory and machine learning integration in asset pricing and portfolio optimization: A bibliometric analysis and conceptual framework. *Journal of Computer Science*, 20(10):1291–1309.

Barua, R. and Sharma, A. K. (2023). Using fear, greed and machine learning for optimizing global portfolios: A black-litterman approach. *Finance Research Letters*, 58:104515.

Black, F. and Litterman, R. (1990). Asset allocation: combining investor views with market equilibrium. *Goldman Sachs Fixed Income Research*, 115(1):7–18.

Black, F. and Litterman, R. (1992). Global portfolio optimization. *Financial analysts journal*, 48(5):28–43.

Board of Governors of the Federal Reserve System (US) (2026a). 10-year treasury constant maturity minus 2-year treasury constant maturity (t10y2y). FRED, Federal Reserve Bank of St. Louis. Accessed: 2026-02-25.

Board of Governors of the Federal Reserve System (US) (2026b). 3-month treasury bill: Secondary market rate, discount basis (dtb3). FRED, Federal Reserve Bank of St. Louis. Accessed: 2026-02-25.

Cboe Global Markets (2026). Cboe volatility index: Vix (vixcls). FRED, Federal Reserve Bank of St. Louis. Accessed: 2026-02-25.

Choi, C. and Kim, J. (2024). Outperforming the tutor: Expert-infused deep reinforcement learning for dynamic portfolio selection of diverse assets. *Knowledge-Based Systems*, 294. Cited by: 16; All Open Access, Hybrid Gold Open Access.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Didisheim, A., Ke, S. B., Kelly, B. T., and Malamud, S. (2024). Apt or "aipt"? the surprising dominance of large factor models. Working Paper 33012, National Bureau of Economic Research.

Elman, J. L. and Zipser, D. (1988). Learning the hidden structure of speech. *The Journal of the Acoustical Society of America*, 83(4):1615–1626.

Engle, R., Ferstenberg, R., and Russell, J. (2006). Measuring and modeling execution cost and risk. *NYU Working Paper No. FIN-07-044*.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Gao, Y., Gao, Z., Hu, Y., Song, S., Jiang, Z., and Su, J. (2021). A framework of hierarchical deep q-network for portfolio management. In *ICAART (2)*, pages 132–140.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., and Su, J. (2020). Application of deep q-network in portfolio management. In *2020 5th IEEE International Conference on Big Data Analytics (ICBDA)*, pages 268–275. IEEE.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Ha, M., Yang, Y., and Wang, C. (2017). A portfolio optimization model for minimizing soft margin-based generalization bound. *Journal of Intelligent Manufacturing*, 28(3):759–766.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

He, G. and Litterman, R. (2002). The intuition behind black-litterman model portfolios. *Available at SSRN 334304*.

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

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Idzorek, T. (2007). 2 - a step-by-step guide to the black-litterman model: Incorporating user-specified confidence levels. In Satchell, S., editor, *Forecasting Expected Returns in the Financial Markets*, pages 17–38. Academic Press, Oxford.

Ko, H. and Lee, J. (2025). Portfolio management transformed: An enhanced black–litterman approach integrating asset pricing theory and machine learning: H. ko, j. lee. *Computational Economics*, 66(5):3841–3887.

Kolm, P. N., Ritter, G., and Simonian, J. (2021). Black-litterman and beyond: The bayesian paradigm in investment management. *The Journal of Portfolio Management, to appear*.

Kolmogorov, A. N. (1957). On the representations of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Dokl. Akad. Nauk USSR*, volume 114, pages 953–956.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Kumlungmak, K. and Vateekul, P. (2023). Multi-agent deep reinforcement learning with progressive negative reward for cryptocurrency trading. *IEEE Access*, 11:66440 – 66455. Cited by: 12; All Open Access, Gold Open Access, Green Open Access.

Lee, J., Kim, R., Yi, S.-W., and Kang, J. (2020). Maps: multi-agent reinforcement learning-based portfolio management system. *arXiv preprint arXiv:2007.05402*.

Lo, A. W. (2004). The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *Journal of Portfolio Management, Forthcoming*.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1):77–91.

Martin, R. A. (2021). Pyportfolioopt: portfolio optimization in python. *Journal of Open Source Software*, 6(61):3066.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mikriukov, D., Sun, R., and Jiang, Z. (2025). Deep reinforcement learning-based portfolio optimization with black-litterman model under elliptical distributions. In *International Conference on Intelligent Computing*, pages 273–284. Springer.

Min, L., Dong, J., Liu, D., and Kong, X. (2021). A black-litterman portfolio selection model with investor opinions generating from machine learning algorithms. *Engineering Letters*, 29(2):710–721.

Minsky, M. and Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104.

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

Nakano, K. (1971). Learning process in a model of associative memory. In *Pattern Recognition and Machine Learning: Proceedings of the Japan—US Seminar on the Learning Process in Control Systems, held in Nagoya, Japan August 18–20, 1970*, pages 172–186. Springer.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703.

Punyaleadtip, K., Kantavat, P., Nimmanunta, K., and Kijsirikul, B. (2024). Black–litterman portfolio management using the investor's views generated by recurrent neural networks and support vector regression. *Journal of Financial Data Science*, 6(1).

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

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rosenblatt, F. (1960). Perceptual generalization over transformation groups. *Self Organizing Systems*, pages 63–96.

Rosenblatt, F. et al. (1962). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 55. Spartan books Washington, DC.

Rumelhart, D. E., Hinton, G. E., McClelland, J. L., et al. (1986). A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(45-76):26.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report.

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

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Sharpe, W. F. (1998). The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 3(3):169–85.

Smirnov, N. V. et al. (1939). On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bull. Math. Univ. Moscou*, 2(2):3–14.

Sun, Q., Wei, X., and Yang, X. (2024a). Graphsage with deep reinforcement learning for financial portfolio optimization. *Expert Systems with Applications*, 238. Cited by: 48.

Sun, R., Stefanidis, A., Jiang, Z., and Su, J. (2024b). Combining transformer based deep reinforcement learning with black-litterman model for portfolio optimization. *Neural Computing and Applications*, 36(32):20111–20146.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., Cola, G. D., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. (2025). Gymnasium: A standard interface for reinforcement learning environments.

U.S. Bureau of Labor Statistics (2026a). Consumer price index for all urban consumers: All items (cpiaucsl). FRED, Federal Reserve Bank of St. Louis. Accessed: 2026-02-25.

U.S. Bureau of Labor Statistics (2026b). Unemployment rate (unrate). FRED, Federal Reserve Bank of St. Louis. Accessed: 2026-02-25.

Vasilevich, S. I. (2025). Black-litterman portfolio optimization using machine-learning, deep learning and reinforcement learning algorithms. *Deep Learning and Reinforcement Learning Algorithms (August 18, 2025)*.

Wang, F., Li, S., Niu, S., Yang, H., Li, X., and Deng, X. (2025). A survey on recent advances in reinforcement learning for intelligent investment decision-making optimization. *Expert Systems with Applications*, 282.

Wang, J., Zhang, Y., Tang, K., Wu, J., and Xiong, Z. (2019). Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1900–1908.

Wang, Z., Huang, B., Tu, S., Zhang, K., and Xu, L. (2021). Deeptrader: a deep reinforcement learning approach for risk-return balanced portfolio management with market conditions embedding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 643–650.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Xu, K., Zhang, Y., Ye, D., Zhao, P., and Tan, M. (2021). Relation-aware transformer for portfolio policy learning. In *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pages 4647–4653.

Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. (2020). Deep reinforcement learning for automated stock trading: An ensemble strategy. Cited by: 183.

Zhu, E. and Yen, J. (2024). Enhancing portfolio optimization with transformer-gan integration: a novel approach in the black-litterman framework. *arXiv preprint arXiv:2404.02029*.