ERASMUS UNIVERSITY ROTTERDAM

ERASMUS SCHOOL OF ECONOMICS

Master Thesis Quantitative Finance

November 27, 2023

Towards Interpretable High-Dimensional Continuous Reinforcement Learning

Applied to Goal-Based Wealth Management

Author:
Gert Lek (657669)

Supervised by:

Dr. Andrea Naghi (Erasmus University Rotterdam)
Dr. Hens Steehouwer (Ortec Finance)
Afrasiab Kadhum (Ortec Finance)

Second Assessor:

Dr. Maria Grith (Erasmus University Rotterdam)

Abstract

This study addresses the pressing challenge of improving interpretability in high-dimensional continuous Reinforcement Learning (RL) models, using the specific context of goal-based wealth management as a case study. We provide an analysis of how principles from interpretable RL can be effectively applied to this nuanced domain while also highlighting the obstacles encountered in real-world implementation. Our work is segmented into three key areas: 1) We introduce the meta-controller framework, an advancement of existing lazy frameworks, which structure the RL architecture, tailored for the complexities of continuous spaces. 2) Building on this, we develop the regime change framework, an enhancement of the lazy framework that ensures a more intelligible, consistent and coherent action sequence over time. 3) We employ a robust Shapley value method that accounts for feature dependencies present within RL, thereby providing a nuanced insight into the agent's decision-making process. Through an empirical exercise, our proposed explainability methods verify the utility of our approaches in interpreting the decision process of RL agents in high-stakes, complex settings like goal-based wealth management. Our study substantiates that in high-dimensional continuous RL, the strategic application of structure and simplicity offers the most promising pathway for improving interpretability without sacrificing utility in performance.

Contents

Glossary

ANN Artificial Neural Network

DDPG Deep Deterministic Policy Gradient

DQN Deep Q-Networks

DDQN Double Deep Q-Networks

TD3 Twin Delayed Deep Deterministic Policy Gradient

EM Emerging Markets

EW Equity World

FI Fixed Income

FIHY Fixed Income High Yield

GBWM Goal Based Wealth Management

HF Hedge fund of funds

MDP Markov Decision Process

OOS Out-Of-Sample

PPO Proximal Policy Optimization

RE Real Estate

RL Reinforcement Learning

SHAP SHapley Additive exPlanations

TD Temporal Difference

DSG Dynamic Scenario Generator

1 Introduction

Although Reinforcement Learning (RL) is revolutionising complex problem-solving, its potential for a greater significant societal impact remains bottle-necked by the challenge of making these algorithms interpretable. RL involves learning from interactions to achieve a goal. In an era of increasing computational power and data availability, RL is critical in addressing real-world challenges, ranging from healthcare and finance to autonomous vehicles.

An RL *agent* interacts with its environment through a sequence of steps. The decisions made by this agent are called *actions*. The environment manifests itself for each time step in the form of *states*, which form the basis for the chosen action. Actions are associated with *rewards* that evaluate the choices made by the agent. The agent chooses actions from states by its *policy*. Each RL problem uniquely defines different environments and numerous reward functions. In a broader sense, RL is a computational approach to learning from interaction. This interaction occurs with the environment, some of which the agent cannot control. The agent seeks to maximise reward over a period of time. To achieve this, the agent trades off exploration with exploitation. This interactive trade-off is what differentiates RL from (un)supervised learning. Recent developments in RL show its promise but also reveal its weaknesses. Large-scale RL implementations are currently being deployed by notable companies such as OpenAI and Tesla. For example, ChatGPT is tuned partly by a certain branch of RL (Radford et al., 2019). Another example is autonomous driving models trained with RL (Chen et al., 2019). Finance has been at the forefront of solving complex and dynamic optimisation problems for quite some time; thus, applying RL in this field is not far-fetched. Advanced techniques such as RL allow us to relax assumptions and solve more complex problems. Therefore, RL research in finance has been growing in the areas of pricing, hedging or portfolio optimisation (Kolm & Ritter, 2020). However, the need for interpretation increases as complex models are implemented for a wider range of applications. To illustrate, notable criticism has been made against the black-box nature of autonomous driving and virtual assistants (Lugano, 2017).

Our inability to effectively explain and justify RL agents' decisions makes it hard to deploy them. Diving deeper, RL interpretability research lacks the same variety and depth as Machine Learning (ML) interpretability (Alharin et al., 2020). RL is mostly uninterpretable, as the agents usually consist of Artificial Neural Networks (ANNs) and complex training algorithms. This black-box aspect of RL hinders its acceptance, but many bold attempts have been made in current research to solve this problem; for example, a widely-used method in interpreting RL algorithms is the use of saliency maps to highlight important features in each state (Greydanus et al., 2017 and Gupta et al., 2019). Further, there is promising research in creating fully interpretable RL techniques for simple environments, such as those described in Verma et al. (2018). Focusing on the scope of this paper, we deal with a particular subset of RL interpretation: high-dimensional continuous RL. High-dimensional continuous state and action spaces pose a significant challenge for RL algorithms and an even greater one when interpreting them (Dulac-Arnold et al., 2021). The existing research primarily focuses on discrete and low-dimensional state spaces. For instance, popular interpretable RL surveys such as Qing et al. (2022) and Vouros (2022) do not extensively cover continuous action or state spaces. The state-of-the-art methods often rely on decision rules or feature importance measures better suited for discrete actions. Interpretability methods that depend on human-understandable rules or visualisations become impractical or ineffective when faced with many dimensions. These limitations in the research give rise to a need for the development of novel approaches that explicitly target high-dimensional continuous action and state spaces. The ob-

jective of this paper is to interpret, clarify, justify and rationalise agents' behaviour in high-dimensional continuous RL by devising and applying interpretability and explainability methods in this complex field.

We apply these methods to a specific setting in high-dimensional continuous RL: (scenario-based) Goal-Based Wealth Management (GBWM). Brunel (2011) notes that GBWM can be complex as there are multiple objectives, and the dynamic nature of these requires continuous monitoring. In GBWM, we seek an asset allocation strategy that maximises the probability of achieving an investor's goals. This goal may be to accumulate a specific amount of wealth within a certain period. Investors need to trust the models with their wealth, but this trust is difficult to establish in a black-box setting. For this reason, a more interpretable RL architecture is advantageous for GBWM.

Ortec-Finance, a company specialising in financial decision-making, uses scenario-based RL to solve the GBWM problem. Previous research conducted by Ortec-Finance explores different RL approaches to optimise the goal performance for the GBWM problem (Kadhum, 2022). We will use the author's algorithm as an interpretability baseline but then depart from their implementation and focus purely on interpretation and explainability. For this research, we compare several RL interpretability techniques with different algorithms for the GBWM task. This is done to achieve more interpretability in the agent's decision-making process. For the agent's decision-making process to be interpretable, all of its components must be intelligible, for which we will aim in this research.

The limited amount of research complicates the task of creating an interpretable framework. We aim to fill this knowledge gap in the literature by investigating, implementing and extending interpretability techniques that can be applied to high-dimensional continuous RL (and therefore, GBWM). This makes the research relatively novel and meaningful, as little to no research deals with the broad problem of interpreting and explaining high-dimensional continuous RL algorithms. Broadly, we contribute to the ongoing effort to improve the interpretability of RL models, which is coined as one of the ten grand challenges in ML (Rudin et al., 2022).

From a methodological standpoint, our contributions are three-fold. First, we provide a novel implementation of the lazy framework for improved interpretability (Jacq et al., 2022) in continuous action spaces, coined the meta-controller framework. In Jacq et al. (2022), this idea was hinted at but was not explored due to its complexity. Second, we devise the regime change framework, which builds upon the meta-controller framework, regularising and fixing its flaws regarding the continuity of actions. This framework grants temporal consistency and, therefore, intelligibility in the actions needed in our complex GBWM setting. Third, this structured RL architecture allows us to use a feature dependency robust Shapley value approach to explain an agent's decision process, combining these approaches in a novel dimensionality-independent way.

As for the empirical contributions of this paper, we display the difficulties when aiming for an applicable RL agent, defined via the concept of a rational agent. Further, we describe and implement explainability techniques using Shapley values that deal with the complexities of understanding agents' decision-making processes in the high-dimensional continuous RL setting. Furthermore, we demonstrate the limitations of traditional Shapley value methods in RL and explore a more robust approach. These techniques are all applied in a case study that attempts to interpret an RL agent in GBWM, providing practical insights into these techniques and displaying how they improve interpretability and explainability.

Last, from a practical standpoint, this research interests professionals in the financial industry involved in applying RL within finance, specifically for GBWM. Asset management strategies require high trust in the underlying models. More interpretable architectures for RL models can address this by providing investors with a better understanding of the agent's decision-making process and increasing their confidence in RL models. Thus, the results from this research can aid the development of more interpretable and trustworthy RL models in finance, contributing to the continuing endeavour to improve financial decision-making.

Regarding the structure of this paper, we initially guide the reader through the field of interpretability in high-dimensional continuous RL. Starting with RL prerequisite knowledge in Section 2, whereafter in Section 2.2, we provide an overview of how interpretable RL techniques fit with the high-dimensional continuous setting. Next, the methodology and results are partitioned into two categories: "Interpretability", which focuses on the intrinsic structural techniques within reinforcement learning, and "Explainability", which addresses post-hoc (visualisation) methods to explain the now more interpretable agents. To elaborate, the discussion of our financial (scenario) data precedes the methodology in Section 4, where we introduce broad concepts, structural interpretability techniques and post-hoc explanation methods tailored to the high-dimensional continuous setting. Section 5 applies these methods to the GBWM setting and examines the resulting outcomes. Finally, we summarise these results in the conclusion and discuss the practical implications, limitations and further research recommendations.

2 RL Prerequisites

This section covers the necessary RL background knowledge for this paper. These concepts require some understanding of RL and its associated mathematical fields. We first give the reader a broad walkthrough of the relevant advances in RL, then focus on important algorithms used in this paper. Lastly, in Section 2.2, we turn to interpretability in (high-dimensional continuous) RL.

Each RL problem uniquely defines different environments and reward functions. RL environments can be formulated as a Markov Decision Process (MDP) (Bellman, 1957). The MDP comprises a set of states S, a set of actions A, a transition probability function T regarding the subsequent state transition given the current state and action, and a reward function R that assigns a reward for each state-action pair. The following notation is borrowed from Sutton & Barto (2018). A policy π has state and state-action value functions v_π and q_π, respectively. We define $V_\pi(s) = \mathbb{E}\pi[G_t \mid S_t = s]$ and $Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$ for $s \in S, a \in A$ as the expectations of these functions, where G_t is the total discounted return. These assign an expected return to each state or state-action pair if the agent proceeds with policy π. Here, the return is a function of the future rewards. The core assumption of an MDP is the Markov property (Bellman, 1957), which notes that future states are solely dependent on the current state and action, disregarding the history of past states or actions. This property renders MDPs particularly beneficial for modelling decision-making in scenarios where outcomes are partially random and partially controlled by a decision-maker. Therefore, throughout this paper, similar to the existing literature, we formalise RL concepts as MDPs.

Next, we discuss various properties which characterise RL algorithms, starting with value-based and policy-based methods. Value-based methods learn the value function (the expected future reward) for each state or state-action pair. A popular early version of such a method is Q-Learning (Watkins &

Dayan, 1992), which learns the state-action value (Q) function $Q_\pi(s, a)$ iteratively. Conversely, policy-based methods directly learn the policy function, which maps each state to an action, eliminating the need for a value function. Each method has its strengths and weaknesses. Value-based methods are generally easier to comprehend due to their learning of the value function. Policy-based methods are often more effective in exploration and less sensitive to hyperparameters (Arulkumaran et al., 2017).

Next, in Actor-Critic algorithms (Konda & Tsitsiklis, 1999), value and policy-based methods are combined. Here the actor is responsible for selecting actions, and the critic guides the actor by estimating the value function. Both the actor and critic are commonly represented by neural networks as function approximators.

The policy that the agent learns can be further differentiated: an agent's policy to interact with its environment can be the same as the one that it's learning (on-policy) or different (off-policy). On-policy methods learn the value of the agent's current policy to make decisions. Off-policy methods, like Q-Learning or DDPG, can learn the value of a policy differently from the one used to make decisions. This makes them more flexible and data-efficient as they learn from the experiences of other policies or previous policies.

2.1 RL concepts

We proceed by diving deeper into specific RL concepts that are of importance to this paper. Function approximators, such as neural networks are crucial to modern RL. It is common for neural networks to approximate the value functions $V_\pi(s), Q_\pi(s, a)$ or the policy $\pi(s)$. For simple environments with small cardinality state-action spaces, tabular methods can store and update values for each state-action pair. However, more sophisticated function approximations are required for more complex problems such as those involving continuous state and action spaces. Neural networks are able to handle high-dimensional input data and have the universal function approximation property (Hornik et al., 1989), which is leveraged in proving optimality for their associated MDPs.

Let us briefly consider the different RL algorithms used in this paper. Proximal Policy Optimisation (PPO) (Schulman et al., 2017) is a variant of the policy gradient method which performs updates using a sample of data collected under the current policy, similar to vanilla policy gradient methods (Williams, 1992). The actor is responsible for the policy, and the critic evaluates the state value function V_π. Meanwhile, the algorithm directly optimises the policy itself and follows the same policy to generate samples. Therefore, PPO is characterised as an actor-critic, on-policy and policy-based algorithm. PPO utilises a specialised objective function that penalises large policy updates, aiming to combine the benefits of first-order policy gradient methods with the robustness and reliability of a trust region method (Conn et al., 2000). PPO has been extensively used in numerous RL tasks due to its robust performance.

The algorithm Deep-Q-Networks (DQN) (Mnih et al., 2015) is a type of RL algorithm that combines the Q-Learning algorithm with deep neural networks as function approximators. Therefore, DQN can handle high-dimensional state and action spaces but is designed purely for discrete state and action spaces. DQN lacks learning stability and has an inherent overestimation bias for the Q approximator. Double-DQN (DDQN) (van Hasselt et al., 2015) employs a separate network for action selection and evaluation to enhance learning stability. The network for action selection when updating the networks is updated more slowly. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) is a further extension of DQN, which uses a neural network to represent the policy and noise injection in the action

space as exploration. This allows DDPG to handle continuous action and state spaces relatively well. Next, the Twin-Delayed DDPG (TD3) algorithm is an advanced version of the DDPG algorithm, first introduced in Fujimoto et al. (2018). TD3 is characterised as an actor-critic, off-policy and value-based algorithm. As we use TD3 extensively in this paper, let us briefly discuss the 3 extensions TD3 implements over DDPG:

1. Dual Q-functions: Following the principles of DDQN, TD3 employs two Q-functions to mitigate overestimation bias, a common issue in Q-learning methods where the algorithm overestimates the Q-values of the actions, leading to sub-optimal policies. This is a significant departure from DDPG, which uses only one Q-function.

2. Delayed policy updates: TD3 updates the policy and target networks less frequently than the Q-function. This delay reduces the variance of the policy updates and enhances learning stability, hence the "delayed" in TD3.

3. Target policy smoothing: TD3 incorporates noise into the target action, making the policy update more robust against errors in the Q-function approximation. This form of regularisation helps prevent the policy from over-fitting to the noise in the Q-function estimates.

In essence, TD3 builds upon DDPG, introducing modifications that enhance performance and stability, particularly in continuous control tasks. TD3 estimates a value function, which makes it easier to interpret. This will prove to be useful in our proposed techniques.

2.2 Interpretability in high-dimensional continuous RL

This section aims to give the reader a thorough understanding of the various RL interpretability techniques that are available. Moreover, we explore how these techniques relate to the continuous high-dimensional setting to provide a comprehensive approach to interpretability in this complex domain. First, we briefly discuss the general concept of interpretability in machine learning. Interpretability and explainability are often used synonymously. For this research, interpretability is a quality of the RL model that makes agents' decisions easier to understand. Explainability aims to provide insight into the decisions of a trained model, which may be uninterpretable. Explainability performs additional post-hoc steps to offer an explanation that clarifies the model's decision.

This paper addresses a tough challenge: interpretability in high-dimensional continuous action and state space RL. High-dimensional refers to the high-dimensional state and action space, which implies that the function approximations have high-dimensional inputs and outputs. Many interpretability techniques fail in this environment. To arrive at a (more) interpretable RL algorithm, we can make the inputs within the algorithm and/or make the agent's decision-making process interpretable. We start with how to make the inputs interpretable, an overview of the broadly available techniques for interpretable inputs is partitioned in Figure 1. Next, we introduce these methods and discuss how they relate to the high-dimensional continuous RL setting.

1. The structured approach attempts to create a structured environment where the agent can reason and learn in an interpretable setting. For example, by training agents to make decisions using first-order logic. However, naturally one has to work in a simple environment for this to work, and simplifying

our complex environment to this extent is unreasonable. These methods have trouble dealing with the complex reasoning real-life environments require.

2. Another technique for improving interpretability is using symbolic representations. In high-dimensional continuous RL, actions and states are often represented as real-valued vectors, which can be unintuitive for human understanding. Symbolic representations aim to encode the actions and states in a more understandable form, e.g. in the form of symbols or rules. This approach can benefit high-dimensional continuous action and state spaces, allowing for a better understanding of the patterns and relationships in the space. However, finding a symbolic representation that preserves the complex interactions of a high-dimensional space can be challenging. Here, assessing the quality of symbols and quantifying the concept of the 'meaning' of symbols (symbol grounding problem, Harnad, 1990) both pose significant hurdles.

3. Hierarchical approaches divide an RL problem into different levels of abstraction. This can be especially useful in high-dimensional continuous state and action spaces because the actions or states can usually be broken down into manageable and more interpretable sub-parts. For example, an agent must control numerous joints and motors in robot control. A hierarchical approach could divide this into higher-level actions, such as "grab object", and lower-level actions, such as the specific motor controls to achieve this. This hierarchical division can facilitate interpreting what the agent is trying to achieve at different levels of abstraction. Complex settings such as ours benefit significantly from this, as some parts of our state and action space have inherent meanings, allowing for interpretable levels of abstraction.

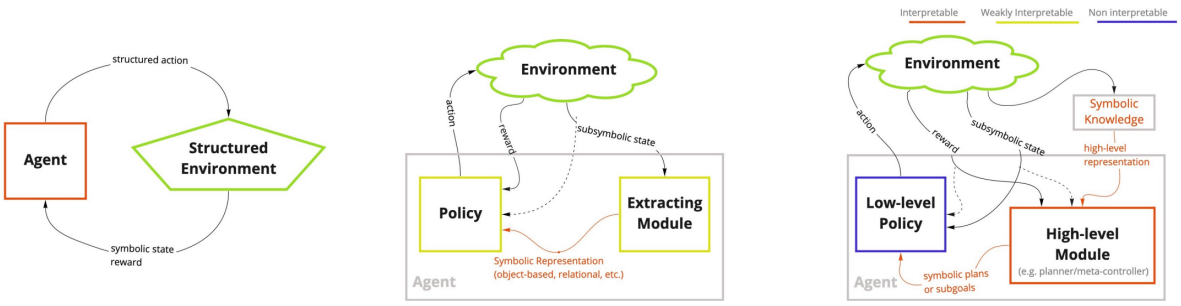


Figure 1: Overview of interpretability techniques for interpretable inputs in RL (Glanois et al., 2022). From left to right: structured approach, symbolic representation and hierarchical approach.

Upon recognising the (in)effectiveness of these three approaches in high-dimensional continuous RL settings, we continue with making the agent's decision-making process more interpretable. Following a categorisation of interpretability techniques for agent's decision making by Glanois et al. (2022), there are four general categorisations: direct approach, indirect approach, architectural inductive bias, and intelligibility-driven regularisation. These are illustrated in Figure 2. It is essential to grasp how these categories fit into high-dimensional continuous RL. Therefore, we will review them individually and explain their applicability and limitations for this setting.

1. With the direct approach, we attempt to design inherently interpretable models. These models are often simplistic, such as linear models, decision trees or rule lists. However, in high-dimensional continuous action spaces, the complexity of the environment and the interactions within it may be too

high for these simple models to capture them accurately. As a result, the interpretability of the model comes at the cost of performance since we have to force (discrete) low-dimensional spaces. For example, an interpretable regression tree agent would have > 17 inputs and > seven outputs in our setting, from which it would be infeasible to create an interpretable regression tree, thus sacrificing performance for no additional interpretability.

2. The indirect approach attempts to train a complex model (like a neural network) and then approximate it with a simpler, more interpretable model. While this can provide some level of interpretability, the approximation may not accurately capture the behaviour of the original model, especially in high-dimensional continuous action spaces. For example, one of the most common and effective methods is Linear Model U-Trees (LMUT) (Liu et al., 2018). This method does achieve good results in approximating Q-Functions, but for example; the authors establish that it fails in high-dimensional environments.

3. The architectural inductive bias deals with designing the architecture of a model in a way that encourages interpretability. For example, using attention mechanisms to highlight important features. These techniques rely on some data format, such as image/text/relational data, which they then leverage for interpretability.

4. Intelligibility-driven regularisation involves adding a regularisation term to loss functions that encourage interpretability. For example, a penalty for erratic behaviour to improve smoothness. While this can promote the model to be more interpretable, it does provide an interpretability guarantee. This method can make the architecture more robust and thus more reliable and interpretable.

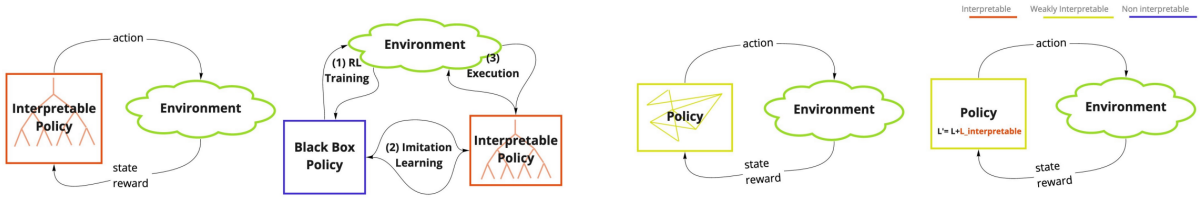


Figure 2: Overview of interpretability techniques for agent decision-making in RL (Glanois et al., 2022). From left to right: direct approach, indirect approach, architectural inductive bias and intelligibility-driven regularisation.

Tackling the challenge of interpretability in high-dimensional continuous action-space RL appears to be an intricate task. We notice that each technique comes with its trade-offs and limitations. The key is devising an effective strategy that combines these techniques, capitalising on their strengths. One has to be realistic in acknowledging that most of these methods would not achieve the desired results in our setting.

A strategy that integrates the hierarchical approach with intelligibility-driven regularisation looks promising in its simplicity and applicability. These two methods are characterised by making complex environments more interpretable by making parts of the algorithm intelligible, while other approaches focus on making simple environments fully interpretable. Additionally, if the data permits it, a form of architectural inductive bias is suitable for effectively leveraging this data in favour of interpretability.

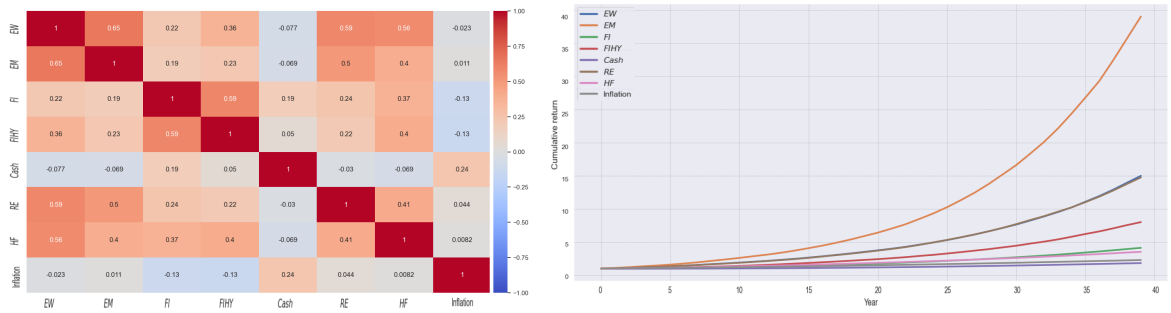
In addition to interpretability techniques, this paper will utilise post-hoc explainability methods (Qing et al., 2022) to analyse the decision-making process of the agent. In this paper, we will consistently use these methods to gain insight into the effect of the aforementioned interpretability techniques.

Simply put, this paper represents a crucial first step in uncovering the strategies that work in real-world high-dimensional continuous RL settings.

3 Data

RL requires a large amount of data to converge; therefore, real financial data is insufficient for training a robust agent with a long-term goal. Ortec-Finance uses a Dynamic Scenario Generator (DSG), which we use to generate 15,000 possible future economic scenarios covering 40 years from 2022 onwards (Steehouwer, 2005). The scenarios consist of seven asset categories and annual inflation. We also have a historical data set from 1991 to 2021 for initialisation and testing. The DSG uses over 600 financial/economic time series as input and decomposes them into numerous factors. These are inputs to a dynamic factor model to generate time series. Our return series consists of two equity series, two fixed-income series, cash, real estate and a hedge fund index.

We briefly investigate the data characteristics and plot the scenario return correlation matrix and the cumulative return of the scenario instruments in Figure 3. The correlation matrix is computed with Kendall's tau (Kendall, 1938) to capture nonlinear dependencies since our models can also capture these. We notice high correlations within our scenario data in the correlation matrix of Figure 3a. Certain asset returns appear highly correlated, such as different equity funds (EW, EM, HF) and bond funds (FI, FIHY). This is expected, as any risky assets share some risk factors. By Figure 3b, the long-term equity outperformance in our data is apparent. These observations share similarities to established economic stylised facts regarding investment instruments.



(a) Matrix showing Kendall's tau correlations among different scenarios.

(b) Mean cumulative dollar return for seven asset classes and inflation across scenarios.

Figure 3: Descriptive visualisation of full scenario set.

Next, we inspect the return summary statistics displayed in Table 1 and detect that, by and large, higher risk (volatility) appears to be rewarded with a higher return, both in the short and long term. Hedge fund of funds and fixed income have low volatility and can therefore be considered as more safe assets.

Cum. return	EW	EM	FI	$FIHY$	$Cash$	RE	HF	Inflation
Volatility	0.18	0.27	0.088	0.16	0.021	0.21	0.083	0.019
5-year	0.22	0.37	0.10	0.093	0.011	0.22	0.11	0.096
20-year	2.3	4.4	0.68	1.2	0.17	2.3	0.78	0.47
40-year	13	35	3.0	6.6	0.82	13	2.5	1.3

Table 1: Asset return volatility and cumulative returns over different time horizons, rounded to two significant digits

4 Methodology

This section discusses various effective methods for enhancing the interpretability of high-dimensional continuous RL. These methods will be investigated further for the GBWM setting in the results, Section 5. Initially, we clarify what we deem "achievable" in the domain of high-dimensional continuous RL: namely, the development of rational agents. Afterwards, we dive into interpretability and explainability techniques that fit this complex setting.

4.1 Rationalising an agent

Rationalisation of an agent can be seen as a form of intelligibility-driven regularisation, where the objective is to constrain or guide the agent's behaviour such that its decisions and actions are interpretable, understandable and relatable by humans. This involves generating behaviour that is coherent and aligns with human reasoning. The goal of this intelligibility-driven regularisation is not only to ensure that the agent generalises well but also to ensure that its actions and decisions can be understood and trusted. Hence, rationalisation plays a crucial role in determining the real-world practicality of machine learning models.

For RL agents to be used in real applications, their choices must be made intelligible; this is why we choose to define the concept of a "rational agent". We define "rationality" in the case of an RL agent as the capability of an RL agent to process and analyse states and rewards effectively and logically. Thus minimising biases and ensuring that learning and actions align with human intuition. This concept is separate from interpretable, as a black-box agent can still be rational without being interpretable. This concept is different from an agent being explainable, as an explainable agent does not necessarily need to make intuitive decisions. Thus, although explainability, interpretability, and rationality intersect, rationality focuses more on application than technicality. Rationality can be seen as a prerequisite for interpretability and explainability when one aims for practical implementation. Rationality provides a logical framework upon which decisions are made, and this framework can be scrutinised, understood, and modified if necessary. Interpreting and explaining an agent's behaviour becomes a significant challenge without a rational basis for action.

For example, if an AI model recommends a particular treatment for a patient, doctors are more likely to trust the recommendation if they can follow the logic that led to it. This is easier to achieve if the model operates on rational principles, which is quite a challenging objective to aim for.

Next, we want to highlight the distinction between rational and intuitive agents, as these concepts may initially seem interchangeable. It is important to note that intuition often relies upon processes we

may have difficulty fully explaining. Although intuitive judgements can be quick and accurate, they may not be easily analysed to uncover their reasoning. This lack of transparency makes intuition harder to interpret and explain than rationality. Conversely, rational decision-making usually involves logical steps based on specific criteria. This method is easier to examine, making it more understandable and justifiable. In domains where accountability and understandability are critical, opting for rational agents could offer advantages regarding trust, interpretability and explainability whilst sustaining much needed performance.

Note why this definition is particularly useful in the context of high-dimensional continuous RL, as with the current state of research, it is perhaps the most one can hope for. As we argued in Section 2.2, most interpretability methods do not apply here, and most exact explainability methods prove infeasible. We finish with a metaphor of a chef agent to finalise this concept further.

1. Interpretable chef: The chef works in a kitchen made of glass, including all pots, pans and utensils. An outsider can observe each ingredient, every flame and stir. One can follow the chef's techniques from start to end.

2. Explainable chef: This chef works in a rather traditional kitchen, out of view. After preparing a dish, they come out and explain to you a guide of what they did: which ingredients and what techniques. The chef thus offers insights into their decision-making process, but you cannot follow him in real-time.

3. Rational chef: This chef works behind closed doors and does not always provide a detailed explanation. However, after tasting his dish, it makes sense to your palate. You can distinctively taste the ingredients and recreate them. The chef's choices complied with your culinary intuition, and were you a skilled chef; you would probably have made them too.

Hence, we recommend that users who apply complex RL models primarily seek rational agents. This is particularly important since interpretable and fully explainable agents are only available in highly simplified settings, thus incompatible with high-dimensional continuous RL. It can be difficult to confirm (or impossible, as per the Falsification Principle (Popper, 1977)) rationality, so using these agents as advisers rather than directly applying their actions seems more feasible.

4.2 GBWM environment

For exemplary purposes, from now on, we focus on a specific environment within high-dimensional continuous RL for our results in this paper. However, our methodological results generalise to the broader field of high-dimensional continuous RL. Our GBWM environment consists of a state definition and reward function. Our state vector aims to contain all the necessary information for the agent. We include the previous asset returns, inflation, portfolio weights, wealth and time left in our state. This makes our state 17 dimensional, as we have seven assets and weights. The wealth and time left are normalised with the target wealth and time horizon. There are numerous ways to define such a state; as this research focuses more on interpretability than performance, we borrow the state definition from (Kadhum, 2022). We define the wealth goal our agent strives to achieve to be \$400,000 (inflation-adjusted) whilst starting with \$100,000 and having a 40-year time horizon, consistent with our scenario

data. Our reward function is chosen to be simple and concise;

$$r(s,a) = \mathbb{1}_{(\text{Wealth} \geq 400{,}000 \text{ at year } 40)}\ , \tag{1}$$

which is exactly one if we attain the wealth goal at the terminating state. Observe that, given the expectation of an indicator function corresponds to the probability of the condition, our aim is to maximise the probability of achieving the wealth goal across all scenarios. One may "shape" the reward (Ng et al., 1999) to engineer the agent towards the desired behaviour, which we discuss in Appendix A.1. This complicates training and interpretation, and we proceed with the binary reward function of Equation 1. One notices that this specific GBWM environment does indeed constitute a high-dimensional continuous action and state space environment: The state and action space are 17 and 7 dimensional, respectively. Further, both the state and action space exhibit continuous variables, with asset weights, returns, wealth, and inflation being continuous.

4.3 Tuning and evaluating

When we refer to a trained agent in this paper, the underlying assumption is that this agent's hyperparameters are tuned. We implement TD3, PPO and DDQN to train our agents; these algorithms each have hyperparameters, such as the learning rates, neural network parameters, exploration parameters and batch sizes. Various methods exist for hyperparameter tuning, including exhaustive grid search across a high-dimensional parameter space and applying k-fold cross-validation (Kohavi, 1995). As training agents in high-dimensional RL is computationally costly, we opt for a random hyperparameter search and a 70/15/15 train/validation/test split. The authors of Bergstra & Bengio (2012) find that compared to grid search (trying combinations over a defined grid), random search finds good or better solutions in a fraction of the time. Using k-fold cross-validation would be a more robust way of training and comparing our agents, but it requires multiple times the computing power. Besides this, there would be little benefit, as all scenarios stem from the same underlying data-generating process (the DSG). After training agents for random combinations of our hyperparameters, we select the one with the highest test accuracy based on the percentage of wealth goal achieved over the test scenarios. A downside of this method is that multiple agents may have similar achieved wealth goal percentages, while their strategies can be completely different. RL is known to be sensitive to hyperparameters as well as path-dependent. Therefore, it is difficult to conclude if differences in performance are significant. Our paper mainly focuses on interpretation; the improvements in this aspect should be displayed regardless of this inherent randomness.

4.4 Dynamic and Static agent

In our study on the GBWM problem, we consider two types of agents. The first type adopts a simplified approach: it is completely restricted to a single action, which is both time- and scenario-independent. Despite the static agent's lack of flexibility, it serves as a valuable benchmark to evaluate the performance and interpretability of RL agents. Furthermore, in practical terms, boards typically revise long-term strategic asset allocations on a yearly basis, regardless of the strategy's dynamic nature. Thus, a static one-step strategy simplifies compliance with time and regulatory constraints for boards. The concept of a static agent is inspired by Ghraieb et al. (2021), where the authors develop a degenerate version of PPO

for training agents that can only choose one state-independent action. In this context, most RL concepts, such as the discount factor, lose significance, and all rewards are aggregated. For example, a scenario that spans multiple years is reduced to a single observation, given that the action remains constant throughout this extended period, thus, we require more training scenarios. In terms of interpretability, a constant action is easy to interpret because it provides consistent and straightforward decision-making. When optimising complex non-convex temporal problems, one could argue that the static agent offers the best interpretability. However, prioritising interpretability at the cost of significant performance might be an overly simplistic approach.

The counterpart to the static agent will be referred to as the dynamic agent, which is free to alter its action in response to its current state. This adaptability comes with a cost in terms of interpretability. Numerous factors, such as prior actions or the current state or reward, can influence the dynamic agent's decision-making process. This complexity makes it challenging for stakeholders to understand the agent's decisions, which could be problematic in regulated industries or any situation requiring high levels of accountability and transparency. Thus, practically, this interpretability burden hinders adoption (Glanois et al., 2022).

The complexity of our dynamic agent makes interpreting it challenging, and the simplicity of our static agent lacks performance. Inspired by this, we investigate the "golden mean" of these two concepts, which we introduce in the next section. This golden mean involves leveraging the static agent's interpretability and the dynamic agent's performance, capturing the best of both worlds.

4.5 Interpretability

The following subsection delves into how interpretability can be incorporated into the architecture of high-dimensional continuous RL models. We introduce several frameworks, first discussing the core concept of Lazy-MDPs, which we then extend to fit in the continuous domain, arriving at the meta-controller framework. Finally, we introduce the regime change agent, a further revision adapting this to dynamic environments where temporal consistency is required. These frameworks rely on the hierarchical approach and intelligibly-driven regularisation of the actions, as discussed in Section 2.2.

4.5.1 Lazy-MDPs

RL focuses on determining **how** an agent should act optimally. In the Lazy-MDP framework, we also focus on **when** to act. Humans naturally possess the ability to combine instinctive decisions with careful planning when necessary; this framework attempts to achieve this for RL agents. It is much easier to interpret an agent if their actions are infrequent but essential. Moreover, we can interpret if an agent's actions are valuable enough to justify the overhead of not acting (Jacq et al., 2022).

4.5.2 Framework

The Lazy-MDP framework allows an agent to defer decision-making to a pre-specified default policy. Besides this, we penalise the agent for not taking this default policy. Let us briefly discuss the theoretical properties of Lazy-MDPs as proven in Jacq et al. (2022). For illustration, we will simplify some notations; we assume a constant default action as a policy and refer to the paper for the full notation. We extend the normal MDP $\mathcal{M}$'s action space A with lazy action $\bar{a}$, and introduce a default policy $\bar{\pi}$ and

penalty η. The new reward function becomes

$$r_+(s,a) = \begin{cases} r(s,a) - \eta \ , & a \neq \bar{a} \\ r(s,\bar{a}) \ , & a = \bar{a} \quad \text{(default policy action)} \end{cases} . \tag{2}$$

With state s and action a. Thus, we get an η penalty for choosing the default action (an action from the default policy). With this formalism, the authors prove optimality for the Lazy-MDP. A useful metric the authors derive is the "lazy gap". For convenience, we remove the π subscript from V and Q to prevent visual clutter. The lazy gap is defined as

$$G_Q(s) = \max_{a \in A} Q(s,a) - \mathrm{E}_{\bar{\pi}}[Q(s,a)] \ . \tag{3}$$

Equation 3 displays the gap between the value of the best action in $\mathcal{A}$ and the expected action value when the immediate next action is from the default policy $\bar{\pi}$. An optimal agent would choose the lazy action if the lazy gap of the non-lazy policy is less than η. For the extended policy, the agent would take the lazy action if the lazy gap is less than zero, as that suggests at least one action maximises expected returns (including η) better than the lazy action. The lazy gap is thus a measure of state importance, useful for interpreting the current weaknesses of the default policy.

In a discrete action space, finding the action with the maximum Q-value, $\max_{a \in A} Q(s,a)$, is relatively easy: One calculates the Q-value for each action and takes the maximum. However, finding the action that maximises the Q-value in a continuous action space is a non-trivial problem. The maximisation becomes an optimisation problem which can involve complex dynamics and constraints. This optimisation problem can be computationally intensive or even unsolvable for high-dimensional action spaces.

Furthermore, the expected value over all actions, denoted as $\mathrm{E}_{\bar{\pi}}[Q(s,a)]$, can likewise be difficult to calculate in continuous spaces. In a discrete action space, we can compute a weighted sum. But in a continuous action space, this becomes an integral over the action space, which can be a complex and computationally expensive operation, especially if the policies or the Q-function are complex non-linear functions, as is often the case when using deep neural networks. Therefore, a form of function approximation for these metrics is required.

Next, we proceed with extending the lazy framework to continuous state and action spaces, such that it is applicable within high-dimensional continuous RL. The authors of Jacq et al. (2022) focus on discrete action spaces for simplicity. However, the GBWM problem contains (high-dimensional) continuous asset weights (actions). A discrete implementation is relatively straightforward, as we can add the default action to the action space. No "one action" concept exists in continuous action space, only distributions of actions over a continuous range. To tackle this, we create a "meta-controller" framework, as visualised in graph format in Figure 4. A DDQN agent chooses between a default static action and a dynamic action. A TD3 agent chooses the dynamic action. These agents are learned simultaneously, which is not unusual in hierarchical/multi-agent learning (Panait & Luke, 2005). By varying the lazy-penalty η, we influence the degree to which the meta-controller chooses the default action. A sufficiently high η would imply that the meta-controller only chooses the lazy action. Conversely, $\eta = 0$ grants no penalty in choosing the dynamic action. Since the default action is a subset of the possible actions of the dynamic agent, there would be no reason to select the default action. During training, the dynamic

agent learns to adapt to the meta-controller and vice versa. The dynamic agent's task is thus to "deviate optimally from the default action". The meta-controller's Q-values correspond to the default or dynamic action value and indicate the expected discounted reward, considering the η penalty.

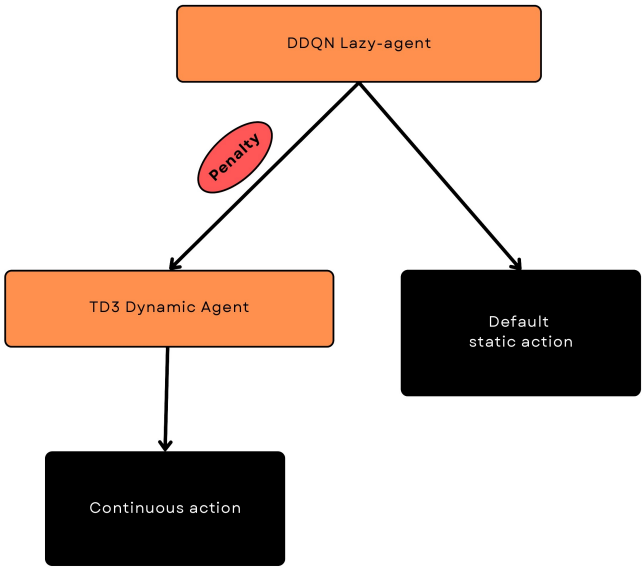


Figure 4: Diagram as visualisation for the meta-controller framework.

The difference in Q-values of the meta-controller will be our version of the "lazy gap", which we coin the "lazy preference". Thus, the meta-controller acts as a function approximator for the lazy-gap terms, simplifying continuous lazy gap calculation. Let us formalise this proposition:

$$\begin{aligned} P(s) &\equiv Q^m(s, a_{dyn}) - Q^m(s, \bar{a}) \\ &\approx \max_a Q^{dyn}(s, a) - Q^m(s, \bar{a}) \qquad \text{(meta-controller approximates optimal } Q^{dyn}\text{)} \\ &\approx V^{dyn}(s) - Q^m(s, \bar{a}) \qquad \text{(as } Q(s,a) \text{ converges to } V(s) \text{ at optimality)} \\ &= -A^{dyn,m}(s, \bar{a}) \ . \end{aligned} \tag{4}$$

Where m refers to the meta-controller and dyn to the dynamic agent, here, $A(s, a)$ is an advantage (Sutton & Barto, 2018) calculated using different agents for V and Q. a_{dyn} is the action chosen by the dynamic agent. This is based on the intuition that the meta-controller approximates the state-value for the dynamic agent with $Q^m(s, a_{dyn})$. This intuition is based on the idea that the meta-controller is effectively learning to predict how well the dynamic agent will perform given different states and uses this information to guide the agent's decisions. Of course, as both the meta-controller and the dynamic agent have the same reward function, this intuition is forced in the framework. Under optimal $Q(s, a)$ and $V(s)$ functions, both agents should have converged to the same optimal functions since they learn from the same reward signals. In this setting, and considering the extended policy, the notation simplifies to:

$$\begin{aligned} &P(s) \equiv Q(s, a_{dyn}) - Q(s, \bar{a}) = \max_a Q(s, a) - Q(s, \bar{a}) \\ &\implies P(s) = V(s) - Q(s, \bar{a}) = -A(s, \bar{a}) \ . \end{aligned} \tag{5}$$

Equation 5 is thus the "disadvantage" (negative advantage) of choosing the default action when the optimal action differs from this action. Next, we dive into the implementation details of this architecture.

The RL architecture of the meta-controller framework from Figure 4 consists of a DDQN agent and a TD3 agent. The DDQN agent (meta-controller) receives the state as input, outputs state-action Q-values for the static action and defers the action to the dynamic agent. Since the action space for the meta-controller is only two-dimensional, a simple network with ample exploration should suffice. For exploration purposes, we flip an annealing percentage of the actions so that a default action becomes a dynamic one and vice-versa. This type of exploration is common with discrete action spaces, such as with DQN, and is referred to as "epsilon-greedy exploration" (Sutton & Barto, 2018). Epsilon is the probability of exploring, thus in our case flipping the action, where a default action becomes a dynamic action and vice-versa. Without epsilon-greedy exploration, the agent would get stuck in a local minimum where it consistently chooses the default action since the dynamic agent initially performs poorly.

The dynamic agent implements TD3 with annealing Dirichlet noise and constant target policy smoothing. Thus for exploration, we generate Dirichlet noise. The Dirichlet distribution is defined over a multi-dimensional simplex, thus, it models vectors which sum to one. As discussed in Section 4.5.2, η is pre-determined and subtracted from the state reward if the action is not the default action. In the meta-controller implementation, no alignment of agents is used, e.g. we do not use the output of the dynamic agent as input to the meta-controller.

The DDQN meta-controller uses the Temporal-Difference (TD) error as a loss function (van Hasselt et al. (2015)), a discrepancy measure between the predicted and actual reward, to optimise its policy. The TD error is given by

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \; . \tag{6}$$

The controller's primary objective is to minimise this TD error. By doing so, it refines its ability to predict future rewards more accurately, thereby improving the efficacy of the policy it adopts. Note that r_t is dependent on choosing the lazy action or not. Thus, the controller estimates the value of selecting the lazy action and the value of diverting the right of action to the agent.

4.5.3 Interpretation

The Lazy-MDP framework provides interesting properties in terms of interpretability. States where the policy deviates from the default policy offer valuable information. By analysing the differences between the default and lazy policies, we can understand the key states where taking control over the default policy is essential for performance improvement. This could be seen as state importance, but the variety of options between the default policy and η grants various interpretations. On the one hand, the trained agent can be interpreted as an "advisor" for a default action given by the human. The preference indicator then serves as "advice" to the proposed action through a simple scalar metric which can be further elaborated on with post-hoc explainability techniques. The scale of η determines the severity of the circumstance in which the agent acts. On the other hand, by using the lazy gap as a "barometer" for the performance of the investor's strategy, the investor can gain insight into how he can drastically improve his performance by changing his strategy in just a few states. Thus highlighting the weaknesses of the investors' proposed strategy. Further, the low dimensional output of the meta-controller makes post-hoc

explanations easier. This low-dimensional output, represented by the default action value and the value of dynamic deviation, carries significant meaning. When interpreting and explaining this agent, one now has the ability to efficiently analyse the (proposed) actions and different default strategies through two (the outputs of the meta-controller) or even one (the preference) output.

4.5.4 Regime change agent

Since the lazy agent is penalised for performing actions, its chosen actions will differ significantly from its default actions. Furthermore, if the agent chooses to maintain its allocation, it will incur a penalty of η at each time step. This erratic behaviour serves to compensate for the η penalty. The following section aims to address this challenge using a more sophisticated architecture.

Our proposed regime change agent extends the Lazy-MDP model from Jacq et al. (2022), answering the question "**When** and **how** to switch strategy"? We proceed to discuss its architecture and thereafter the interpretability insights, highlighting its improved decision-making capabilities and strategic consistency insights. The architecture again consists of a high-level Q-network and a TD3 agent. However, the architecture is different and more complex from the Lazy-MDP meta-controller architecture in two ways:

1. To improve aligned decision makers in this more complex setting, the high-level Q-network receives both the state and the action as input and outputs state-action Q-values for the regime change. This extends the role of the meta-controller in the Lazy-MDP agent, where it only received the state as an input. The meta-controller now has access to the default action (in the state) and the proposed deviation from this (through the dynamic action).

2. The default action (now time-dependent), $\bar{a}_t$, is now exactly the previous action in this setup, implying that the agent has an embedded continuity in its policy. This addition to the architecture forces the agent to consider the cost of switching between different actions; each different action introduces a new "regime" e.g. a further default action from which to consider departing. Again, η is pre-determined and subtracted from the state reward if the action is not the default. The term "regime change" thus denotes the change of default action after switching, as the agent is incentivised to stick to this strategy for a while; we name this a "regime".

These adaptations are displayed in Figure 5; we notice that the state first enters the dynamic agent as input, where afterwards, both its output and the state itself collectively enter the meta-controller. Moreover, the continuous action becomes the new default action when it is chosen by the controller.

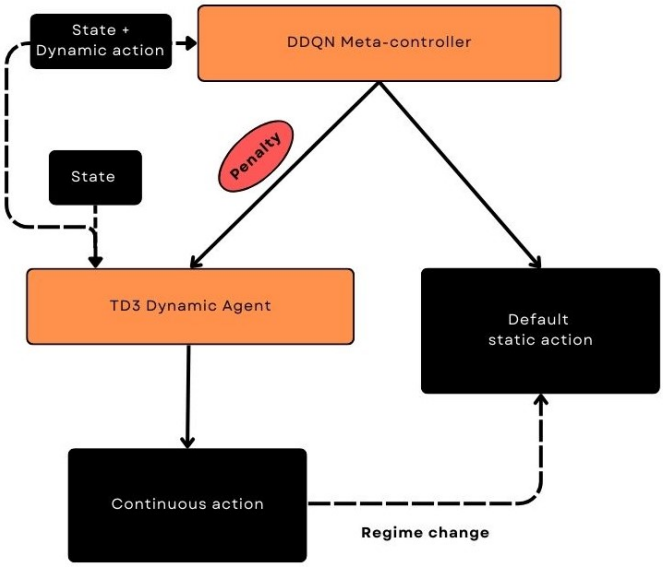


Figure 5: Diagram as visualisation for the regime change agent's framework.

An implication of this extension is that the imposed cost of switching between actions makes the agent more sensitive to change, promoting more strategic and efficient decision-making. This sensitivity enables the agent to choose long-term strategies tailored to the current state. When the agent selects a strategy, it is likely based on long-term trends rather than short-term dynamics. The agent is incentivised to maintain an action if it continues to be worthwhile rather than switching strategies frequently, which could be costly.

The lazy preference in the regime change agent works similarly to the Lazy-MDP model, with the main difference being that it is calculated considering the time-dependent default action. Therefore, it reflects the "disadvantage" of the default action over the proposed optimal dynamic action and includes a regime change's potential cost and long-term benefit. In this way, the regime change agent tries to balance regime fidelity and optimal deviation, and lazy preference is a critical measure of this balance. A high η discourages the agent from deviating from the current regime, while a low η allows for more frequent regime changes. By adjusting η, the agent's behaviour can be adjusted to best fit the environment in which it operates.

The regime change agent provides a more nuanced interpretation of its actions than the Lazy-MDP agent. The time-dependent default action grants insight into how the agent transitions from one strategy to another, uncovering patterns of strategic consistency. The proposed actions are meant to serve as new static default actions; thus, it can serve as a more advanced advisor. Its preference indicates how the proposed strategy performs if it is held for the medium to long term, guided by the η penalty. Although the complexity of the architecture has increased, the low-dimensional and intelligible output of the controller still provides the interpretability and explainability merits as discussed before.

4.6 Explainability

This section addresses the challenges of explaining agent behaviour in high-dimensional continuous RL environments. Building upon the regime change framework, we show how this yields measures of state and action importance. Further, we discuss the limitation of SHapley Additive exPlanations (SHAP) values in high-dimensional continuous RL and the challenges in visualising these.

4.6.1 Importance

To visually interpret the agent's choices, we suggest combining our preference metric with SHAP values to interpret the agent's choices visually. We use the preference to select a subset of critical scenarios or actions to explain. First let us recall: a negative preference under the extended policy implies that the agent changes its regime and occurs an η reward penalty. Important actions are therefore highlighted by their negative preference; thus, the magnitude of the preference serves as an action importance indicator. One can extend this intuition to formulate a simple scenario importance measure:

$$I(s) = \sum_{t=0}^{T} \frac{\mathbb{1}_{\{P(s_t)\leq 0\}}}{T} = \frac{1 - Lazy_{\%}}{T} \,. \tag{7}$$

Thus counting the number of negative preferences in the scenario equals the number of non-lazy actions, which is precisely equal to the number of times we choose a non-lazy action.

4.6.2 SHAP in high-dimensional continuous RL

A common practice in explaining machine learning models is to use SHAP values (Lundberg & Lee, 2017). These values determine the directional influence of individual features on predictions. Unlike purely feature importance methods, which only attempt to estimate the idiosyncratic effect of a feature, SHAP values take into account the cross-marginal impact of features. Where marginal refers to the effect of removing particular features from the feature set or subsets of the feature set, in this manner, SHAP values grant a better understanding of features and their dependent contributions to the model output. Globally optimal SHAP values are computed as follows (Lundberg & Lee, 2017):

$$\text{SHAP}(X_i) = \sum_{\forall j} \phi_j \cdot (v_j - v_{j\setminus i}) \,. \tag{8}$$

Here, X_i is a feature with j a subset of the complete feature set. The model output for a group of features is determined by $v(.)$, while $\phi(.)$ is a weight function that depends on the cardinality of feature X_i. It is important to note that SHAP values use the power set of features, which leads to an exponential increase in computational workload. When dealing with small data sets, it becomes impractical when there are approximately ten or more features.

Let us outline three key problems when using SHAP values in high-dimensional continuous RL:

1. Interpretation with many features: Calculating SHAP values can be computationally intensive when dealing with a high-dimensional state space. This is because the complexity increases exponentially with the number of features. Additionally, traditional visualisations become ineffective due to visual clutter by the numerous features for which we have calculated SHAP values.

2. Complexity with multiple outputs: In this setting, function approximators map multiple outputs to a single state. Every action dimension needs to be explained for the action (which constitutes all the outputs) to be explained. Therefore, each explainability figure, table or metric must be produced for each action space dimension.

3. Feature dependency: By their temporal nature, RL environments commonly incorporate several dependent/correlated features. Traditional SHAP methods, which assume feature independence,

may produce misleading or inaccurate attributions when applied directly, as feature impact is misattributed across the dependent features.

A popular solution for resolving problems related to problem one is to utilise approximation methods. Among these methods, Kernel SHAP (Lundberg & Lee, 2017) is often used, this method uses weighted linear regressions to estimate feature influence. Two of the critical assumptions in Kernel SHAP are: a) Linearity of explanations, thus that a change in model output is proportional to a change in a feature, and b) Independence of features, Kernel SHAP implicitly assumes this when filling in missing features in the perturbation process. Most RL problems have a sequential nature, implying that the state variables (most commonly the state) can be heavily (auto)correlated. For instance, it is clear that time and wealth are interdependent.

To tackle the independence assumption, the authors of Aas et al. (2020) propose an "empirical conditional approach" to handle feature dependence in Kernel SHAP. This approach, which we refer to as extended SHAP, involves the following steps:

1. Distance Computation: Calculate the distance between the instance to be explained and all training samples using a scaled version of the Mahalanobis distance (Mahalanobis, 1927). The Mahalanobis distance measures the distance between a point and a distribution.

2. Weight Calculation: Compute weights for all training samples based on these distances, similar to a Gaussian kernel. A bandwidth parameter, σ, controls the weight distribution.

3. Weight Sorting: Sort the weights in increasing order, and assign x_k to the training samples corresponding to the kth largest weight.

4. Integral Approximation: Approximate the integral in the SHAP value computation with a weighted version of the approximation used in the original Kernel SHAP method. This approach is more sample efficient, using each training observation only once and incorporating their weights in the integral computation.

The number of samples, K, used in the approximation is chosen such that most of the total sum of weights is accounted for by the $\mathcal{K}$ largest weights. The bandwidth parameter, σ, is selected using a small-sample-size corrected version of the Akaike information criterion to balance bias and variance. This empirical conditional approach allows for the generation of samples from the conditional distribution $p(x_S|x_S = x_S^*)$ that considers feature dependence, leading to more accurate SHAP value approximations and, thus more precise individual prediction explanations. However, this method is computationally more expensive than regular Kernel SHAP. The authors note that for > 13 features, the computation becomes intractable. Note: In our basic GBWM setting, we have 17 features (state variables). Thus while dealing with one shortcoming, we stumble upon another, which can be mitigated by increasing the computational power.

4.6.3 Visualising high-dimensional continuous RL explanations

After stating the challenges with computing SHAP values for high-dimensional continuous RL agents, we proceed with visualising these values. The high dimensionality of the state and action space causes

networks in this environment to have high-dimensional input and output layers. Further, the explainability burden becomes too large as actions can be frequent. To explain a specific action for an actor with n inputs and m outputs, one would need to inspect the $\frac{n(n+1)}{2}$ pairwise feature dependencies, and for each action, the SHAP values m times. Forcing structure and simplicity in the architecture can cause these numbers to decrease. For example, the controller in the regime change framework has two outputs, one for the dynamic regime change and one for the previous default action. Further focusing on the preference (the difference between these two outputs), we reach just one output.

We can display feature dependencies with partial dependence plots (Hastie et al., 2009). This can be done for either Kernel or Extended SHAP values, where with dependent features, the scale of the values may be distorted for kernel SHAP. In a partial dependence plot, we display one feature vs. its SHAP values, coloured by a gradient of another feature's values. This shows the dependent/marginal effects the model (neural network) captured over all the actions in one plot.

To explain the preference, one can utilise multi-output decision plots. In this figure, we leverage the fact that the sum of the directional feature impacts is precisely the model's prediction. We therefore display a line where the feature SHAP values are cumulatively summed, which displays the marginal feature impacts on the preference output. Since the regime change agent has infrequent but essential actions, and the actions are summarised by preference, we have fewer actions to visualise and can now efficiently visualise these in one figure.

5 Results

This section investigates the empirical results of applying our proposed RL methodology to the GBWM problem. We begin by analysing our dynamic/static agents, followed by our interpretability techniques regarding lazy and regime change agents. Finally, we perform an empirical exercise using SHAP values to explain high-dimensional continuous RL agents.

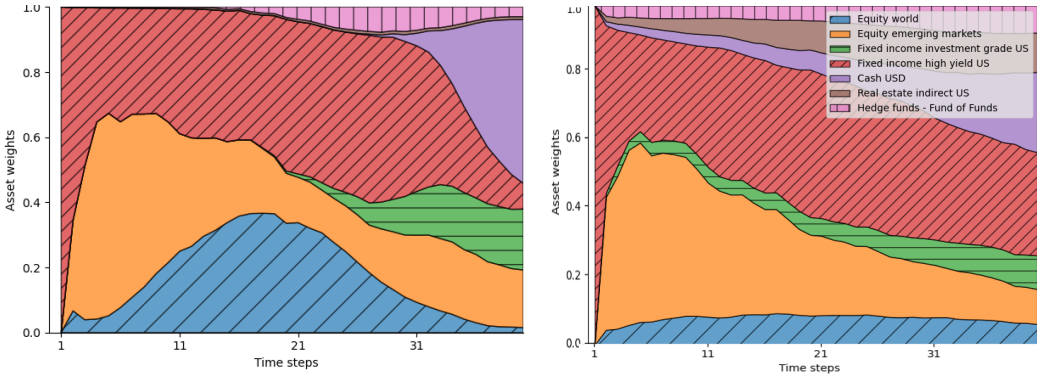
5.1 Benchmark agents

Setting the stage, we begin by analysing our benchmark GBWM agents to understand the behaviour of our GBWM agents. These agents will be used as a baseline, relative to which we measure the efficacy of our interpretability techniques for the high-dimensional continuous RL setting. With PPO, we train a dynamic and static agent. Further, we train a dynamic agent with TD3. The resulting hyperparameters for all benchmark agents are in the appendix under A.2. For evaluation, we compute several metrics over 2250 Out-Of-Sample (OOS) scenarios, displayed in Table 2. This table shows that the dynamic agents outperform the static agent, which is trivial, as the static agent is a heavily restricted version of its dynamic counterpart. The following agents are trained with a binary reward goal (Section 4.2), which is one if the goal is reached at the end of the scenario and zero otherwise.

Agent	% Wealth goal achieved	Avg final wealth	Avg portfolio risk
PPO Dynamic	87	822	0.16
TD3 Dynamic	87	1013	0.17
PPO Static	60	776	0.22

Table 2: Benchmark agent scores. Metrics include % Wealth goal achieved (percentage of the wealth goal met over the scenarios), Avg final wealth (average wealth over the scenarios at the end of the simulation, in thousands), and Avg portfolio risk (average volatility of the agent's portfolio returns).

We display the average weights of the PPO and TD3 dynamic agents over the scenario duration in Figure 6. One notices that the weights follow a similar pattern, starting with an aggressive portfolio and allocating to less risky assets over time. As agents are trained with binary rewards, it is rational to allocate to cash when the wealth goal is achieved; both agents appear to do this. Further, the PPO agent seems to utilise all assets and allocates to cash earlier; we notice that the PPO agent has a lower average final wealth but the same binary goal percentage. This may be attributed to the more diverse allocation. We urge the reader to remember that RL agents can fluctuate their actions with minor hyperparameters/random-seed/training data changes. Therefore, the actions are not representative of all PPO/TD3 agents, just these tuned ones.



(a) Average TD3 weights over all OOS scenarios. (b) Average PPO weights over all OOS scenarios.

Figure 6: Comparison of average weights over all OOS scenarios.

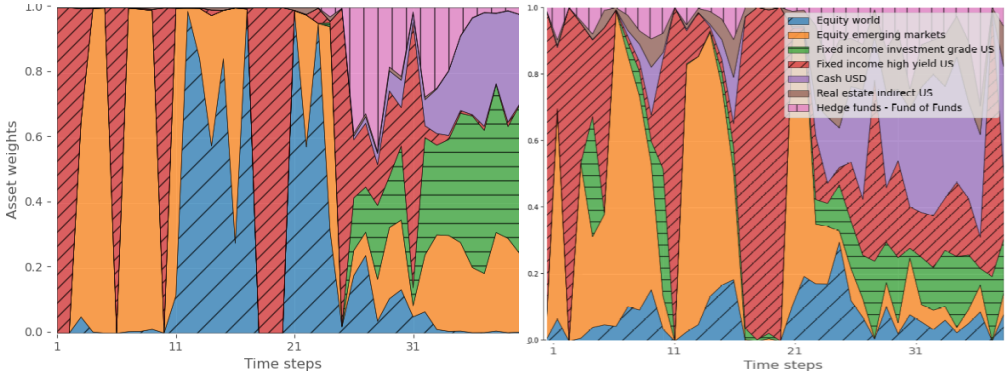
For the PPO static agent, we notice a high allocation to risky assets in its weights, which are shown in Table 3, with the most weight allocated to equity emerging markets. When optimising over long time horizons, such as 40 years in this setting, a higher allocation to riskier assets like emerging markets can be considered, as historically, the probability of negative returns decreases over prolonged periods (Markowitz, 1952).

Equity-world	Equity-EM	FIIG	FIHY	Cash	Real estate	Hedge funds
0.07	0.52	0.0	0.17	0.0	0.24	0.0

Table 3: Static allocation weights used by the benchmark agent.

Inspecting our agents further, we choose a particular OOS scenario and display the agent weights over the scenario duration in Figure 7. One quickly notices that the behaviour of these agents appears rational on average but erratic in a particular scenario. The agents frequently allocate close to 100% in one

asset. We could enforce more rational weights with portfolio constraints but prefer an inherently more rational agent whose behaviour is regularised. Although particular to the GBWM/asset-allocation setting, these findings prompt our search for a more interpretable agent in the high-dimensional continuous RL setting.



(a) TD3 weights over scenario 4.

(b) PPO weights over scenario 4.

Figure 7: Comparison of weights for OOS scenario 4.

5.2 Interpretability

Before one can interpret/explain an agent's actions, the actions must adhere to the desired behaviour. Due to biases present in RL architectures concerning the training or layers in the ANN, this exercise is not trivial. In the Appendix Figure A.1, we tackle two of these problems with regard to the GBWM setting. Now, let us examine the impact of altering the architecture of the RL model for interpretability purposes. We start by implementing the meta-controller framework and displaying the results. Afterwards, in Section 5.2.2, we move on to our regime change framework results.

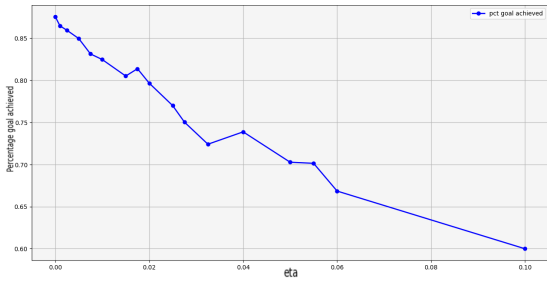
To inspect if the Lazy-MDP framework works well for high-dimensional continuous RL, we fit various lazy agents according to our meta-controller framework (Figure 4). We aim to gather valuable insights into the behaviour of lazy agents by analysing the outcomes of varying the lazy penalty hyperparameter η.

5.2.1 Lazy-MDPs

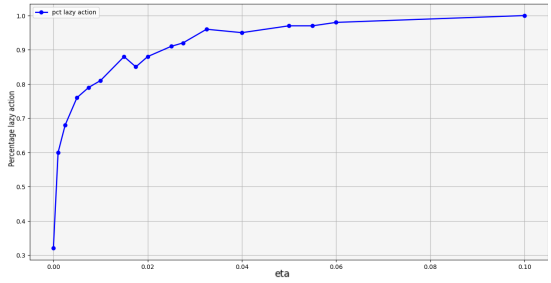
In the Lazy-MDP framework, choosing the default action can vary from being trivial to quite complex. For example, in a maze-solving game, the default action could be to walk in a random direction. In this environment, the cost of making an incorrect decision is low. In our case, the default action has to align with economic reasoning, such that the agent picks the non-lazy action when the state supports this instead of the default action performing poorly. Therefore, we select the optimal static action as the default action for the lazy agents. Hence, the lazy agent attempts to deviate from the static action optimally.

Our trained lazy agents' respective achieved goal and average lazy action percentages for various η values are displayed in Figure 8. Here the average lazy action percentage is defined as the percentage of lazy actions over all years and all scenarios. Through these plots, we can investigate the underlying relations between the importance of actions in GBWM and the η penalty. Examining Figure 8c, one notices that η and the number of lazy actions are exponentially related, resembling nth root functions.

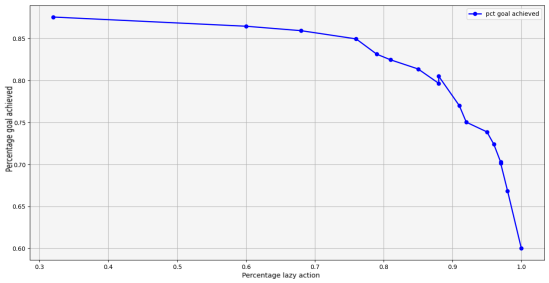
To elaborate, in Figure 8c, the line resembles the function Lazy$\%_i \approx \eta_i^{\frac{1}{n}}$, with i an index over the agents. Thus implying Lazy$\%_i^n \approx \eta_i$. Next, in Figure 8b, one notices that the percentage goal achieved and lazy actions are again exponentially related. Figure 8a shows us that η and the agent's goal appear to be linearly related, but recall that the amount of lazy actions here increases exponentially. We derive from this empirical observation that we can drastically improve performance by intervening in a small $(< 20\%)$ percentage of states. Thus, there exist certain "critical states". In other words,Figure 8b shows something similar to the "Pareto principle" where 20% of efforts yield 80% of results.



(a) % goal achieved vs η for different agents.



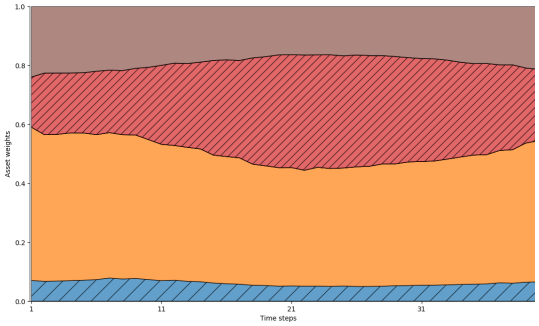
(b) Lazy % vs η for different agents.



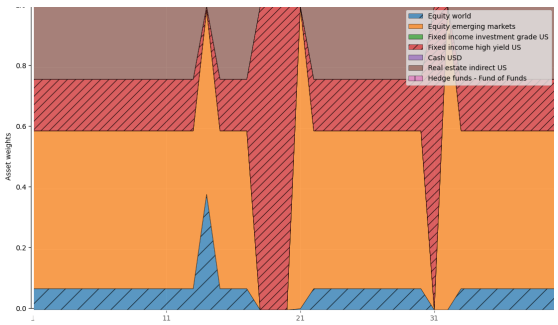
(c) % goal achieved vs % lazy action for different agents.

Figure 8: Agents trained with different η's compared with respect to their average % goal achieved and % lazy action.

Now, let us inspect if the agent's behaviour is rational by inspecting the agent's weights and preferences, which we display in Figure 9. We choose to inspect the agent with close to 80% lazy actions, this agent achieves the goal 79% of the time. Notice the erratic behaviour in Figure 9b; this was highlighted in Section 4.5.4: We suggest that since there is a penalty each time step for choosing an action, the action needs to be impactful and short; thus, it needs to be a radical short-term change. This is a problem, as the agents' rationale is hard to follow, missing practical grounding.



(a) Avg weights over all OOS scenarios.



(b) Scenario 4 weights

Figure 9: Lazy agents weights over time.

In Figure 10, we plot the preferences for scenario four as derived in Equation 5. As mentioned in Section 4.5.4, we can use this agent as an advisor. A largely positive preference indicates little reason to diverge from the current default action. In contrast, a largely negative preference implies the economic situation is alarming and changing strategy would be beneficial.

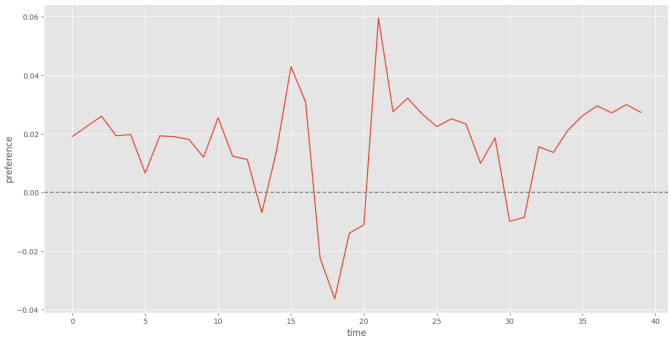
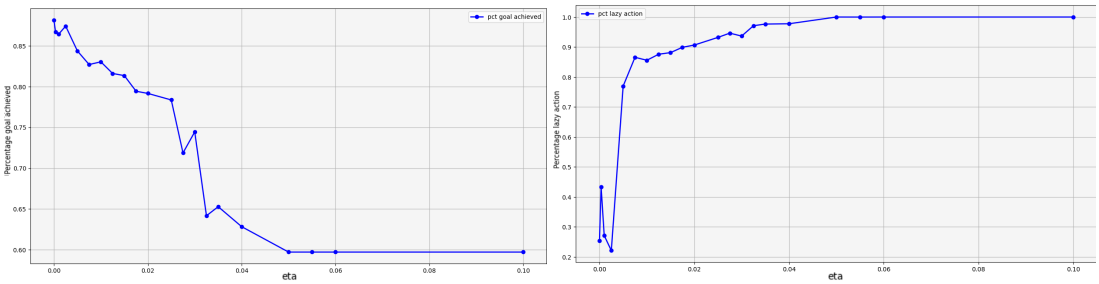


Figure 10: Lazy preferences for scenario 4.

We reserve a further investigation into the agent's behaviour using SHAP values and economic data after providing a solution to the erratic behaviour with the regime change framework.

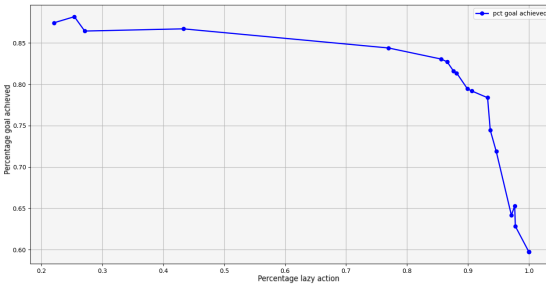
5.2.2 Regime change agent

Having established how the lazy agent behaves in the high-dimensional continuous setting, we move on to implementing our regime change framework to deal with the observed erratic behaviour. Similar to Section 5.2.1, we vary the lazy penalty η and train different agents with these. These plots are displayed in Figure 11. Recall that the non-lazy action has a different meaning here, indicating a regime change. The relations between η and our lazy/goal action/achieved percentages seem similar. In a regime change setting, it is anticipated that a higher percentage of lazy actions will result in greater rewards due to the increased impact of those actions because of their persistence over time. For example, we notice in Figure 11c that with approximately 90% lazy actions, we still achieve the goal close to 85% of the time, compared to less than 80% in the lazy setting. Thus, an even smaller percentage of actions account for almost all the results since actions have significantly more impact.



(a) % goal achieved vs η for different agents.

(b) % lazy action vs η for different agents.

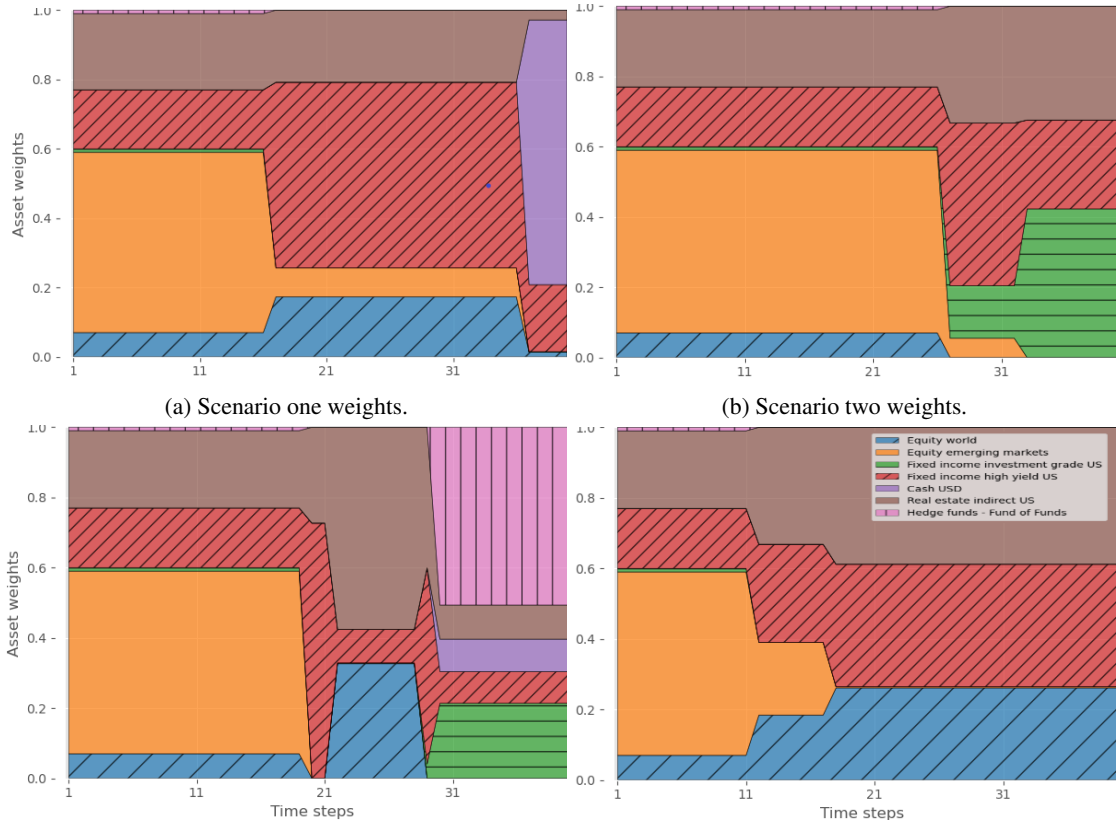


(c) % goal achieved vs % lazy action for different agents.

Figure 11: Agents trained with different η's compared with respect to their average % goal achieved and % lazy action for the regime change model.

Revisiting the observed erratic behaviour, we inspect if we solved the problem and have a more rational and interpretable agent. We examine the weights showcased in Figure 12. These weights are computed with the agent that takes the lazy action 77% of the time and achieves the wealth goal in 84% of the scenarios. Indeed, the agent's actions appear as long-term allocations, as seen in the temporal consistency of the actions. The ability to take no action allows the agent to consider the long-term effect of actions better. In most of these scenarios, the agent shifts to a lower-risk strategy, whether cash or fixed-income assets, over time.

Based on these results, we state that most dynamic agent actions do not significantly impact performance, as the agents perform comparably with a fraction of the actions. Nevertheless, taking numerous actions can make it more challenging to understand the agent's chosen strategy as it becomes more complex. The agent's rationale is thus easier to comprehend with few actions and temporal consistency.



(a) Scenario one weights.

(b) Scenario two weights.

(c) Scenario three weights.

(d) Scenario four weights.

Figure 12: Regime change individual scenario weights over time for individual scenarios one to four.

For the average weights over all scenarios, we refer the reader to the Appendix Figure 25. The regime change framework appears to work well in structuring and simplifying the architecture sufficiently to provide interpretability while maintaining adequate performance. We now move on to explaining this agent with the proposed explainability methods of Section 4.6.

5.3 Explainability

Our established regime change agent appears to demonstrate the sought-after attributes (Sections 2.2, 4.1) in terms of interpretability; next we focus on utilising these attributes to improve the explainability. We have simplified and structured the architecture such that we leverage the inherent intelligibility, offering more straightforward explanations. In this section, we dive into the explainability methods that contribute to an increased understanding of the agent's rationale in high-dimensional continuous RL. Through a backtest case study, we demonstrate how applying these SHAP methods tackles the challenges of high-dimensional continuous RL explainability. We start by evaluating the regime change agent on the backtest data, then show how we can leverage the meta-controller's preference, the infrequent but essential actions and more robust SHAP values for explanations. Finally, we explore the dependent feature relations for these actions using partial-dependence plots and finish with a detour resulting from this framework.

5.3.1 Backtest case study

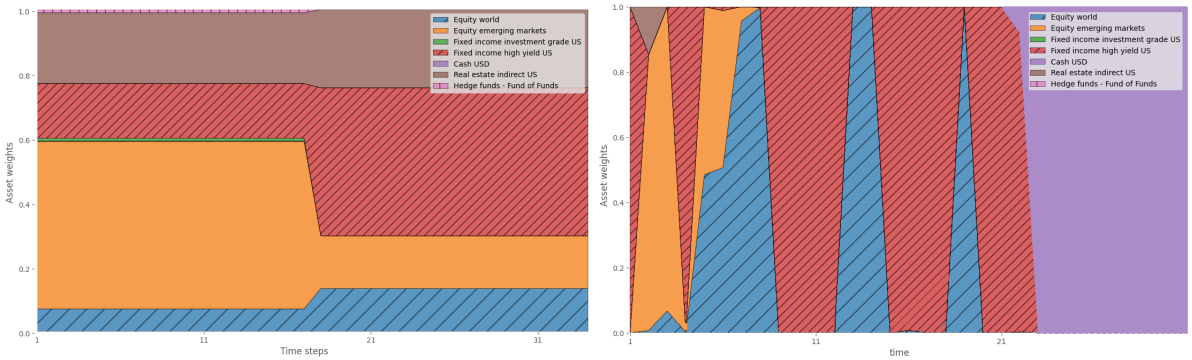
Our backtest uses historical financial data from 1991 to 2022 to evaluate agents and inspect their performance, and focus on the most promising to interpret agent; the regime change agent. Initially, we examine the weights and performance of our agents. Subsequently, we leverage our explainability methods from Section 4.6. Combining our structured architecture with (extended) SHAP values provides a novel proof of concept for a robust post-hoc explanation of high-dimensional continuous RL agents. Further, we use kernel SHAP values to provide a post-hoc explanation of the regime change agents' pairwise feature dependencies. The computation of these values is derived from the R package Shapr 0.2.2 by the authors of Aas et al. (2020), but which we adapted to Python. There are two points to consider:

1. First, the scenarios are partly generated from this evaluation data, as the DSG uses relations derived from historical data combined with expert views. Therefore, this is not a real OOS backtest, as this constitutes data leakage.

2. Second, The small amount of data combined with the meta-controller only having two outputs makes using extended SHAP values possible. With a longer horizon or slightly more features, this method becomes infeasible. Therefore, we afterwards perform a more extensive investigation with only kernel SHAP values.

We hope that this procedure provides a proof-of-concept regarding the applicability of the proposed methods in Section 4.6.2. Furthermore, we aim to investigate the problem of dependent features in RL with approximate explainability methods. Let us start by comparing a (tuned) dynamic actor vs. our regime change agent with a 77% Lazy-action percentage, which achieves the goal in 84% of OOS scenarios. Inspecting the weights over the backtest data in Figure 13, we notice that the regime change agent only changes its regime once in our historical data set. This is unlike the scenarios, as can be seen by comparing the weights with Figure 12. Possibly because the static portfolio performs well in this data set and, therefore, does not need to be altered much in the historical data set. As seen in the regime change agents' preference in Figure 14, the regime change agent changed its regime in 2009, after the financial crisis. Moreover, the regime change agent appears confident (high preference) throughout the backtest.

The dynamic agent appears aggressive, with mostly 100% concentration in equities; after reaching far beyond its wealth goal, it fully allocates to cash. The agent mostly switches its allocation between emerging markets equity and high-yield fixed income, which performs rather well. Next, we calculate portfolio risk using the average covariance matrix. The first regime has a risk of 0.185, and the second has a risk of 0.140, so the agent switches to a slightly less risky portfolio.



(a) Regime change agent, with 84% goal achieved in the OOS scenarios.

(b) Dynamic goal achieved actor, with 88% goal achieved in the OOS scenarios.

Figure 13: Regime change backtest weights over time.

Figure 14: Regime change agent backtest preference values over time.

Let us focus on the portfolio wealth of the dynamic and regime change agent, which we display in Figure 15. The wealth goal is scaled in the backtest, as the number of years in the set is different (31 instead of 40). Here, the wealth goal is approximately $300,000. Notice that the dynamic agent allocates everything to cash around $800,000, thus well beyond the wealth goal. The regime change agent steadily built up wealth, changing strategy after its largest 1-year loss in 2008.

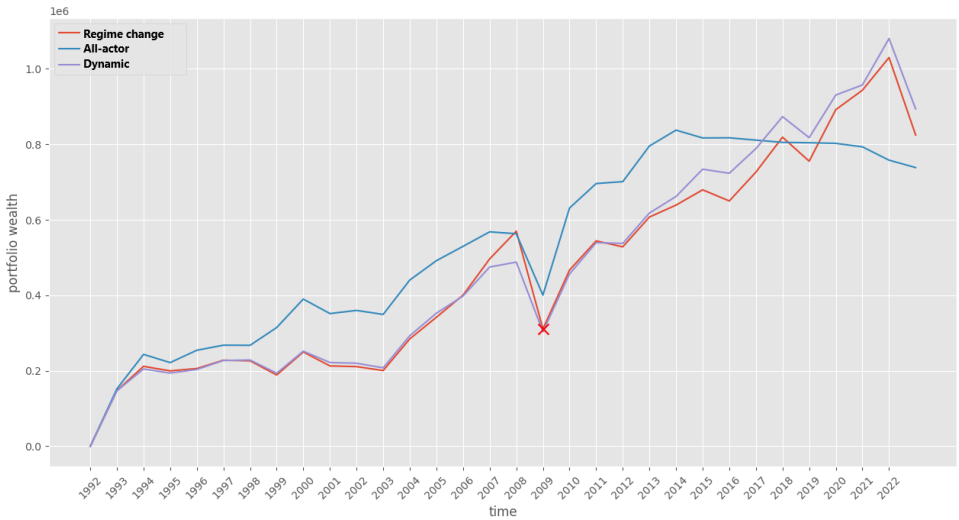
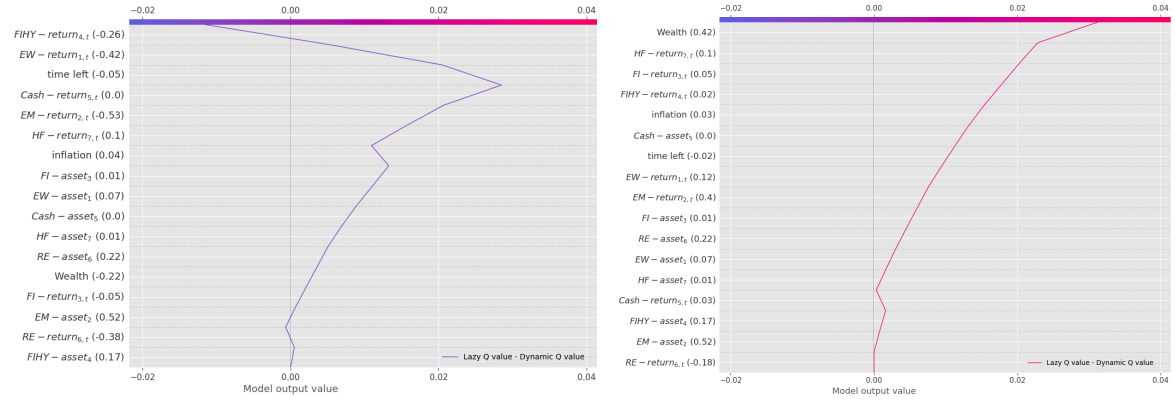


Figure 15: Backtest agents' wealth over time with a cross where the regime changes.

We will now proceed by attempting to explain the 2008 regime switch. To achieve this, we inspect

the 2009 extended SHAP multi-output decision plot with the preference as the target output in Figure 16a. Here, the agent is convinced to change its regime due to a large negative equity world and high yield fixed income return. Moreover, the large negative equity emerging market return (-53%) appears to push the agent towards not changing its regime. This is possibly attributed to the fact that the agent has a large allocation to emerging markets (53%). Thus, when the agent has a low allocation to a risky asset, a significant negative return prompts the agent to allocate towards this asset. This is in line with the concept of mean-reversion in financial instruments, where the authors of Poterba & Summers (1988) argue that these substantial moves can lead to asymmetric long-term upside.



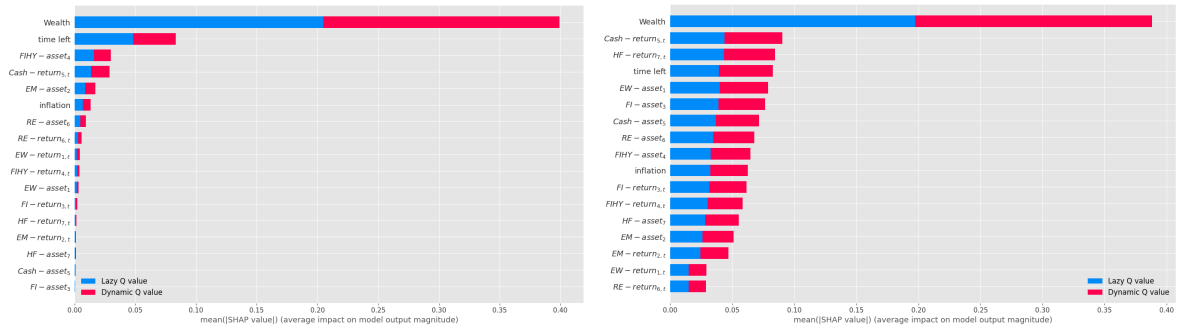
(a) Regime change agent backtest 2009 preference SHAP values multi-output decision plot.

(b) Regime change agent backtest 2008 preference SHAP values multi-output decision plot.

Figure 16: Multi-output decision plots based on the backtest SHAP values for the regime change agent preference.

Our plot moves back one year to 2008. In Figure 16b, we notice that in 2008 everything appears to be in order, and almost every feature causes the agent to stay with its regime. Only the interest rate (3%) appears to affect the preference negatively. In Figure 16a, we saw that a 0% interest rate significantly prompted the agent to stick to its regime, possibly due to the high allocation to risky assets, which benefit from low-interest rates.

The low dimensional output of our meta-controller combined with the low amount of backtest data grants us the computational ability to compute extended SHAP values. Let us return to kernel SHAP values; in Figure 17, we display the SHAP variable importance figures for extended SHAP and kernel SHAP. In kernel SHAP, the importance of one feature might be incorrectly inflated because it captures the effect of another correlated or interacting feature. Extended SHAP would correct for this, potentially resulting in a rank swap between the features. Confirming this, one notices that in Figure 17 the ranks change between SHAP varieties, and the Kernel SHAP likely misattributes significant importance to the highly correlated features. Wealth and time left are the highest correlated state variables (0.79), thus there is possibly a significant misattribution of either time left or wealth. Figure 17 shows that relative to the other features, time left loses the most importance (average absolute SHAP value) when correcting for this correlation.



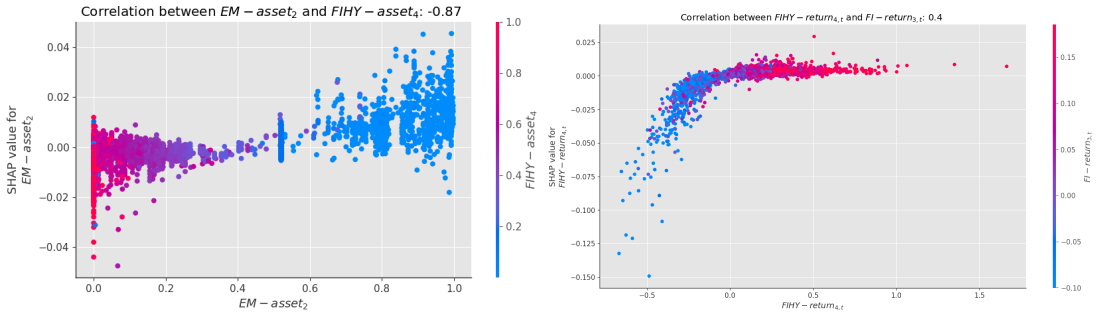
(a) Kernel SHAP feature importance plot.

(b) Extended SHAP feature importance plot.

Figure 17: Average absolute SHAP for every feature on the historical data set.

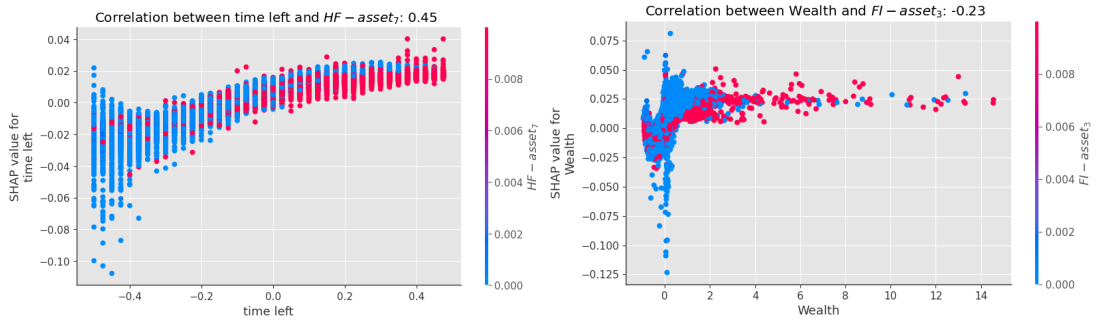
Next, we inspect the behaviour of the regime change agent over the OOS scenario set, computing kernel SHAP values over this set and displaying the partial dependence plots to retrieve a post-hoc explanation. We need many SHAP values to display the relation accurately; thus, we cannot use extended SHAP values here. The previously discussed shortcomings of kernel SHAP may cause the scale of the SHAP values to differ from the actual scale. The following segment illustrates how SHAP partial dependence plots can be employed to explain an agent's actions.

Figure 18 shows the partial dependence plots for our feature set, selected based on Kendall's tau (Kendall, 1938), computed with the actor in the regime change framework. For most feature pairs, there is no clear relation; an example of this is shown in Appendix Figure 26. Figure 18a, we notice that a low allocation to high-yield fixed income usually implies a high allocation to emerging markets equity and a low positive SHAP value for emerging markets equity. The converse also holds a high allocation to high-yield fixed income, implying a negative SHAP value and a low allocation to emerging markets equity. One intuitive explanation might be that emerging markets equity is used as a wealth builder and high-yield fixed income as a wealth conservation category. Next, in Figure 18b, a large fixed income return often implies a close to zero SHAP value for high-yield fixed income. When the fixed income and high yield fixed income returns are negative, we notice that the SHAP value for high yield fixed income returns becomes large and negative, thus negatively impacting the expected future returns. In Figure 18c, we further notice that a high allocation to hedge fund of funds is associated with higher importance for the time left and high time left. For our agent, hedge fund of funds can be a suitable long-term investment for building wealth, thus having a long time horizon. Last, by Figure 18d, we notice that a high fixed income allocation usually implies a high wealth (as it is a wealth-preserving instrument). The impact of wealth on the expected returns also increases with the fixed income allocation, following the same reasoning. When wealth is closer to zero, different factors drive the SHAP value, so fixed income only explains the tail.



(a) Emerging markets and high-yield fixed income allocations partial dependence plot.

(b) Fixed income investment grade and high yield returns partial dependence plot.



(c) Time left and hedge fund of funds allocation partial dependence plot.

(d) Wealth and fixed income allocation partial dependence plot.

Figure 18: Partial dependence plots for different feature combinations with the correlations in the testing set. All values are computed with kernel SHAP.

We notice that, in the context of high-dimensional continuous RL, producing accurate and rational explanations is challenging due to the intricacy of the state and action spaces, making most methods infeasible. For example, we displayed how the commonly used kernel SHAP values fail to capture the correct scale for RL features (which are interdependent). The case study of our proposed methods shows that if we follow two key principles; simplicity and structure, it is feasible to achieve a balance between robust explainability and performance in this setting. Expanding on these two concepts:

1. Simplicity is accomplished by constraining the action space to fewer but meaningful actions; this is done by leveraging our DDQN meta-controller agent, which is aligned with the dynamic agent. The regime change framework makes it so we explain fewer actions, making explanations more accessible and actionable.

2. We achieve structure through the lazy/regime framework. Complex architectural components gain meaning by leveraging the inherent meaning of our environment and are organised into intelligible units. Thus, the agent's decision-making is linked to explicit temporal (when to take action) and value (preference indicator) considerations, granting interpretability.

Combining these two aspects makes more accurate and rational explanations possible without sacrificing significant performance. Thus opening up high-dimensional continuous RL agent's deployment to real-world settings.

5.3.2 All actor detour

We enter a brief excursion to investigate an interesting consequence of the regime change framework. With this framework, we force a sense of long-term impact into the agent's actions. Of course, the dynamic agent also estimates the long-term value of its actions through the critic. However, since the dynamic actor can always take action, its action's value is computed with the idea that the strategy can be altered in the future if the scenario needs it. Suppose we remove the meta-controller and allow the hierarchical dynamic agent to make all decisions. In the regime change framework for a trained agent, this equates to never selecting the default action. In that case, it must consider long-term economic trends while reacting to the current state since the agent perceives that its actions will establish a new regime.

In Figure 15, we have displayed the wealth of this agent, coined the "all-actor" agent, through our backtest data. We notice a steady climb and less severe downturns than the other agents. Further inspecting the all-actor agent's weights, which we plot in Figure 19, we see that the all-actor agent starts with a portfolio close to the static portfolio and balances the portfolio towards a more risky or diversified portfolio when needed. These weights over time seem more intelligible than the dynamic actor's weights in Figure 13b.

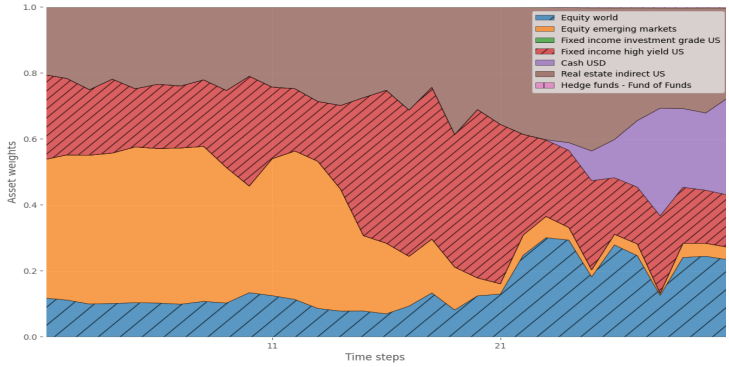


Figure 19: Regime change agent from Section 5.3.1 actor (all-actor agent) backtest weights over time.

One can interpret the all-actor agent as a temporally regularised dynamic agent since we penalise the action if it does not attribute to long-term performance. However sub-optimal this may sound when striving for the highest goal metric, in a real-world application, funds cannot be allocated erratically and must always align with long-term logic. Therefore, it can be considered a form of intelligibly-driven regularisation, as in Section 2.2.

6 Conclusion

Interpreting high-dimensional continuous RL agents forms a tough challenge, especially when a high degree of trust is required, such as in wealth management. Despite the encouraging results of RL algorithms, their adoption is hindered by our inability to explain and justify their actions adequately. Interpretable RL research appears promising, but the specific subset of high-dimensional continuous RL interpretability lacks the same depth and variety. This research has made progress in improving the interpretation capability for high-dimensional continuous RL agents by introducing various frameworks and techniques. Besides providing a methodological approach to interpreting agents in this domain, we created an overview of how RL interpretability methods relate to it.

Our main contribution, the novel regime change agent, powered by the meta-controller framework, introduces structure and simplicity into the RL architecture. This agent builds upon the Lazy-MDP framework, extending it to continuous action spaces and providing temporally consistent actions. The regime change agent can be seen as a form of intelligibly driven regularisation integrated with a hierarchical interpretability approach. We depict how this framework forms the "golden mean" of the interpretable but overly restrictive static agent, and the performant but complex dynamic agent. Our results show how this agent provides a more interpretable agent without redundant complexity. This study introduces the concept of an RL agent's rationality, which constitutes what is possible to attain within high-dimensional continuous RL in terms of interpretability.

Further, this framework makes post-hoc explainability methods viable in high-dimensional continuous RL through the preference indicator. This creates a one-dimensional output incorporating all vital information for decision-making, such that considerably fewer explanations are required per action. The problems of using post-hoc explainability methods such as SHAP in this domain are highlighted in this paper, and we tackle these problems with our inherent structured and simplistic architecture in conjunction with more robust SHAP values (Aas et al., 2020). Thus, we demonstrate that these two aspects; structure and simplicity, make more accurate and rational explanations possible without sacrificing significant performance. The computation and usage of these SHAP values, which correct for feature dependence, provide a first-of-a-kind way to use SHAP for RL robustly.

Finally, using 15,000 generated economic scenarios, we train our agents to tackle the GBWM problem. Through this data, we substantiate the improved interpretability attained by the regime change agent whilst retaining reasonable agent performance. As a practical guide to interpreting this agent, we perform a case study utilising real financial data. This case study demonstrates how integrating a structured environment with robust but costly explanations creates a synergistic effect, where one improves the other. This backtest also gives empirical substance into how kernel SHAP values, which are generally utilised, fail to capture the complete impact of interdependent RL features.

Regarding the practical implications arising from this paper, our devised interpretation framework makes implementing high-dimensional continuous RL to solve real-world problems feasible. Instead of solving complex problems in a one-step static manner, sacrificing significant performance, one can opt for the regime change framework. An example could be a boardroom devising a strategic asset allocation strategy. Instead of optimising non-convex problems in a simplified static way, we use the regime change agent's preference as an advisor to the board's proposed allocations. Moreover, if the allocation proves poor, the dynamic agent can provide an optimal long-term aligned allocation. This agent delivers the benefits of a dynamic strategy while maintaining the interpretability of a static case.

Next, there are several limitations to this research. For the practical part of our research, using generated scenario data, which is partly from long-term economic series and expert views, while allowing the training of RL agents, removes our ability for true OOS evaluation. Since all data is generated by the same model, there is a form of data leakage in the train set, causing the agent to potentially exploit the generator. Second, the computation of feature dependency robust SHAP values remains computationally intensive; this limitation forces us to use a limited number of samples, making the results less reliable. Third, the scope of this paper concerns just one RL environment, whilst it is common in RL research to compare numerous environments when making architectural changes.

For further research, the broad subject of high-dimensional continuous RL interpretability remains largely unexplored. Many RL interpretability techniques can probably be extended to this domain but are currently not due to the complexity. We recommend testing different RL environments and explainability methods with these/our techniques. Besides this, to truly test these methods, one needs to perform more advanced tuning methods, such as k-fold cross-validation, which proved too computationally intensive for this research. Last, while providing several techniques for improving high-dimensional continuous RL interpretability, there are countless and often environment-specific approaches to increasing interpretability. This paper proposes just one approach to intelligibility-driven regularisation, the hierarchical approach and post-hoc explainability, while others are possible. Thus, in the context of high-dimensional continuous RL, this work proposes one path to greater interpretability, but this serves as the start and not the end of this exploration.

References

Aas, K., Jullum, M., & Loland, A. (2020). Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *AI 2021*.

Alharin, A., Doan, T.-N., & Sartipi, M. (2020). Reinforcement learning interpretation methods: A survey. *IEEE Access*.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*.

Bellman, R. (1957). A markovian decision process. *Indiana University Mathematics Journal*.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *26th International Conference on Machine Learning*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*.

Brunel, J. L. (2011). Goal-based wealth management in practice. *The Journal of Wealth Management*.

Chen, J., Yuan, B., & Tomizuka, M. (2019). Model-free deep reinforcement learning for urban autonomous driving. *IEEE Intelligent Transportation Systems Conference*.

Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). Trust region methods. *MPS/SIAM Series on Optimization, Society for Industrial and Applied Mathematics (SIAM)*.

de Brebisson, A., & Vincent, P. (2016). An exploration of softmax alternatives belonging to the spherical loss family. *ICLR 2016*.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., & Hester, T. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *35th International Conference on Machine Learning*.

Ghraieb, H., Viquerat, J., Larcher, A., Meliga, P., & Hachem, E. (2021). Single-step deep reinforcement learning for open-loop control of laminar and turbulent flows. *PHYSICAL REVIEW FLUIDS*.

Glanois, C., Weng, P., Zimmer, M., Li, D., Yang, T., Hao, J., & Liu, W. (2022). A survey on interpretable reinforcement learning. *arXiv preprint arXiv:2112.13112*.

Greydanus, S., Koul, A., Dodge, J., & Fern, A. (2017). Visualizing and understanding atari agents. *arXiv preprint arXiv:1711.00138*.

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. *34th International Conference on Machine Learning*.

Gupta, P., Puri, N., Verma, S., Singh, S., Kayastha, D., Deshmukh, S., & Krishnamurthy, B. (2019). Explain your move: Understanding agent actions using focused feature saliency. *International Conference on Learning Representations*.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. *The Elements of Statistical Learning Data Mining, Inference, and Prediction, Second Edition*.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning*.

Jacq, A., Ferret, J., Pietquin, O., & Geist, M. (2022). Lazy-mdps: Towards interpretable reinforcement learning by learning when to act. In *Proceedings of the 21st international conference on autonomous agents and multiagent systems.*

Kadhum, A. (2022). Deep reinforcement learning for goal-based wealth management: a model-based approach.

Kanai, S., Fujiwara, Y., Yamanaka, Y., & Adachi, S. (2018). Sigsoftmax: Reanalysis of the softmax bottleneck. *31st International Conference on Neural Information Processing Systems*.

Kendall, M. (1938). A new measure of rank correlation. *Biometrika*.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection..

Kolm, P. N., & Ritter, G. (2020). Modern perspectives on reinforcement learning in finance. *The Journal of Machine Learning in Finance*.

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

Laha, A., Chemmengath, S. A., Agrawal, P., Khapra, M. M., Sankaranarayanan, K., & Ramaswamy, H. G. (2018). On controllable sparse alternatives to softmax. *Advances in Neural Information Processing Systems*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*.

Liu, G., Schulte, O., Zhu, W., & Li, Q. (2018). Toward interpretable deep reinforcement learning with linear model u-trees. *Lecture Notes in Computer Science*.

Lugano, G. (2017). Virtual assistants and self-driving cars. *2017 15th International Conference on ITS Telecommunications (ITST)*.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Conference on Neural Information Processing Systems*.

Mahalanobis, P. C. (1927). Analysis of race mixture in bengal. *Journal and Proceedings of the Asiatic Society of Bengal*.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*.

Martins, A. F. T., & Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. *arXiv preprint arXiv:1602.02068*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Ostrovski (2015). Human-level control through deep reinforcement learning. *Nature*.

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *International Conference on Machine Learning*.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*.

Popper, K. (1977). The logic of scientific discovery. *Philosophy of Science*.

Poterba, J. M., & Summers, L. H. (1988). Mean reversion in stock prices: Evidence and implications. *Journal of Financial Economics*.

Qing, Y., Liu, S., Song, J., Wang, H., & Song, M. (2022). A survey on explainable reinforcement learning: Concepts, algorithms, challenges. *arXiv preprint arXiv:2211.06665v2*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*.

Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., & Zhong, C. (2022). Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*.

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

Steehouwer, H. (2005). Macroeconomic scenarios and reality: A frequency domain approach for analyzing historical time series and generating scenarios for the future. *PhD Thesis Vrije Universiteit Amsterdam*.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*.

van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double q-learning. *30th AAAI Conference on Artificial Intelligence*.

Verma, A., Murali, V., Singh, R., Kohli, P., & Chaudhuri, S. (2018). Programmatically interpretable reinforcement learning. *35th International Conference on Machine Learning*.

Vouros, G. A. (2022). Explainable deep reinforcement learning: State of the art and challenges. *ACM Computing Surveys*.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*.

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Yang, Z., Dai, Z., Salakhutdinov, R., & Cohen, W. W. (2017). Breaking the softmax bottleneck: A high-rank rnn language model. *Research Intelligence*.

A Appendix

A.1 ANN biases

ANNs are powerful function approximators which can be successfully applied to various tasks in RL. However, ANNs are inherently complex, and their decision-making processes are often opaque and difficult for humans to understand. This opacity can sometimes result from certain biases ingrained in the architecture and training. Before we force structure and simplicity in the agent's architecture, it is vital to consider and explore these biases, as biases make it difficult to interpret an agent's rationale.

The output layer for the actor is possibly the most important component of our networks, as it dictates precisely how assets are allocated. Under the constraint of weights summing to 1 and bounded by zero and 1, there are three common options for the output layer (Laha et al., 2018):

- Softmax, which cannot be sparse.
- Sum-normalisation, which is not full-domain i.e. the domain of the function is restricted.
- Spherical softmax (de Brebisson & Vincent, 2016), which is not monotonic.

As portrayed in Yang et al. (2017) and Kanai et al. (2018), the literature has questioned the efficacy and flexibility of the softmax activation function. The softmax function is commonly used to convert a set of real numbers into a probability distribution. For a given input vector x of length n with elements $x_i, i \in 1, ..., n$, the softmax function $s(x)$ is computed as

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \,. \tag{9}$$

Softmax performs well in the context of classification; however, the authors of Guo et al. (2017) have established its difficulties in approximating probability distributions. In asset allocation, this bias leads to over-concentration in assets due to the temperature bias when the temperature is set to one. To elaborate, we will demonstrate that the softmax temperature parameter correlates highly with diversification without sacrificing returns. Thus its inability to be sparse grants unwanted non-zero weights for higher temperatures (as we consider a temperature of one an inherent assumption). Many alternatives to softmax have been proposed, for example, sparsemax (Martins & Astudillo, 2016).

The sparsemax function is an alternative to the softmax function used to compress an arbitrary real-valued vector into a probability distribution. One interesting property of sparsemax is that it can output exactly zero, making the resulting probability distributions sparse. It is calculated in the following way:

1. Sort x in decreasing order and obtain the sorted vector $x^{(sort)}$. Also, compute the cumulative sum of $x^{(sort)}$, denoted by $cumsum(x^{(sort)})$.
2. Define $k(x)$ as the maximum k such that $k + 1 > \frac{cumsum(x^{(sort)})_k - x_k^{(sort)} \times (k+1)}{k}$. The function $k(x)$ finds the number of elements with non-zero probability after applying the sparsemax transformation.
3. The sparsemax transformation of x is given by $sparsemax(z) = max(z - \tau(z), 0)$ where the threshold function $\tau(z) = \frac{cumsum(z^{(sort)})_{k(z)} - 1}{k(z)+1}$.

Contrary to softmax, even after increasing the temperature, the agent can choose exactly 0 weight for assets.

To make the effect of the temperature apparent in the results of an RL agent, we have to force the inputs to be bounded. In practice, this can be achieved using normalisation layers before the output layer, such as batch normalisation (Ioffe & Szegedy (2015)). Let us briefly show that the ANN can indeed circumvent the temperature if the input is from an unbounded layer, e.g. a linear layer.

$$\begin{aligned} \mathrm{s}_T(kx_i') &= \frac{\exp(kx_i/T)}{\sum_j \exp(kx_j/T)} \\ &= \frac{\exp(k \cdot x_i/T)}{\sum_j \exp(k \cdot x_j/T)} \\ &= \frac{\exp(x_i/(T/k))}{\sum_j \exp(x_j/(T/k))} \,. \end{aligned} \tag{10}$$

Thus, by choosing $k = T$ as the scaling factor for the inputs, the neural network can theoretically make the effects of the temperature redundant, which is not desirable as we then keep the concentration bias.

As will be displayed in our experiments with softmax, inherent architectural biases are unwanted, as they make the outcomes of our agents more unpredictable and thus unreliable. We cannot interpret agents from which there is not a sense of continuity in their action rationale.

A.1.1 Curriculum learning

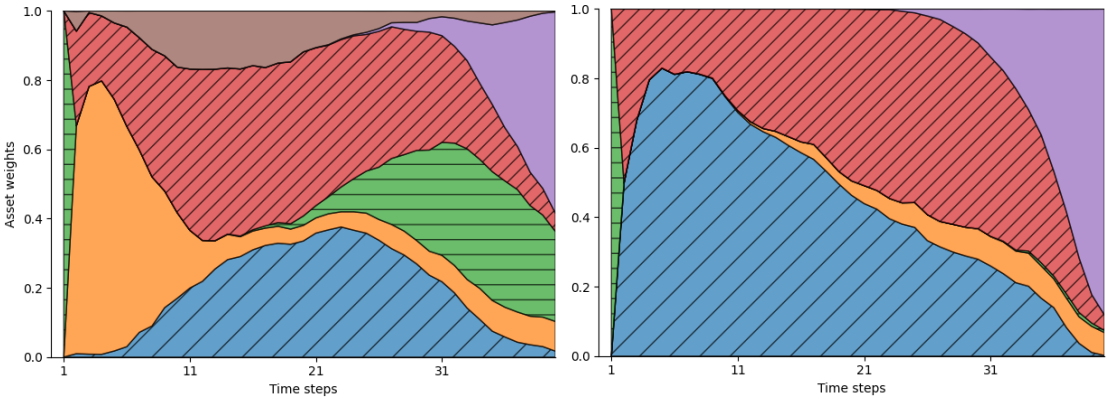
In order to get the desired behaviour of an RL agent under complex reward functions, one has to take a more structured approach when learning this reward. Since before we can interpret the agent's behaviour, it must display the correct behaviour. Therefore we introduce the concept of curriculum learning, popularised in Bengio et al. (2009) to achieve the desired behaviour, which one can then interpret. Curriculum learning is a training method where an agent or a model is gradually exposed to more complex examples as training progresses. It draws inspiration from human learning processes, where we learn complex concepts by first understanding simpler foundational concepts. Curriculum learning allows humans to observe the learning progression of an agent and offer insight into what kind of behaviour is hard to learn and what unintended behaviour dominates the agent's decisions. For example, curriculum learning can be applied to reward shaping by first learning the primary objective and then training the shaped reward. Thus regulating the behaviour of the agent such that we can subsequently interpret it.

A.1.2 Results

GBWM can contain numerous goals. In this section, we will attempt to make the agent align with these goals and investigate any biases that may hinder the desired result.

We will attempt to use the well-established methods of reward shaping and curriculum learning to create a more conservative and less erratic agent. Our reward shifts from the binary wealth goal reward to $binary\_reward - 5 * portfolio\_std$. Where $portfolio\_std$ is the standard deviation of the 40 portfolio returns at the termination year 40. The learning curves are displayed under Figure 23. In Figure 20, we notice that the average weights of the agent trained with curriculum learning display a

more diversified portfolio, as intended. The agent trained without curriculum learning, thus having the shaped reward during the training, appears more concentrated than without a shaped reward, failing to achieve the intended objective. This can be attributed to more noise while learning; there is already exploration noise, so the agent cannot distinguish the reward components from each other.



(a) Average shaped reward TD3 weights over all OOS scenarios with curriculum.

(b) Average shaped reward TD3 weights over all OOS scenarios without curriculum.

Figure 20: Comparison of average weights over all OOS scenarios with/without curriculum learning.

Again, notice that the actions are erratic, as displayed in Figure 24. This problem will be tackled in Section 5.2.2. Now that we possess the ability to guide the agent's behaviour regarding rewards, we will tackle one ANN bias present in our GBWM setting, which forms an obstacle leading to undesired results.

To illustrate the impact of temperature increase, we plot the softmax and sparsemax functions for a range of portfolio weights in Figure 21. For softmax, we notice that increasing the temperature leads to the weight distribution becoming more uniform. Moreover, increasing the temperature leads to low (0) weights becoming positive. When comparing with sparsemax, we see that increasing the temperature makes the distribution more uniform but keeps low weights precisely 0. This addresses the nonzero weight bias seen in softmax. This change could help an agent make a diverse portfolio while avoiding investments in less productive assets, like cash, which would be forced with the nonzero softmax bias.

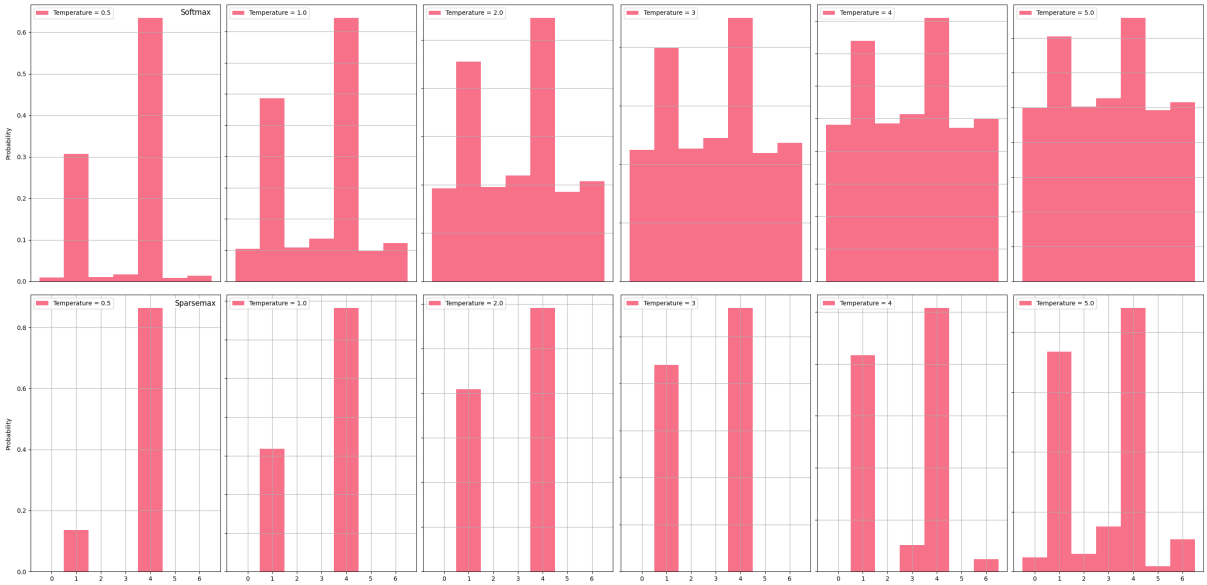
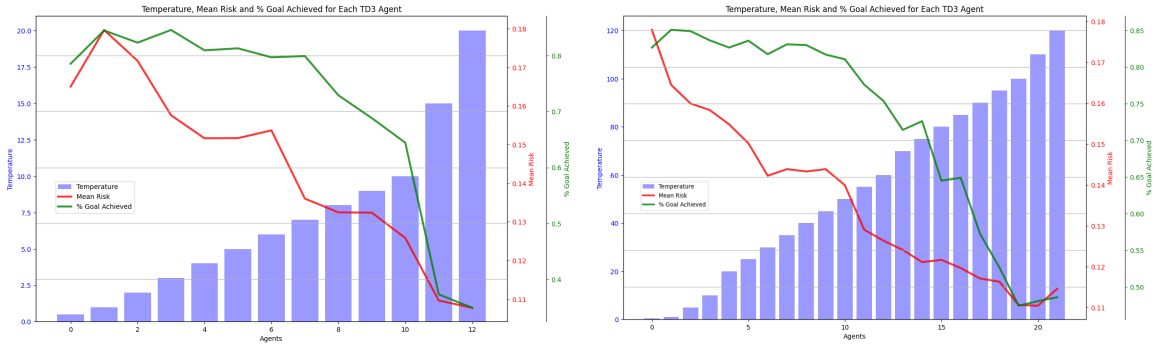


Figure 21: Softmax and sparsemax for the same set of portfolio weights with increasing temperature from $\frac{1}{2}$ to five.

To display the softmax bias in our setting, we train multiple agents with increasing temperature from $\frac{1}{2}$ to five with their achieved goal % and mean risk in the training set, shown together in Figure 22. Inspecting Figure 22a, we notice that we can increase the temperature to a certain point with increasing reward and decreasing risk. The expectation is that an optimal agent should discover the higher reward actions, making this a bias. Furthermore, the softmax bias creates an adverse risk profile for our agent. Notice a large part of the bias is gone in Figure 22b, and since the agent can still assign close to 0 weights with large temperatures, the minimum achieved goal percentage is higher (50%) than with softmax (40%). To elaborate, in the early stages of the scenario, the agent should not benefit from allocating a significant portion to cash. However, this allocation is forced when using the softmax function with a high-temperature parameter. A point of criticism is the inherent randomness associated with complex RL models. Hence marginal changes in statistics may not be significant, so it is essential to focus on observing the overall trend rather than marginal fluctuations.



(a) Influence of varying softmax temperature on percentage of goals achieved and mean risk across training scenarios.

(b) Influence of varying sparsemax temperature on percentage of goals achieved and mean risk across training scenarios.

Figure 22: Comparative analysis of Softmax and Sparsemax temperatures in relation to the percentage of goals achieved and mean risk during training.

To conclude, we have shown the presence of two ANN biases for solving GBWM with RL and how they adversely affect the results. This makes agents counter-intuitive and, hence, more unreliable and uninterpretable. We hope the reader has a more nuanced view regarding the various challenges in training a high-dimensional continuous RL agent.

A.2 Hyperparameters

Hyperparameter	Space	Tuned value
hidden_dim	{64, 128, 256}	128
batch_size	{64, 128, 256}	128
scenario_batch_size	{20, 50, 100}	50
actor_lr	{1e-05, 5e-05, 1e-04}	5e-05
critic_lr	{1e-06, 5e-06, 1e-05}	1e-05
tau	{0.01, 0.05, 0.1}	0.05
gamma	{0.9, 0.95, 0.99, 1}	0.99
uniform_episodes_length	{10, 25, 50, 100}	50
noise_starting_scale	{0, 0.5, 1}	1
min_exploration_noise	{0.025, 0.05, 0.075}	0.05
buffer_size	{1e5, 5e5, 1e6}	100000
episodes	{1000, 2500, 3500, 5000}	3500
policy_update_freq	{1, 5, 10}	1
flip_percentage	{0, 0.25, 0.5, 0.75, 1}	1
eta	{0, ... ,0.035}	0.0025

Table 4: Hyperparameters and their respective (tuned) values for lazy agent discussed in Section 5.3.1. Here, "..." Indicates a continuous range of values.

Hyperparameter	Space	Tuned Value
hidden_dim	{64, 128, 256}	16
batch_size	{64, 128, 256}	256
scenario_batch_size	{20, 50, 100}	1500
actor_lr	{0.0001, 0.001, 0.01}	0.001
critic_lr	{0.0005, 0.005, 0.05}	0.0005
tau	{0.01, 0.05, 0.1}	0.01
gamma	{0.9, 0.95, 0.99, 1}	0.99
uniform_episodes_length	{10, 25, 50, 100}	0
noise_starting_scale	{0, 0.5, 1}	1
min_exploration_noise	{0.025, 0.05, 0.075}	0.05
buffer_size	{100000, 500000, 1000000}	100000
episodes	{500, 1000, 1500, 2000, 2500, 3000}	1500
policy_update_freq	{1, 2, 3}	2
temperature	{1, ... ,20}	2

Table 5: Hyperparameters and their respective (tuned) values for TD3 agent in Section 5.1. Here, "..." Indicates a continuous range of values.

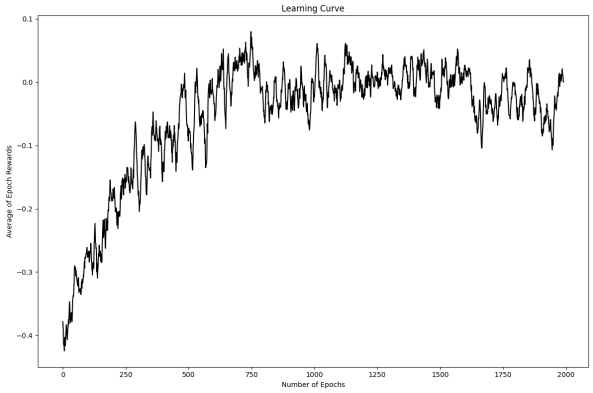
Hyperparameter	Space	Tuned Value
action_log_std	[-0.8, ..., -0.5]	-0.5269
discount_factor	[0.9 , ..., 1]	0.9195
gae_lambda	[0.9, ..., 1]	0.9923
critic_network	{64, 128, 256}	256
critic_learning_rate	{0.0005, 0.001, 0.0015}	0.001
actor_network	{64, 128, 256}	128
actor_learning_rate	{0.0005, 0.001, 0.0015}	0.001
scenario_batch_size	{30, 50, 70}	50
n_loops_scenario_set	{3, 5, 7}	5
clip_ratio	[0.1, ..., 0.2]	0.1123
n_actor_updates	{2, 3, 4}	3
n_critic_updates	{2, 3, 4}	3
update_batch_size	{32, 64, 96}	64

Table 6: Hyperparameters and their respective (tuned) values for the dynamic PPO agent in Section 5.1. Here, "..." Indicates a continuous range of values.

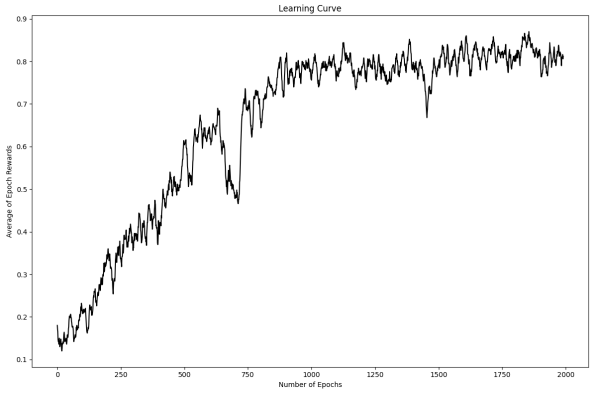
Hyperparameter	Space	Tuned Value
action_log_std	[-0.8, ..., -0.5]	-0.7824
critic_network	{32, 64, 128, 256}	32
critic_learning_rate	{0.0005, 0.001, 0.0015}	0.0005
actor_network	{32, 64, 128, 256}	64
actor_learning_rate	{0.0005, 0.001, 0.0015}	0.001
scenario_batch_size	{250, 500, 750, 1000}	750
n_loops_scenario_set	{10, 20, 30, 40, 50}	20
clip_ratio	[0.1, ..., 0.2]	0.2856
n_actor_updates	{2, 3, 4}	3
n_critic_updates	{2, 3, 4}	3
update_batch_size	{32, 64, 96, 128}	64

Table 7: Hyperparameters and their respective (tuned) values for the static PPO agent in Section 5.1. The increased batch size and scenario loops are explained in Section 4.4. Here, "..." Indicates a continuous range of values.

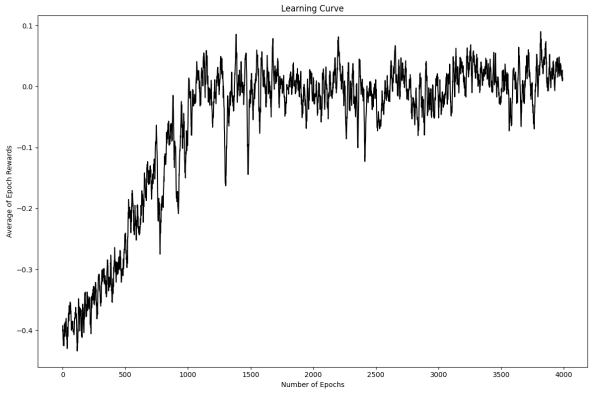
A.3 Learning curves



(a) Curriculum learning curve binary reward 2000 episodes.



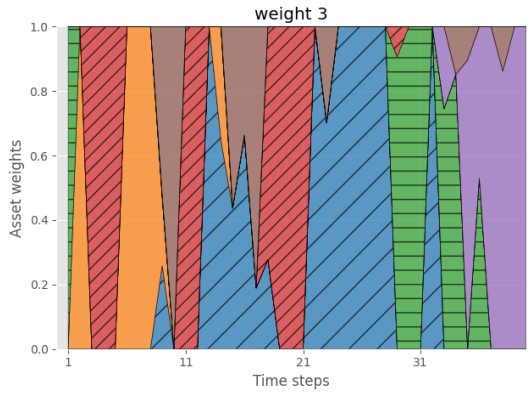
(b) Curriculum learning curve shaped reward 2000 episodes.



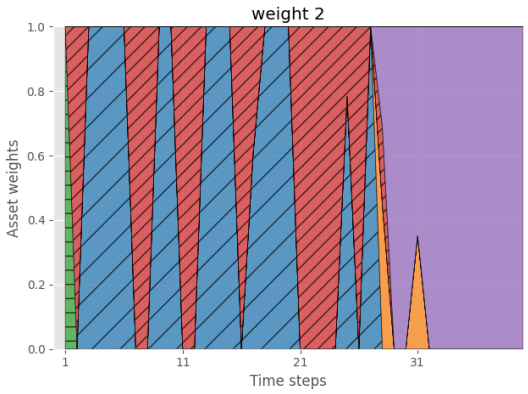
(c) Shaped reward learning curve 4000 episodes

Figure 23: Learning curves for the shaped reward agents with reward function: $binary\_reward - 5 * portfolio\_std$. (a) is trained with binary reward only, (b) is warm-started with the final agent of (a), and (c) is trained with shaped reward.

A.4 Weights



(a) Shaped reward TD3 weights for scenario 4 with curriculum.



(b) Shaped reward TD3 weights for scenario 4 without curriculum.

Figure 24: Comparison of weights over scenario 4 with/without curriculum learning.

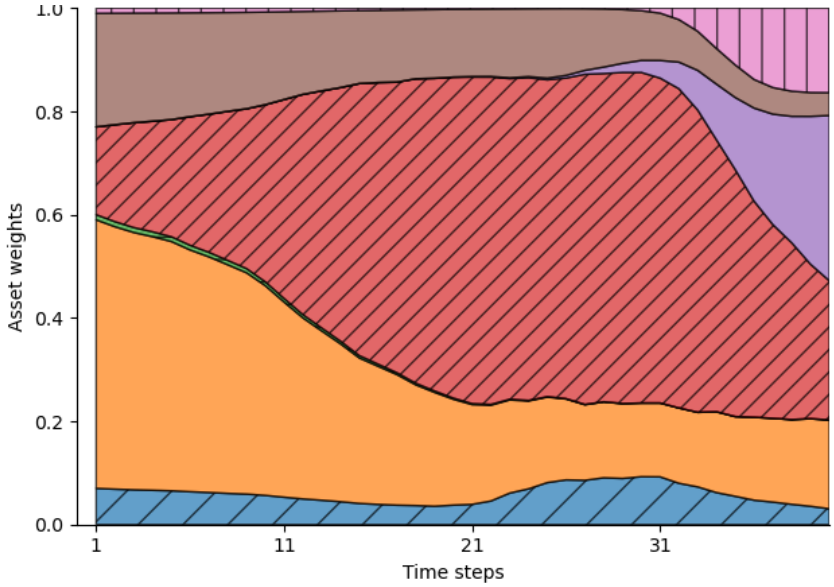


Figure 25: Regime change average scenario weights over time.

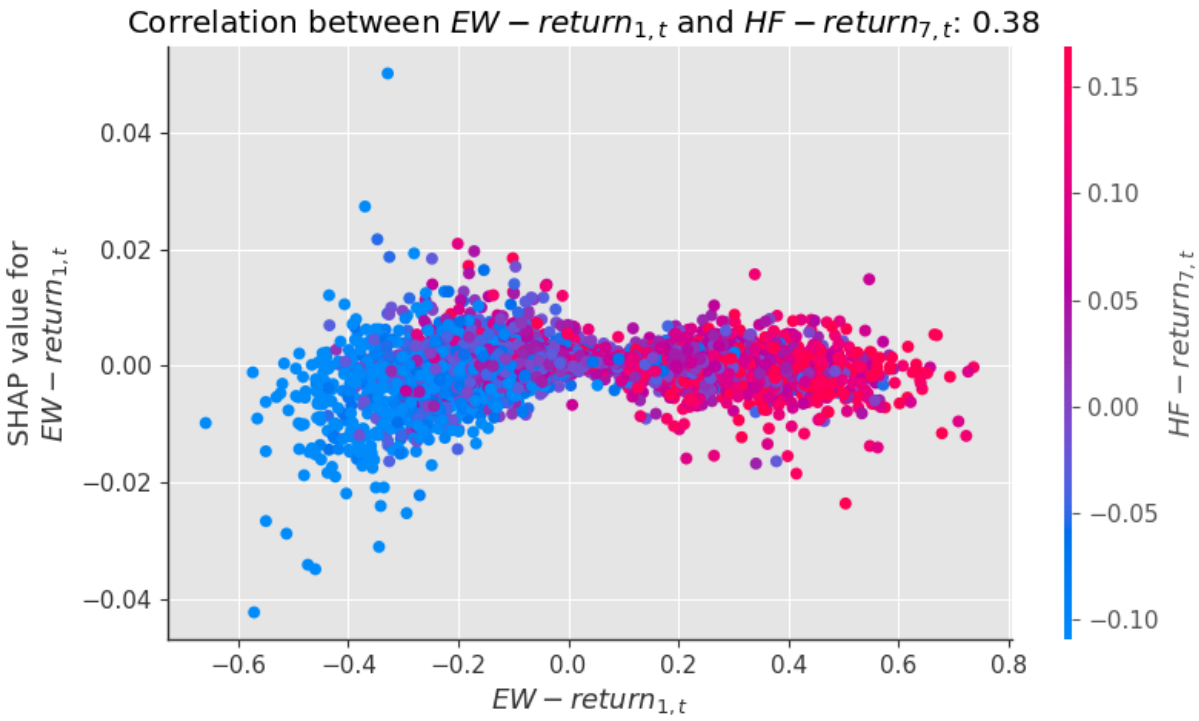


Figure 26: Example of partial dependence plot with no clear relation; EW return and HF return.