A Meta Reinforcement Learning Approach to Goals-Based Wealth Management

Sanjiv R. Das
Santa Clara University
srdas@scu.edu

Harshad Khadilkar
Franklin Templeton
harshad.khadilkar@franklintempleton.com

Sukrit Mittal
Franklin Templeton
sukrit.mittal@franklintempleton.com

Daniel Ostrov
Santa Clara University
dostrov@scu.edu

Deep Srivastav
Franklin Templeton
deepratna.srivastav@franklintempleton.com

Hungjen Wang
Franklin Templeton
hungjen.wang@franklintempleton.com

Abstract

Applying concepts related to pre-training foundation models from the realm of generative AI, we develop a meta reinforcement learning approach (denoted MetaRL) that is pre-trained on thousands of goals-based wealth management (GBWM) scenarios. This is analogous to pre-training LLMs on large training datasets. The MetaRL model enables producing near-optimal dynamic decisions for (i) goal-taking and (ii) investment portfolio selection within a few hundredths of a second in inference mode, by not requiring separate training and optimization for each new investor scenario. Using inference with MetaRL delivers expected utilities that are, on average, 97.8% of the optimal expected utilities (determined via dynamic programming). Further, the MetaRL approach can enable solving problems with larger state spaces where dynamic programming becomes computationally infeasible. *JEL codes*: G11, C61

1 Introduction

"One Ring to rule them all..." (Lord of the Rings)

This paper introduces a new meta-model approach that brings the benefits of AI and large neural networks to find optimized dynamic portfolio strategies in wealth management. We train a reinforcement learning (RL) model on thousands of individual investors' retirement scenarios, to optimize achieving their financial goals. We show that this pre-trained *meta-model*, denoted "MetaRL," can be queried via inference to determine a near-optimal strategy for a new investor's financial scenario in hundredths of seconds without re-training the model. This provides a substantial advantage over dynamic programming (DP) (see Bellman (1952) or Bellman (1966)), where each problem has to be re-solved individually, as shown in papers such as Brandt et al. (2005) and Das et al. (2020). Solving using inference with a meta-model is much faster than solving from scratch. This is especially the case as the state space grows, which slows down re-solving from scratch using RL and slows down DP so much that DP quickly becomes computationally impossible.

Working Paper, NBER/RFS Conference on Big Data, Artificial Intelligence, and Financial Economics (Cambridge, MA, December 13, 2024).

The MetaRL model mimics ideas in large language models (LLMs), which are trained on large text corpora (e.g., Brown et al. (2020); Wei et al. (2022); Izacard et al. (2022); Kojima et al. (2023); Team et al. (2024)).[1] LLMs are pre-trained and may be applied to text tasks without re-training the model. Similarly, the MetaRL model in this paper is trained on thousands of goals-based investor problems with varying financial circumstances. Analogous to the use of LLMs in inference mode, MetaRL provides zero-shot answers to both goal-taking and investment portfolio decisions. While LLMs are pre-trained using self-supervised learning, MetaRL, as we will see, is pre-trained using reinforcement learning.

MetaRL is similar to training a single model that can play many different types of Atari games (Mnih et al., 2013, 2016). A video game can be thought of as a controlled digital arena in which an agent/player tries to develop the best way to get to a destination while collecting the most rewards. Goals-based wealth management (GBWM) is similar in that the investor wants to achieve over time as many financial goals (weighted by their importance) as possible. To achieve this, the investor must manage their wealth in an optimal manner by choosing the best investment portfolios in response to market conditions and their desired goals. The investor must also make decisions regarding which goals they should obtain and which goals they should abandon so they are more likely to have the wealth needed to obtain more important future goals. To the best of our knowledge, such a meta-model has not been developed in the literature in finance for dynamic portfolio optimization with utility maximization, nor has it been used for goals-based wealth management. At a practical level, MetaRL can be deployed on thin clients for inference. Its parameters and weights can be distributed for inference, mirroring the release of open models in the realm of LLMs.

Traditional wealth management does not address each investor's financial situation individually. Instead, investors are generally bracketed into three to five risk/return appetite groups and then assigned a single, static investment portfolio to that group. This "three-to-five-sizes-fits-all" approach leaves investors dissatisfied, since it fits no one in particular. An important objective of GBWM is the customization of a dynamic portfolio strategy tailored to best address the specific goals over time of each individual investor. However, solving large-scale bespoke problems from scratch every time can be very expensive. With the MetaRL model in this paper, a one-time pre-training of the meta-model may be undertaken and then every new problem can be handled by using the pre-trained model in inference mode. That is, without re-training the MetaRL model, we can change parameters like the nature of the investor's goals (how many goals, when they occur, the cost of each goal, and the importance of each goal to the investor), the time horizon, the initial wealth, future infusions (how many infusions, when they occur, and the amount of money in each infusion) — or even the nature of the investment portfolios available to the investor, such as the number of the investment portfolios or where they lie on the efficient frontier or even the efficient frontier changing. In contrast, a change to any of the above parameters would require re-solving the entire problem if we are using dynamic programming.

We implement an interesting variant of a popular RL approach known as PPO (Proximal Policy Optimization—see Schulman et al. (2015, 2017) and Haarnoja et al. (2018)) to create the MetaRL model. The main aspects of this process are described in Section 3, with full technical details contained in the Appendix. We will see that even when all the parameters above are fixed, RL inference (with 26 state variables) is still over 100 times faster than DP (with 2 state variables) for determining the current best strategy, meaning whether or not to take a currently available goal and which investment portfolio is best to select over the next time step. Further, we will see that repeatedly querying the MetaRL model at every time step will lead to results that are, on average, 97.8% as good as the exact optimal dynamic strategy over time, which DP produces.

1.1 Previous Literature

Dynamic optimization of investment portfolio decisions over time was introduced in Merton (1969) and Merton (1971). These papers helped establish the theoretical underpinnings for life-cycle investing and consumption-portfolio decisions. They also laid the foundations for optimal lifetime investment portfolio selection under uncertainty using a continuous-time framework. Merton showed how consumption and investment portfolio choice should adapt over the life-cycle in response to changes in financial market conditions. Wealth management decisions also change with employment

[1]These models are closed (only API access) or they may be open (some of the code, training data, and model weights are publicly available).

status, health status, etc. Several additional papers comprise this literature (e.g., Browne (1999); Guidolin and Timmermann (2007); Infanger (2008); Topaloglou et al. (2008); Vo and Maurer (2013); Singhal and Biswal (2019)). Dynamic portfolio strategies with regime shifts have also been modeled (e.g., Ang and Bekaert (2002); Zhou and Yin (2003); Ang and Bekaert (2004); Brandt et al. (2005); Brandt and Santa-Clara (2006); Guidolin and Timmermann (2007); Bernhart et al. (2011); Bulla et al. (2011); Grobys (2012); Vo and Maurer (2013); Jiang et al. (2015); Dapena et al. (2019); Singhal and Biswal (2019); Bellalah et al. (2020); Lewin and Campani (2020); Das et al. (2022)). Examples of other AI applications in finance are in papers that research insurance markets (Zhang et al., 2023), explainable financial anomalies (Sabharwal et al., 2024), and credit scoring (Das et al., 2023; Chang et al., 2024), to name a few.

The target date fund or life-cycle fund concept aims to mimic optimal portfolio allocations over the life-cycle by adjusting risk exposure (stocks vs. bonds) as retirement approaches. That is, its investment portfolio strategy is based solely on the time until an investor retires, and nothing else about the investor. Subsequent research has found significant shortcomings in such a simple approach. For example, Duarte et al. (2021) used machine learning algorithms to solve for optimal investment portfolio choices over the life-cycle, accounting for variations in factors like wealth, business cycles, stock valuations etc. They found substantial gains from using customized rules, rather than simple "age-based" rules.

Chhabra (2005) proposed a more holistic wealth allocation framework that integrated factors like human capital, real estate, health, and mortality risk in addition to just financial assets. This provides a more comprehensive view of lifetime risks. This line of work eventually lead to the approach known as goals-based wealth management (GBWM). Brunel (2015) provides an overview of goals-based wealth management and proposes changes to traditional wealth advisory practices.

A series of papers (Deguest et al., 2015; Dempster and Medova, 2011; Wang et al., 2011; Pakizer, 2017; Das et al., 2018, 2020, 2023; Kim et al., 2020; Martellini et al., 2020; Parker, 2020, 2021; Mohammed et al., 2021) introduced GBWM to the retirement and pension planning industries. Stochastic dynamic programming for multi-stage problems been used to obtain solutions to many related asset-liability management optimization problems in papers such as Mulvey and Vladimirou (1992), Dempster and Consigli (1998), Consigli and Dempster (1998), Mulvey and Shetty (2004), Infanger (2008), and Topaloglou et al. (2008). But solving these large-scale, long-horizon optimization problems poses interesting computational difficulties as the state space grows.

Das et al. (2022) and Capponi and Zhang (2023) develop dynamic programming methodology for goals-based wealth management that seeks to maximize outcomes over multiple investor goals like retirement, home purchase, education, etc. This approach uses individual investor preferences about their goals to determine the optimal dynamic strategy for when to fulfill and forgo goals and what investment portfolio should be selected as time changes. The approach can be computed quickly even for many goals over different times, where goals may allow full or partial fulfillment. It computes the probabilities of attaining each goal fully or partially under the optimal dynamic strategy, so investors can ensure their preferences are accurately reflected. It significantly outperforms commonly used Monte Carlo-based static goal-taking and investment portfolio strategies and target-date funds.

Reinforcement learning (RL) has become popular recently. The classic book by Sutton and Barto (2018) introduces RL concepts like Markov decision processes and common RL algorithms like value-based and policy-based methods. The survey paper Hambly et al. (2021) reviews recent developments and applications of reinforcement learning in finance, including for optimal execution (Zheng et al., 2023; Nevmyvaka et al., 2006), portfolio optimization (Jiang et al., 2017), option pricing, market making, order routing, and robo-advising, see also Osterrieder and Gpt (2023).

RL is also being used for dynamic optimization of goals-based portfolios. Das and Varma (2020) leveraged seminal developments in papers on Reinforcement Learning, such as Sutton (1988), which introduced the temporal-difference (TD) learning algorithm, a breakthrough in prediction and control methods for reinforcement learning, and Watkins (1989), which introduced Q-learning, one of the most widely used model-free reinforcement learning algorithms. Dixon and Halperin (2020) and Halperin (2021) propose a reinforcement learning approach called G-Learner for goals-based wealth management problems like retirement planning or target-date funds, and GIRL, an inverse reinforcement learning algorithm that can infer the reward parameters of a G-Learner agent from observed behavior data, allowing it to mimic the G-Learner.

Various algorithms have been applied in RL, some based on deep learning (also known as DeepRL). Mnih et al. (2013, 2015) solved optimal playing of Atari games, demonstrating the successful application of deep neural networks as function approximators in reinforcement learning, leading to the rise of deep reinforcement learning. These works showed that a deep reinforcement learning agent could achieve human-level performance across many Atari games, culminating in the beating of the Go champion, Lee Sedol (Silver et al., 2016).

The MetaRL model in this paper comprises deep neural net policy functions that take in the values of state variables and return the concomitant policy actions. We are in a class of RL algorithms known as "actor-critic" models, where the policy function is the actor and the RL value function is the critic used to evaluate the quality of the policy function. Sutton (1988) was the first to formally introduce and analyze the actor-critic framework. This was improved in the paper by Schulman et al. (2017), which introduced the Proximal Policy Optimization (PPO) algorithm, a widely used policy gradient method for reinforcement learning. Haarnoja et al. (2018) proposed the "soft actor-critic" algorithm, a maximum entropy reinforcement learning method that is robust and sample-efficient. We implement a variant of the PPO algorithm in this paper with a separate actor and critic for the goal-taking strategy and for the investment portfolio strategy of an investor. Hence, there will be two policy functions and two RL value functions, all trained simultaneously, i.e., a dual actor-critic formulation.

In this paper, we implement our dual-PPO algorithm on thousands of investor problems to train a MetaRL model. This complements related papers in MetaRL (Finn et al., 2017; Gupta et al., 2018; Finn and Levine, 2018; Humplik et al., 2019; Lin et al., 2022), and for a survey, see Beck et al. (2023). These papers have made significant contributions to the field of meta reinforcement learning, introducing key algorithms, theoretical foundations, and novel approaches for enabling fast adaptation and generalization with reinforcement learning across different tasks and environments.

1.2 Contributions And Organization Of This Paper

Just as Large Language Models (LLMs) have opened a remarkable number of new possibilities for synthesizing written information, the MetaRL approach in this paper, which is similar in its training to LLMs, opens a number of new possibilities for helping investors with wealth management and retirement planning. The contributions and key aspects of the MetaRL approach are the following:

- **Universality:** The MetaRL algorithm covers a remarkably wide set of scenarios, and therefore, like LLMs, once it is trained, it does not need to be re-trained. It can be used in inference-only mode on new scenarios. The trained MetaRL model is not only able to work with different investors, who will have different wealths, time frames, goals, and planned infusions, but also different companies, who will use different investment portfolios with different assumptions about their expected returns and volatilities, which may change over time. In contrast, DP must be rerun whenever the investor or the company changes anything in their scenario.
- **Speed:** Because we can use RL inference with the already trained MetaRL model but have to rerun DP, RL inference can determine both what current goals to fulfill or forgo and the best investment portfolio for the next time step over 100 times faster than DP can. We note this is the case even though the MetaRL model has 26 state variables and DP only has 2 state variables. Determining some other properties of the solution, such as the expected utility or the probability of attaining future goals, requires repeatedly querying the MetaRL model, leading to essentially the same runtime as DP for simple cases, but progressively better runtimes compared to DP as the scenarios become more complicated.
- **Accuracy:** The fact that RL inference uses scenarios that differ from the scenarios on which the MetaRL model is trained may lead to concerns about how optimal the RL inference solution is. However, we will see that it is very accurate: Over 66 new test scenarios, many of which are well outside the training scenarios, we will find that, on average, the RL inference solution achieves 97.8% of the expected utility from fulfilled goals compared to the exact optimal solution, which can be obtained by DP for these examples. Even the worst of these 66 scenarios still achieves 91.7% of the expected utility from fulfilled goals.
- **Extendibility:** Perhaps more important than all of the above points is the fact that the MetaRL algorithm is extendable to more complex GBWM problems that DP cannot address.

In Subsection 5.2, we consider the GBWM problem in the context of stochastic inflation. For DP, this means changing the state space from 2 variables to 4 variables. Due to the so-called "curse of dimensionality," the computational time for DP increases exponentially with the number of state space variables. Having 4 or more state space variables is too many for DP to be a computationally feasible method. But RL inference with the MetaRL model, as we have said, can work with 26 state variables blindingly quickly. In fact the change to 27 state variables needed to address stochastic inflation creates *no* discernible change to the RL inference computational time. This means that entire classes of GBWM problems with more sophisticated modeling that were out of reach with DP approaches can now be explored and solved with RL, and, just as importantly, the computational speed enabled by the MetaRL approach means the solutions to these problems can be made quickly accessible to investors.

The paper proceeds as follows. In Section 2, we explain the GBWM problem under the assumption that the investor has, at most, one all-or-nothing goal each year. In Section 3, we give a brief overview of the features of the MetaRL approach we use to solve this problem, relegating most of the technical details about the MetaRL model to Appendices A and B. Experimental results and examples showing the runtimes, the produced strategies, and the accuracy of the MetaRL approach are in Section 4. Extensions of the model are taken up in Section 5, where concurrent and partial goals are introduced in Subsection 5.1, while Subsection 5.2 extends the GBWM problem to include the effect of stochastic inflation, illustrating how the algorithm is easily extended to handle more state variables. A concluding discussion is provided in Section 6.

Detailed technical exposition is provided in the Appendices: Appendix A delineates the RL environment used for the GBWM problem when there is, at most, one all-or-nothing goal each year, providing specifics about the state space, actions, and rewards, as well as other details used to create the MetaRL algorithm. Appendix B specifies the neural net architecture and how MetaRL pre-training and inference are implemented. Appendix C describes the test suite of 66 investor scenarios used to gauge the quality of the MetaRL model. Mirroring Subsections 5.1, and 5.2, details for extending the MetaRL method to GBWM problems that allow for concurrent and partial goals are given in Appendix D and for accommodating stochastic inflation in Appendix E.

2 Defining The GBWM Problem

2.1 Decisions Facing Goals-Based Investors

An investor using goals-based wealth management (GBWM) wishes to attain as many of their financial goals—weighted by their importance to the investor—as possible over the time horizon of their portfolio, which may mean throughout their projected lifetime. To weight the importance of each goal to an investor, a utility (reward) is assigned to each goal. Realistic methods for determining these utilities come from the investor balancing the probabilities of attaining their goals against each other, as discussed in Das et al. (2022) and Das et al. (2023). The investor is then looking to make decisions that will optimize their "expected attained utility," meaning the expected value of the utilities of attained goals in the present and future.

To optimize the expected attained utility, there are two decisions that must be optimized at each point in time: (1) whether to pay the cost needed to fulfill a currently available goal and attain the utility assigned to that goal and (2) to determine which investment portfolio (level of risk and return) to hold for the next time step. Because we are in a dynamic setting, where the optimization must be in light of knowing that these decisions can be different in every future time step, this is a complicated optimization problem. Fulfilling a goal requires there to be sufficient wealth available, but even when there is, the investor may be better off forgoing the current goal to increase their probabilities of fulfilling more important future goals.

The investment portfolio decision also depends on how much wealth the investor has. If the portfolio is not doing well, the investor may need to select more aggressive investment portfolios in a measured way to increase the probability that they can achieve more future goals. On the other hand, if the portfolio is doing very well, the investor may wish to move to more conservative investment portfolios to avoid large losses that could adversely impact their ability to fulfill future goals.

2.2 Parameters Defining A Single Scenario

Each investor has their own scenario, which must be defined so that the GBWM solution can be optimized for their individual circumstances. The parameters that must be specified to define a single investor scenario are the following:

1. *Time horizon and time steps:* The investor must specify T, the total number of time steps, and h, the amount of time between time steps. We will use $t = 0, 1, 2, ..., T$ to denote the index of the time step. Further, because h is fixed to equal one year in all of this paper's experiments, we refer to t as the time (in years) and T as the time horizon (in years) of the portfolio.
2. *Initial wealth and future wealth infusions:* The investor must specify their initial wealth, $W(0)$, where $W(t)$ denotes their wealth at time t. Wealth infusions can be specified at any time steps. The amount of money in an infusion at time step t is denoted as $I(t)$ (where $I(t) = 0$ if there is no infusion in year t), or, collectively, as the T-vector $\mathbf{I}$.
3. *Goals:* For simplicity, we will, for the moment, assume there can be, at most, one all-or-nothing goal available at each time step. (We will extend this to allowing concurrent and partial goals in Subsection 5.1, which will require extending the MetaRL model approach as specified in Appendix D.) With this assumption, the investor can specify $C(t)$, the cost of the goal available in year t (where $C(t) = 0$ if there is no goal available in year t), or, collectively, as the T-vector $\mathbf{C}$. The corresponding utilities are then specified by $U(t)$ (where, again, $U(t) = 0$ if there is no goal in year t), or, collectively, as the T-vector $\mathbf{U}$. Note that an investor would be just as happy to attain one goal with a utility of 4 and another with a utility of 7 as they would be attaining just one goal with a utility of 11.
4. *Available investments:* The investor (or, more likely, the financial services company working with the investor) must also specify the expected return and the volatility of each of the P available investment portfolios to the investor, which we denote respectively by μ_p and σ_p, where $p \in \{0, 1, \ldots, P-1\}$, or, collectively, by the P-vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. The investment portfolio choice, p, can be changed in each time step.

These parameters that define a single scenario are summarized in Table 1.

Table 1: Scenario parameter variables: This list of variables constitutes a complete description of the parameters that define a single scenario.

Symbol	Length	Explanation
T	1 (int)	Number of time steps (also called the "time horizon")
h	1 (float)	Time between time steps (assumed to be a year throughout this paper)
$W(0)$	1 (float)	Initial wealth at time $t = 0$
$\mathbf{I}$	T (float)	Planned infusions of wealth at each time step where $t > 0$
$\mathbf{U}$	T (float)	Utility of each time step's goal where $t > 0$
$\mathbf{C}$	T (float)	Cost of each time step's goal where $t > 0$
$\boldsymbol{\mu}$	P (float)	Expected return of each of the P investment portfolios
$\boldsymbol{\sigma}$	P (float)	Volatility of each of the P investment portfolios

2.3 Mathematical Formulation Of The Problem

The two decisions the investor must make at each time step are defined mathematically by

1. *The goal-taking decision:* At each time step, t the investor first decides whether or not to take the goal if one is available. We denote this decision by $g(t) \in \{0, 1\}$, where $g = 0$ means the investor decides not to take the goal (or can't afford it) and $g = 1$ means the investor decides to take the goal.
2. *The investment portfolio decision:* After the goal-taking decision is made, so the investor knows how much of their wealth remains, they decide which investment portfolio to select

for the next time step. We denote their choice by $p(t) \in \{0, 1, ..., P-1\}$, where μ_p and σ_p are the expected return and volatility of investment portfolio p.

With the scenario parameter variables from Subsection 2.2 and the two decision variables defined, we can mathematically define the goal of the GBWM problem, which is to optimize the expected attained utility; that is,

$$\max_{g(t),p(t),t=\{0,...,T-1\}} E\left[\sum_t g(t) \cdot U(t)\right], \tag{1}$$

subject to wealth transitions, which we assume are governed by geometric Brownian motion, meaning

$$W(t+1) = [W(t) + I(t) - g(t) \cdot C(t)] \cdot \exp\left[\left(\mu_{p(t)} - \frac{1}{2}\sigma^2_{p(t)}\right) h + \sigma_{p(t)}\sqrt{h} \cdot Z\right], \tag{2}$$

where $Z \sim N(0, 1)$ is a standard normal random variable.

3 Overview Of The MetaRL Approach And A Comparison To The DP Approach

We give a brief overview of the states, actions, and rewards used to train the MetaRL model. Far more detail about this and other aspects of the training are available in Appendices A and B.

3.1 MetaRL State Space

At first thought, we might consider choosing the state space variables to simply be the scenario parameter variables given in Subsection 2.2. However, this direct form of the variables is neither particularly digestible nor helpful for efficiently training the MetaRL model. Instead we need to use these scenario parameter variables to create state variables that transmit information to the MetaRL model in its most useful, essential, and generalizable form. Our state variables will be normalized to be both dimensionless and generally take values between 0 and 1 to aid the gradient ascent algorithm that underlies training the model, which works best when all the state space variables have similar scalings.

We next give a brief overview of the 26 state variables used by our MetaRL model. These are explained in far more detail in Appendix A.3. The first state variable is the current time, t, normalized (that is, divided) by T, the time horizon. The second and third state variables are the current (time t) wealth $W(t)$ normalized by the costs of current and future goals discounted to time t dollars using a pessimistic investment return projection for one state variable and an optimistic investment return projection for the other state variable.

The next three sets of variables describe the utilities and costs of present and future goals. The first set is, essentially a vector of the utilities in each present and future year normalized by the sum of these utilities. Because the length of this vector would vary with time t and we need a vector of constant length for training, we instead clump these results into time blocks. In our case, we choose 7 time blocks corresponding to 0, 1, 2, 3, 4-5, 6-9, and 10 or more years into the future. So the 6-9 time block contains the sum of the utilities for goals that are between 6 and 9 years into the future divided, as before, by the sum of the utilities of all current and future goals. For the costs, we do the same thing, but we again use the costs discounted to time t dollars both using a pessimistic and an optimistic investment return projection. This gives 7 normalized state variables for the utilities and $2 \times 7 = 14$ normalized state variables for the costs.

The remaining two state variables are closely connected to the two decisions. The first, connected to goal-taking, gives a very rough estimate for comparing the effect on the overall attained utility if the investor takes the current (time t) goal versus if they do not take it. This variable is normalized between 0 and 1, where the closer it is to 0, the more evidence there is to forgo the goal, and the closer it is to 1, the more evidence to take the goal. The second variable, connected to selecting the investment portfolio, considers the effect of freezing over time each of the P investment portfolios on the overall attained utility, and then reports the most effective of these investment portfolios in a normalized manner. Key within this is ordering the P investment portfolios to change from being more conservative to being more aggressive as the index $p \in \{0, 1, ..., P-1\}$ increases.

3.2 MetaRL Actions And Decisions

Appendix A.2 shows how the MetaRL algorithm uses the 26 state variables from the previous subsection to produce two continuous *actions*, $a_g(t) \in [0,1]$ and $a_p(t) \in [0,1]$, that directly determine the discrete *decisions* $g(t) \in \{0,1\}$ and $p(t) \in \{0,1,...,P-1\}$ via monotonic mappings that are given in equations (4) and (7). There are two reasons the MetaRL actions are chosen to be continuous variables instead of discrete: (1) If $a_p(t)$ were discrete, we would need to set $a_p(t) = p(t)$, which would mean needing to retrain the MetaRL model each time P changed, which is undesirable. (2) If $a_p(t)$ were a discrete action, there would be no relationship between the investment portfolios from the MetaRL perspective, which would loose the important connection that the investment portfolios become more aggressive as p increases.

3.3 MetaRL Rewards

Again, at first thought, we might want to select the expected attained utility given in equation (1) to be the reward for the MetaRL model to optimize. Unfortunately, this is an inadequate reward for training because it produces a sparse rewards signal, making it insufficient for training. As explained in detail in Appendix A.4, we define the extrinsic reward to be the expected attained utility, and the reward is this extrinsic reward plus an intrinsic reward (defined in Appendix A.4), which produces a dense rewards signal. While we de-emphasize the intrinsic reward as training progresses, it is never completely removed.

3.4 MetaRL Versus DP

For any given scenario, DP uses a backwards pass, meaning a computation that starts at the final time $t = T$ and works backwards to the starting time $t = 0$, to determine the optimal decisions and the optimal "DP value function," meaning the optimal expected attained utility from present and future goals. The backwards pass computes the optimal decisions and the optimal DP value function at any state, meaning any given time t and wealth $W(t)$ at that time. We note that the state space in DP contains just these two state variables: t and $W(t)$.

MetaRL is different. First, the definition of the "RL value function" is different from the "DP value function" in two ways: (1) The RL value function uses the MetaRL reward, which adds both the extrinsic reward and the intrinsic reward, whereas the DP value function is based solely on the extrinsic reward, meaning the expected attained utility. (2) The RL value function allows the current action variables to be inputs so their effect on the rewards can be gauged, whereas the DP value function requires the decisions to be set at their optimal values.

We emphasize a second key difference between MetaRL and DP: MetaRL uses 26 state variables, whereas DP only uses 2. There is a considerable advantage in using 26 state variables: they encompass the scenario parameters as well as t and $W(t)$, and this is why the MetaRL model does not have to be re-trained when we change scenarios. The fact that RL can work with 26 (or more) state variables in a reasonable amount of time is its key advantage over DP. We use 2 state variables with DP because we have to. We can't use more due to the curse of dimensionality on DP's computational time, discussed earlier. This restriction means that DP must be rerun any time we change scenarios, as well as the fact that it cannot be applied to more sophisticated GBWM models that require more than 2 state variables to describe its state, even within a specific scenario (like in Subsection 5.2, which considers the effect of stochastic inflation).

4 Results

As discussed in Appendix B, and in particular Appendix B.2, the meta-model is trained on a wide variety of randomly generated GBWM scenarios, numbering in the thousands, also known as "curriculum learning" (Bengio et al., 2009). This is an approach to learning where the model is trained using a range of different scenarios. This pre-trained GBWM model, which we will call the "MetaRL" model, is the RL analogy to pre-trained language models like BERT and other well-known large language models (LLMs).

In this section, each new investor scenario is solved using the pre-trained policy functions from the MetaRL model in inference mode, which we will refer to as "RL inference." We show how RL

Table 2: Descriptive statistics for the test suite of 66 GBWM scenarios. The suite comprises 66 new investor problems with varying horizons, initial wealth levels, number of goals, goal costs and utilities, and number of infusions.

	mean	std. dev.	min	25%	50%	75%	max
Time steps (in years)	38	26	3	16	30	60	100
Initial wealth	93	23	12	100	100	100	126
Number of goals	16	23	1	1	4	10	60
Cost of all goals	624,535	3,452,455	34	250	750	1830	2,000,000
Number of infusions	9	19	0	0	0	1	99
Total infusion amount	29	80	0	0	1	21	588
First infusion time	5	11	0	0	0	1	49

inference can quickly and accurately solve GBWM problems for new scenarios. We first describe (in Subsection 4.1) the test suite of 66 GBWM scenarios that we will use in this section. We then explore the speed of using RL inference (in Subsection 4.2), the goal-taking and portfolio investment decisions determined by using RL inference (in Subsection 4.3), and the average attained utility these RL inference determined decisions lead to (in Subsection 4.4). In all cases, we will compare these results to the optimal decisions that can be generated using dynamic programming (DP), noting that DP must be rerun for each of the 66 GBWM scenarios, as just discussed in Subsection 3.4.

4.1 Test Suite Used For Evaluating The Performance Of The MetaRL Model

Once the pre-trained MetaRL model is ready, it is used for inference on a test suite of previously unseen GBWM scenarios to evaluate its performance. The test suite comprises 66 new investor scenarios with varying parameters for the time horizon, the initial wealth, the number of goals along with the timing, cost, and utility of each goal, and the number of infusions along with the timing and amount (i.e., worth) of each infusion. A list of these 66 problems is available in Appendix C. Table 2 shows some descriptive statistics for these 66 problems. We note that the parameters in many of these 66 cases are outside the range of the parameters that were used to train the MetaRL model, which are given in Algorithm 5 in Appendix B.2. For example, a number of our 66 cases use a time horizon of 100 years, whereas the training only uses time horizons up to 50 years.

4.2 Computational Runtime Analysis: Comparing RL Inference To DP

Solving simple GBWM problems using dynamic programming (DP) can be made quite efficient by using techniques like vectorization and just-in-time compilation to C executable code (as has been done for all DP computations in this paper). However, RL inference is quite fast even without these runtime efficiency techniques. More importantly as indicated before, as the dimension of the state space (i.e., the number of state space variables) for DP problems grows, DP will quickly become infeasible due to exponential computational time growth, whereas RL inference's computational time does not grow exponentially. In an application where RL models are intended to solve hundreds of problems in a few minutes, such as in a robo-advising setting, the algorithmic and corresponding software improvements in computational speed that RL inference can provide becomes crucial. In this section, we analyze the computational time needed to run RL inference on the 66 scenarios in our test suite from Subsection 4.1 using the already obtained MetaRL model versus the computational time needed to run DP on the same test suite.

Two types of computational runtime comparisons may be made: (1) comparing the average time for policy inference at a given time for a given problem, meaning we want to determine the optimal actions/decisions to make at the current time, and (2) comparing the average time it takes to determine the optimized expected attained utility at a given time for a given problem, meaning we want to determine the DP value function at the current time. We will see that using RL inference is over 100 times faster than DP in the first case, while the computational times for RL inference and DP are comparable in the second case. For the remainder of this subsection, we explore these two cases.

Computational time to determine the optimal actions/decisions at the current time: To determine the optimal goal-taking decision and the optimal investment portfolio decision at the current time requires DP to run its entire backwards pass to solve for the individual investor's specific scenario. For RL inference, on the other hand, the problem does not need to be solved at all, as the MetaRL

algorithm is already trained over a representative sample of scenarios and can be directly used to determine the optimal actions for goal-taking and selecting an investment portfolio at a given time and wealth. We note that these *actions* are continuous variables in RL, which are easily (and essentially instantaneously) converted to the final (discrete) goal-taking and investment portfolio *decisions* via equations (4) and (7) in Appendix A.2.

Because RL inference only requires querying the MetaRL model once to obtain these actions, it is far faster than DP, as we can clearly see by looking at the "mean" column in Table 3 (noting that results in the table are in milliseconds). At times when there is no goal available, so only an investment portfolio choice must be made, RL inference is approximately twice as fast as when both a goal-taking choice and an investment portfolio choice must be made. This can be seen by comparing the mean of 9.277 msec to the mean of 20.94 msec in the table. DP takes an equal amount of computational time either way, but we note that, approximately on average, it is $2198/20.94 = 105.0$ times longer than RL inference at a point with a goal and $2198/9.277 = 236.9$ times longer than RL inference at a point without a goal. That is, even for this GBWM problem, where there are only 2 state variables used in dynamic programming and 26 state variables used by RL inference with the MetaRL approach, we find that RL inference produces optimized decisions more than 100 times faster than DP can.

Table 3: Average time (in milliseconds) to determine the current optimal actions/decisions over the 66 test suite cases, or, for the second row, the 48 of these 66 cases that have more than one goal. Note from the mean results that DP takes an approximate average of $2198/20.94 = 105.0$ times longer than RL inference at a point with a goal and $2198/9.277 = 236.9$ times longer than RL inference at a point without a goal. See the text for a more thorough explanation of the calculations presented in this table.

	# of cases	mean	std. dev.	min	25%	50%	75%	max
Time (in msec) using RL inference to determine optimal portfolio actions only	66	9.277	2.76	3.80	8.95	9.76	10.49	18.40
Time (in msec) using RL inference to determine optimal portfolio and goal actions	48	20.94	1.34	16.5	20.2	21.1	21.6	23.9
DP backward pass computational time (in msec)	66	2198	2255	83	579	1248	3720	8012

To more precisely explain the calculations presented in Table 3, we describe in more detail how, as an example, we computed the "50%" column, which gives the median results: (1) The first row's result means half of the 66 cases require an average of 0.00976 seconds or less to run RL inference to estimate the investment portfolio decision, where this average is over all times within the case where there is no goal. (2) The second row's result means half of the 48 cases in the 66 test suite examples that have at least one goal before the final time step require an average of 0.0211 seconds or less to run RL inference to estimate both the goal-taking and investment portfolio decisions, where the average is over all times$< T$ within the case where there is a goal. (3) The third row's result means that half of the 66 cases require an average of 1.248 seconds or less to run DP's backwards pass, which determines the optimal goal-taking and investment portfolio decisions, where the average is over all times $(0, 1, ..., T-1)$ within the case.

Computational time to determine the optimal expected attained utility at the current time: DP uses the same backwards pass to determine the optimal expected attained utility (i.e., DP's value function) at the current time that it did to determine the optimal goal-taking and investment portfolio decisions in the case above, so the time DP takes is what is shown in the last row of Table 3. Using RL inference, on the other hand, to find the optimal expected utility is more complicated. As we discuss in more detail in the next subsection, we repeatedly query the MetaRL algorithm to determine the optimal decisions over a dense grid in the (t, W) state space plane. As we explain in more detail in the subsection after that, we use these decision graphs to specify decisions throughout the evolution of each of 10,000 simulated Monte Carlo trajectories, and then average these 10,000 attained utilities to estimate the optimized expected attained utility. This is necessary because the MetaRL algorithm only provides actions, not estimates for the DP value function.[2] This leads to DP and RL inference

[2]Readers familiar will RL might wonder if the critic network can provide the estimate of DP's value function, however, a broad range of empirical studies of actor-critic architectures have found that the values learned

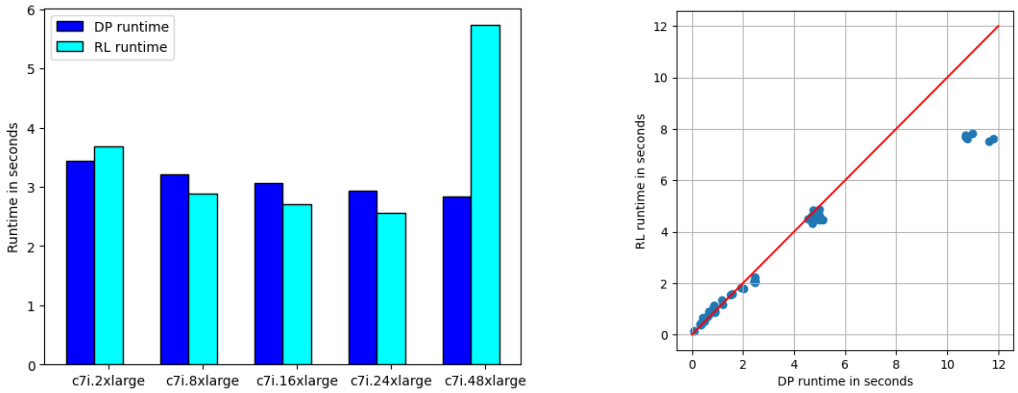


Figure 1: Left panel: Comparison of average runtimes (in minutes) over the 66 test suite GBWM scenarios for DP and RL inference on various hardware. Compute-optimized machine instances on AWS were used for these experiments. (The prefix 'c' in the 'c7i.*' stands for compute-optimized.) The details of these machines are as follows: (i) c7i.2xlarge – 8 CPUs (1 socket, 4 cores, 2 threads, 32GB RAM); (ii) c7i.8xlarge – 32 CPUs (1 socket, 16 cores, 2 threads, 64GB RAM); (iii) c7i.16xlarge – 64 CPUs (1 socket, 32 cores, 2 threads, 128GB RAM); (iv) c7i.24xlarge – 96 CPUs (1 socket, 48 cores, 2 threads, 192GB RAM); (v) c7i.48xlarge – 192 CPUs (2 sockets, 48 cores, 2 threads, 384GB RAM). Right panel: Plot of each of the 66 test suite GBWM scenarios comparing their runtimes (in minutes, with the c7i.24xlarge machine) using DP's backwards pass versus using RL inference (and 10,000 Monte Carlo simulations).

having comparable runtimes for calculating the optimal expected attained utility starting at the current time, as can be seen in Figure 1, which we discuss next.

In addition to analyzing algorithms and their software implementation, it is important to assess the role of hardware on runtimes. The left-side plot in Figure 1 shows how the choice of server and its configuration matters. This plot clearly shows that DP and RL inference have comparable runtimes. The main difference between the two comes from the number of CPU cores. At first, as the number of CPU cores increases, RL inference's runtime becomes lower than DP's. However, simply choosing a large machine does not always mean a better runtime, as can be seen when the `c7i.48xlarge` machine is chosen and RL's runtime rises sharply. This sudden rise is because the `c7i.48xlarge` machine has 2 sockets (split chip), and the cross chip bits traffic slows down RL inference. We therefore have run all of our problems in this paper (both DP and RL inference) on the `c7i.24xlarge` machines, since, from the plot, we see that it gives the best combination of DP and RL inference runtimes.

Overall, we see that RL inference runtimes are quite fast, averaging a little better than the 2.198 seconds DP requires for our 66 test suite of GBWM scenarios. A comparison of the runtimes for each of these 66 test problems is shown in the right-side plot in Figure 1. Since the points that are the farthest to the right on the plot correspond to larger problems (longer horizons, more goals) that require greater runtimes, we see from Figure 1 that RL inference becomes progressively better than DP in terms of runtime as the problems become larger. This can also be seen in Table 3 by comparing the max and 75% columns to the median, and noting that the significant increase seen in the DP computational times is not reflected in the RL inference computational times.

4.3 Decision Analysis: Comparing RL Inference To DP

We next look at how the decisions determined using RL inference compare to DP's optimal decisions. DP determines its optimal decisions over a grid in the (t, W) state space plane. We then use RL inference to determine both the goal-taking decision and the investment portfolio decision at each point in the same (t, W) grid. To speed computational time, the MetaRL algorithm is queried in parallel using all the grid's wealth values at once for each given time t. For any specific scenario, the resulting decisions can be compared by computing heatmaps over the (t, W) grid using both DP and RL inference for comparing either the goal-taking decisions or the investment portfolio decisions.

by the critic are indicative rather than exact. For example, see Section III-D of `https://hal.science/hal-00756747/document` or page 2 of `https://arxiv.org/pdf/1910.08412`.

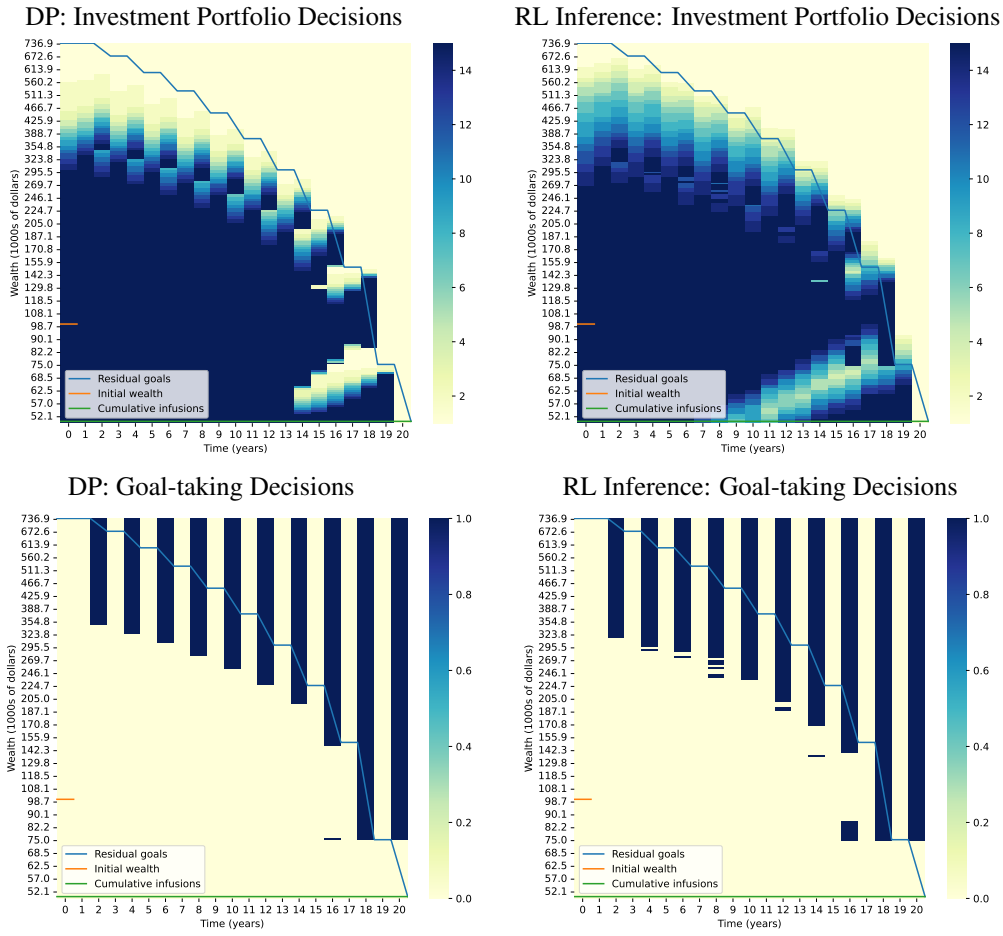


Figure 2: Heatmaps comparing the investment portfolio decisions and goal-taking decisions made by DP versus RL inference for test suite case 20. In the investment portfolio plots, the darker the color, the more aggressive the investment portfolio. In the goal-taking plots, the dark colored bars are regions in which the available goal is taken. The initial wealth, 100 (thousand) dollars, is denoted by the orange bar at the initial time. The jagged blue line denotes the cost to fulfill all the remaining goals.

We demonstrate this with two representative examples from our 66 test suite scenarios. The first example is case 20 of the 66 cases. From Table 8 in Appendix C, we see that in this case: (1) There are 20 time steps (i.e., 20 years). (2) The initial wealth is 100 (thousand) dollars. (3) At each even time step (i.e., 2,4,...,20), there is a goal available that costs 75 (thousand) dollars, which, if taken, creates a utility of 1 for the investor. (4) There are no infusions. A comparison of the decisions made by RL inference versus DP can be found in Figure 2. In this case the optimal expected attained utility (i.e., the DP value function) is 4.10 according to the DP backwards pass calculation. In the next subsection, we will discuss how we estimate the optimal expected attained utility using the RL heatmaps given here, which will produce an average attained utility of 4.02.

The second example is case 57 of the 66 cases. From Table 9 in Appendix C, we see that in this case: (1) There are 60 time steps (i.e., 60 years). (2) The initial wealth is 100 (thousand) dollars. (3) At each time step $t = 1, 2, ..., 60$, there is a goal available that costs t (thousand) dollars, which, if taken, creates a utility of $100 - t$. (4) In each time step $t = 1, 2, ..., 59$, there is an infusion worth $\frac{10}{59}1.03^t$ (thousand) dollars. A comparison of the decisions made by RL inference versus DP can be found in Figure 3. In this case the optimal expected attained utility (i.e., the DP value function) is 3128 according to the DP backwards pass calculation. Using the next subsection's method, our estimate for this value using the RL heatmaps will produce an average attained utility of 3090.

In both examples, we see that the RL inference graphs retain remarkable amounts of details that are shown in the optimal DP graphs. Because of this, it is no surprise that the estimate of the expected

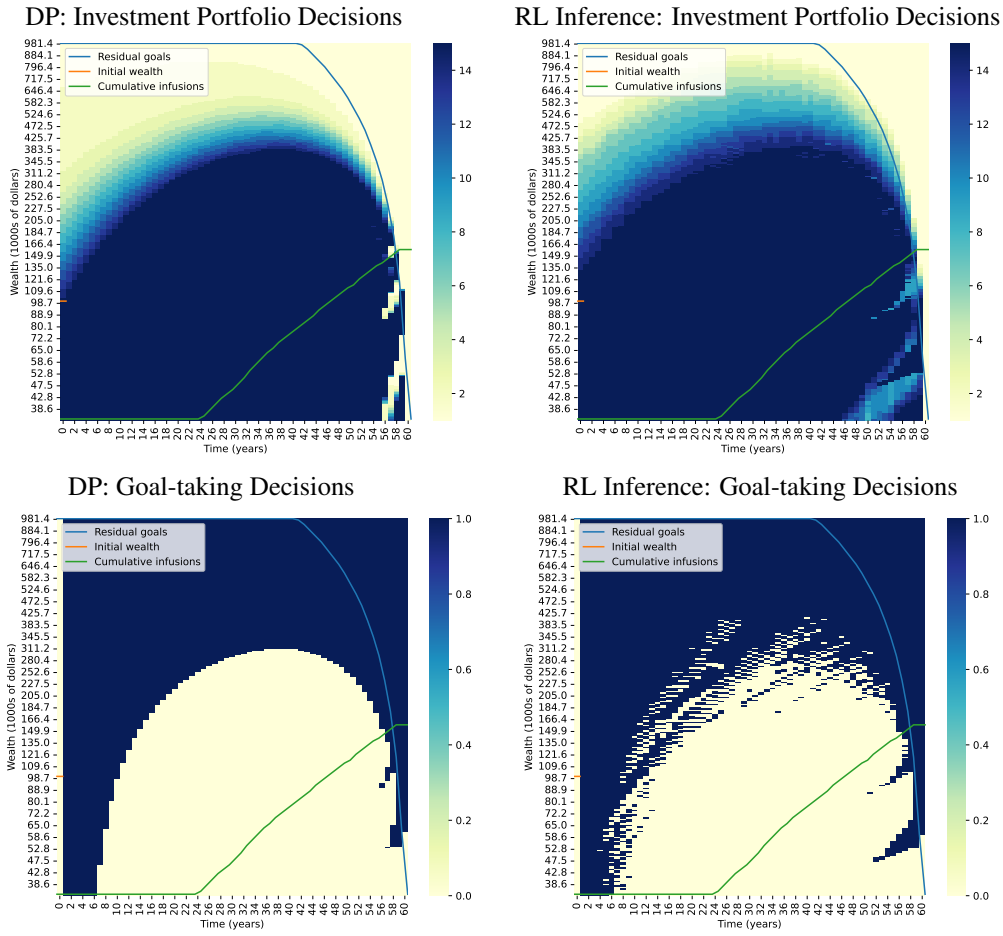


Figure 3: The same information shown in Figure 2, but for case 57 instead of case 20. Note that the green line representing cumulative infusions and the blue line representing the cost to fulfill the remaining goals are both truncated by the vertical axis's limited range.

attained utility using RL inference is so close to its optimal value as determined by DP. We next show how this estimate is formed.

4.4 Expected Attained Utility Analysis: Comparing RL Inference To DP

How do we use RL inference to estimate the expected attained utility? And how good is this estimate?

To answer the first question, after the process in the previous subsection is run to use the MetaRL algorithm to create decision heatmaps, we run 10,000 Monte Carlo trajectories, corresponding to 10,000 different histories for the value of Z at each time step in the geometric Brownian motion model given in equation (2). At each time step for each trajectory, the goal-taking decision (if there is one) and the investment portfolio decision are taken from the nearest wealth grid point to the trajectory's wealth at that time in the heatmap. These 10,000 trajectories are run in parallel in each time step to save computational time. The RL inference estimate for the expected attained utility is the average of these 10,000 attained utilities.

We look to compare this RL inference estimate to the actual optimal expected attained utility determined by DP for each of the 66 problems in the test suite given in Subsection 4.1. We wish to define the "*RL-Efficiency*" to equal to the ratio of the RL inference estimate to the DP result. In theory, this ratio should always be less than or equal to 1, because DP produces the optimal result. However, because the RL inference estimate is determined by using 10,000 different Z histories, it is possible that these Z histories will lead to better than average returns, meaning an RL-Efficiency greater than one is possible. To minimize this, when we compute the expected attained utility for DP,

we look for an apples-to-apples comparison with RL inference. Therefore, instead of taking the value function produced by DP, we instead use the optimal heatmaps that DP produces and use these on the same 10,000 trajectories (with the same 10,000 histories for the value of Z at each time step) that we used with RL inference. We then take the RL inference estimate and divide it by this DP produced number to define the RL-Efficiency.

We note that this still does not guarantee that the RL-Efficiency will not exceed 1, since the slightly suboptimal RL inference strategy may happen to work better for the specific 10,000 paths chosen, but this occurrence will be minimized by using the apples-to-apples strategy above. In Table 4, we present statistics for the RL-Efficiencies for each of the 66 test suite problems. We note that the maximum RL-Efficiency is 0.999, below 1, whereas if we had defined the RL-Efficiency using the DP value function in the denominator, the maximum ratio is 1.021, which would distort the results, making RL inference's estimate of the optimal expected attained utility look better than it actually is.

Table 4: RL-Efficiencies for the 66 test suite examples in Subsection 4.1. The RL-Efficiency measures how optimal the RL inference solution is by seeing how close the RL-Efficiency is to 1. More technically, for each test suite example, the RL-Efficiency is the ratio of the attained utility using decisions from RL inference averaged over 10,000 Monte Carlo simulations over the attained utility using the optimal decisions determined from DP averaged over the same 10,000 Monte Carlo simulations.

	mean	std. dev.	min	25%	50%	75%	max
RL-Efficiency (10,000 MC simulations)	0.978	0.016	0.917	0.975	0.983	0.987	0.999

We note in Table 4 that the mean RL-Efficiency of the 66 test suite examples is 0.978. But how stable is this result? In particular, is 10,000 a sufficiently high number of Monte Carlo trajectories? To test this, we computed the mean RL-Efficiency of the 66 test suite examples 30 times using 10,000 simulations each time. The mean of these 30 times was 0.978293 with a standard deviation of 0.000380, meaning 0.978 is a very stable number, and 10,000 Monte Carlo trajectories gives almost three digits of accuracy for the mean RL-Efficiency.

The fact that our mean RL-Efficiency is 0.978 answers the second question given at the beginning of this subsection: The estimate of the optimal expected attained utility starting at the initial time using our RL inference method is quite close to the optimal answer produced by DP. This is a particularly strong result given that we are just using RL inference without training on the specific scenario being considered. There are other advantages of our meta-model approach. The pre-trained MetaRL model can be pushed to thin clients for inference. It also enables distribution of the trained policy functions (parameters and weights) without revealing the code for the training procedure and the data and problems on which the MetaRL model has been trained. This open model distribution mirrors the release of open models in the realm of LLMs.

5 Extensions Of The MetaRL Model

5.1 Incorporating Concurrent And Partial Goals

Up to this point, we have restricted ourselves to at most one all-or-nothing goal each year. In Appendix D, we show how the metaRL model can be extended to more than one concurrent goal and to allowing some or all goals be partially filled. In this case of "partial goals" the investor may ideally want the full goal, like a luxury trip to Paris for a month, but may also be open to partial goals like only having two weeks in Paris or having a month in New Jersey instead. Each partial goal has a reduced cost from the full goal, but also has a reduced utility if it is chosen. Dynamic Programming can also be used with concurrent or partial goals (see Das et al. (2022) or Capponi and Zhang (2023), for example), so we can continue to compare the RL inference results to DP, as we did in Subsections 4.4 and 4.3.

We consider a test suite of 4 examples here, based off the cases given in Das et al. (2022). The first case (CP1) is identical to the case used in Subsection 4.3.1 of Das et al. (2022). In this case there are three concurrent goals at time 5: one all-or-nothing goal (so two possibilities since the either take or not take the goal), one goal with one partial goal (so three possibilities: take the full goal, take the partial goal, do not take the goal), and one goal with three partial goals (so five possibilities) and there is one goal at time 10 that has one partial goal. We note that this gives $2 \times 3 \times 5 \times 3 = 90$ different

goal-taking combinations to choose from over time. The second case (CP2) is identical to the case used in Subsection 4.5 of Das et al. (2022). This is a far more complicated scenario involving over 301 full goals and 138 partial goals to choose from over the course of $T = 60$ years. The third case (CP3) only uses the car goals from the previous example. These occur every five years, and each allows for a fancy car (full goal) or less fancy car (partial goal) to be considered. The final case (CP4) is a more complicated variation of CP3. It also allows for a fancy car (at a cost of 32 (thousand) dollars for a utility of 125) or a less fancy car (at a cost of 22 (thousand) dollars for a utility of 110) to be purchased every five years (i.e., at $t = 5, 10, ..., 55$ years) but it also considers purchasing a trip every ten years (i.e., at $t = 10, 20, ..., 50$ years) at a cost of $\left(\frac{t}{2} + 10\right)$ thousand dollars for a utility of 100 each time. For CP4, we start with 25 (thousand) dollars and add (infuse) 5 (thousand) dollars every year.

Table 5 shows the results for each of these four cases. The first three columns of the table look at the computational time. While these examples with concurrent and partial goals generally take more time than the all-or-nothing goals, which is no surprise, the more interesting information is in the time ratio, where we see that as the test suite example gets more complicated, RL inference takes progressively less time to produce a solution compared to DP. Further, from the RL-Efficiency column, we see that the additional complexity of the example does not necessarily mean a degradation in the quality of the solution, noting that the worst RL-Efficiency, which is for case CP4, is the second least complex case.

Table 5: Speed (measured in seconds) and accuracy (measured by the RL-Efficiency) for RL inference versus DP for each of the four cases with concurrent and/or partial goals in our new test suite

Test suite case	DP time (in sec)	RL inference time (in sec)	DP/RL time ratio	RL-Efficiency
CP1	0.276	0.271	1.02	0.990
CP2	14.6	4.50	3.24	0.985
CP3	2.94	1.48	1.98	0.986
CP4	3.36	1.54	2.18	0.972

Given that CP4 has the lowest RL-Efficiency, we choose to look at the heatmaps for CP4's RL inference decisions in relation to DP's, since a lower RL-Efficiency indicates the potential for a larger difference between the DP and RL inference heatmaps. Figure 4 compares these heatmap results. Figure 4's format largely parallels the format in Figures 2 and 3, however since the goal-taking each year is no longer all-or-nothing, we introduce colors in the second row of graphs to indicate the fraction of the average utility that is attained over the utility attained if all the full goals were fulfilled that year. We also have added the bottom row to indicate this same fraction for each goal individually. For example, in the DP graph in the bottom row, the colors indicate that in year 20, the expected utility from the car goal is 50-60% of the maximum possible utility attained if the full car goal were fulfilled, and this number jumps up to 70-80% for the trip goal. Figure 4 largely shows that the goal-taking decisions between DP and RL inference are quite similar, while the investment portfolio decisions have the same qualitative behavior but differ in that RL inference tends a bit towards more aggressive investment portfolios than DP.

5.2 Incorporating Stochastic Inflation

Due to what is the commonly referred to as the "curse of dimensionality," Dynamic Programming, from a computational point of view, is generally limited to 2 or 3 state variables. For any GBWM model that also addresses the effect of stochastic inflation, we have our two previous state variables, the current wealth and the current time, along with two new state variables, the current inflation and, to update real infusions and the costs of real goals, the cumulative effect of inflation up to the current time must also be taken into account. That is, we have 4 state variables, meaning that Dynamic Programming is not a computationally feasible method if we wish to incorporate stochastic inflation.

MetaRL, on the other hand, easily extends to incorporate stochastic inflation models. As an example, we consider the Vasicek (1977) model for inflation, which is governed by the stochastic differential equation

$$dI_{\text{infl}} = -\kappa_{\text{infl}}(I_{\text{infl}} - \theta_{\text{infl}})dt + \sigma_{\text{infl}}dW, \tag{3}$$

where W is a Wiener process. This model is a mean reverting process, where θ_{infl} is the mean value of I_{infl}, κ_{infl} is the constant strength of the reversion to this mean value, and σ_{infl} is a volatility

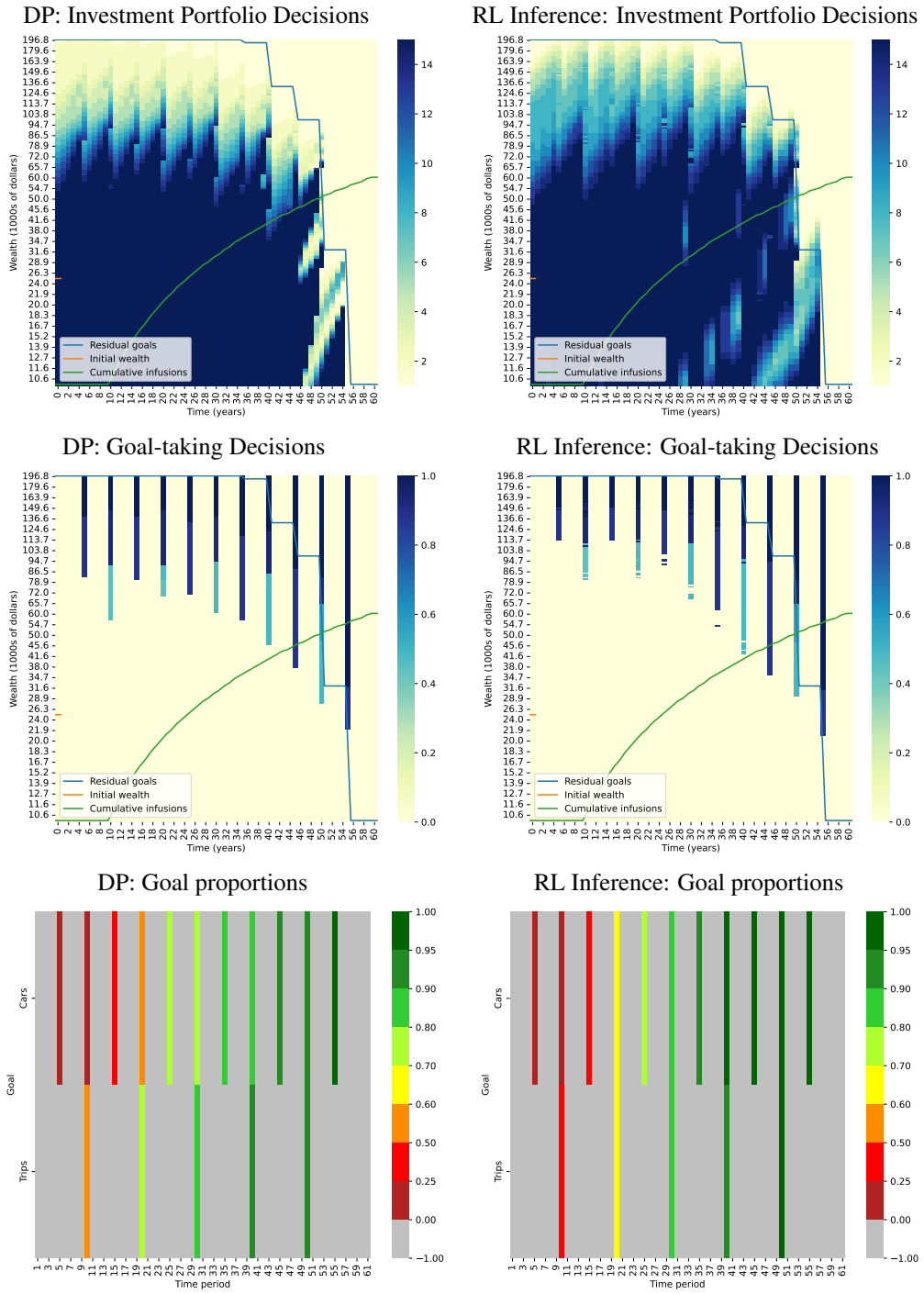


Figure 4: Heatmaps for investment portfolio decisions and goal-taking decisions with case CP4, which has both concurrent goals (cars and trips) and partial goals (for both cars and trips). The specifics of case CP4 are in the text. The colors in the second row (Goal-taking Decisions) indicate the ratio of the expected attained utility from that year's goals over the utility attained if all of that year's full goals were fulfilled. The graphs in the bottom row correspond to this same ratio, but for each individual goal (i.e., for cars and for trips).

constant representing the strength of the randomness. The method for incorporating the current inflation and the cumulative effect of the inflation is contained in Appendix E, which we note uses 27

Table 6: The effect of inflation as we vary the three inflation parameters, θ_{infl}, κ_{infl}, and σ_{infl}, in the Vasicek model. The table shows the fraction of expected attained utility in the presences of stochastic inflation over the expected attained utility when there is no inflation. In all cases, the initial inflation rate is set equal to the mean, θ_{infl}.

Panel A: $\kappa_{\text{infl}} = 0.1$

	σ_{infl}				
θ_{infl}	0.00	0.01	0.02	0.03	0.06
0.01	0.943	0.940	0.931	0.907	0.804
0.02	0.883	0.878	0.876	0.869	0.783
0.03	0.821	0.817	0.818	0.823	0.764
0.05	0.701	0.706	0.713	0.727	0.724
0.08	0.601	0.609	0.615	0.623	0.666
0.10	0.601	0.599	0.595	0.598	0.632

Panel B: $\kappa_{\text{infl}} = 0.6$

	σ_{infl}				
θ_{infl}	0.00	0.01	0.02	0.03	0.06
0.01	0.943	0.943	0.942	0.942	0.938
0.02	0.883	0.883	0.882	0.881	0.877
0.03	0.821	0.821	0.820	0.819	0.817
0.05	0.701	0.702	0.703	0.704	0.708
0.08	0.601	0.600	0.601	0.606	0.610
0.10	0.601	0.600	0.600	0.600	0.595

state variables. Experiments show the changes given in Appendix E have *no* discernible impact on runtimes, either for training the MetaRL model or for running inferences with it.

Table 6 shows the effect that stochastic inflation has on investors' utility when averaged across the 66 test problems used in Section 4.1. More specifically, the table contains the average optimized utility over all 66 test cases (each using 10,000 Monte Carlo paths) in the presence of stochastic inflation divided by 481.54, which is the average optimized utility with the same cases and paths if there is no inflation. In all cases, the initial inflation is set equal to the mean inflation, θ_{infl}.

One effect that is clear from the tables is that the higher the inflation, the lower the expected attained utility. This is simply a ramification of the fact that the higher the inflation, the higher the cost of the goals becomes (since they increase with inflation), so fewer of goals are financially attainable.

The more important question is what is the effect of the inflation being stochastic, as opposed to constant? If the inflation can be assumed to be constant (i.e., equal to θ_{infl}), then the goals' costs can be adjusted for inflation at time $= 0$, and we can revert back to our previous non-stochastic inflation analysis if desired. Our metaRL approach displayed in Table 6 shows that when θ_{infl}, the strength of the mean reversion, is a weak value like 0.1 in Panel A, the effect of the inflation being stochastic (seen by looking at the variation over a row in the table) can be significant. However, when the strength of the mean reversion is a stronger value like 0.6 in Panel B, there is generally little variation across the rows and it is reasonable to treat the inflation as constant.

Looking at historical inflation in the United States for example, we can approximate $\theta_{\text{infl}} = 0.03$, $\kappa_{\text{infl}} = 0.6$, and $\sigma_{\text{infl}} = 0.03$. From Panel B, we see that using the difference between using the average inflation (i.e., $\sigma_{\text{infl}} = 0$) versus $\sigma_{\text{infl}} = 0.03$ when $\theta_{\text{infl}} = 0.03$ and $\kappa_{\text{infl}} = 0.6$ is trivial. In fact, choosing $\theta_{\text{infl}} = 0.031$ makes about three times more of a difference to the expected accumulated utility than including the effect of inflation being stochastic. Therefore, our metaRL analysis shows that using dynamic programming with the average rate of inflation instead of stochastic inflation is justified for current calculations in America—a conclusion that could not be found through dynamic programming—but that may not be the case for other countries or different circumstances. In each new $(\theta_{\text{infl}}, \kappa_{\text{infl}}, \sigma_{\text{infl}})$ scenario, however, our metaRL analysis can be used to determine if using the average inflation is justified or, if not, how to accommodate stochastic inflation.

6 Concluding Comments

This paper looks to develop an innovative approach that combines advanced computational techniques with a comprehensive financial planning framework to significantly enhance the effectiveness and accuracy of wealth management strategies. We developed a meta-model based on reinforcement learning (RL), known as MetaRL, that is pre-trained on thousands of goals-based wealth management (GBWM) scenarios. This model provides rapid, near-optimal solutions for new investor scenarios without the need for any re-training, delivering expected utilities that closely approximate those obtained through dynamic programming. More specifically, the "RL-Efficiency" (i.e., the ratio of the estimated optimal expected attained utility from using inference with MetaRL to the same result if we use the optimum decisions, which are obtained from dynamic programming (DP)) is shown to be, on average, 97.8% across a suite of test scenarios that have a wide range in initial wealths, investor horizons (up to 100 years), infusions, number of goals (from only one goal at the horizon to goals every year) and goal properties (timing, costs, and utilities).

Rather than training a new model for each new investor's retirement planning scenario, the MetaRL approach invests in a one-time pre-training effort to create a model that can then be called directly for a new investor scenario, thereby offering essentially instant inference to determine near-optimal goal-taking strategies and investment portfolio decisions. Thus, MetaRL leverages ideas from the pre-training of foundation models such as LLMs. MetaRL can be deployed on thin clients for inference. Its parameters and weights can be released without revealing the code for the training procedure and the data and problems on which the MetaRL model has been trained, mirroring the release of open models in the realm of LLMs.

MetaRL efficiently addresses the complexities inherent in multi-goals wealth management. By generating policy decisions within hundredths of seconds that are optimized both for dynamic goal-taking decisions and for dynamic investment portfolio selection, MetaRL offers a substantial computational advantage over Monte Carlo methodologies, which must be based on static, not dynamic, decisions, or over traditional DP. Furthermore, the MetaRL model's capacity to handle larger state spaces where dynamic programming becomes computationally infeasible underscores its robustness and scalability. MetaRL's state space in this paper's implementation has 26 or 27 dimensions, which would absolutely be impossible to accommodate with DP. This RL approach provides fast, personalized wealth management strategies that are, on average, over 100 times faster than DP when implemented in inference mode for financial advising.

The paper also shows that it is easy to extend the MetaRL methodology to GBWM problems that contain concurrent and partial goals. An additional extension is also given where inflation is made stochastic, creating a 4 dimensional state space for DP that it cannot reasonably address but a 27 dimensional state space that inference with MetaRL can. Inflation increases can sharply diminish investor outcomes as goal costs rise with inflation and goals become less affordable. From our MetaRL analysis, we find that inflation volatility matters and diminishes investor utility, though only when mean reversion in inflation is low. Luckily, inflation mean reversion rates in the U.S. are high, mitigating the deleterious effect of inflation volatility.

Future work will explore other extensions like this to the GBWM model, which were not feasible with DP, but now become explorable with MetaRL. Some of these are: enhancing GBWM with tax planning, regime-switching in the environment, multiple accounts (taxable and tax-free), goal postponement (Bae et al., 2024), pension planning, etc. We believe it is also possible to handle GBWM alongside traditional wealth and utility maximization paradigms in a single unified framework using RL.

Appendices

A RL Algorithm For At Most One All-Or-Nothing Goal In Each Time Step

The technical details of the RL algorithm are presented here, i.e., the RL environment, training and inference, and extensions. While we restrict ourselves in this appendix to at most one goal in each time step, which the investor either completely fulfills or completely forgoes, we will extend our model to encompass the possibility of two or more goals in a time step (i.e., concurrent goals) and/or partial goals (for example, where an investor is allowed to choose some less nice vacations with reduced attained utilities but also reduced costs) in Appendix D.

A.1 The Meta-model's Environment

We refer to our model as a "meta-model" (Beck et al., 2023) or, more specifically, our "MetaRL" model to distinguish it from RL approaches that are trained to solve a single task. To create our MetaRL model, we use pre-trained RL models to handle scenarios that are arbitrary in terms of (1) the time horizon, (2) the initial wealth and the times and amounts of any subsequent wealth infusions, and (3) the number of goals (which may include partial goals), and, for each goal, the time it can be purchased, the cost to purchase it at that time, and the utility (representing the goal's importance to the investor) gained by the investor if the goal is purchased. We may think of this approach as being similar to training for zero-shot learning and inference (Pourpanah et al., 2022).

A.2 Actions And The Environment Over A Time Step

We describe the environment for a single time step between time t and time $t+1$. We define time t^- to be at time t, just before the goal-taking choice is made, and time t^+ to be at time t, but just after the goal-taking choice is made. At time t^-, the goal-taking agent (GoalAgent) produces an action $a_g(t) \in [0,1]$. If its value is below a threshold value a_{thresh} or if there is not sufficient money (that is, $W(t^-) < C(t)$), the goal is not taken, in which case the wealth is unchanged ($W(t^+) = W(t^-)$) and there is no attained utility ($U_{\text{attained}}(t) = 0$). On the other hand if $a_g(t) \geq a_{\text{thresh}}$ and $W(t^-) \geq C(t)$, then the goal is taken, in which case, we subtract the cost of the goal from the wealth and set $U_{\text{attained}}(t) = U(t)$. That is, in either the case of not taking the goal or taking the goal, if we define the decision $g(t)$ by

$$g(t) = \mathbb{1}_{a_g(t) \geq a_{\text{thresh}}} \cdot \mathbb{1}_{W(t^-) \geq C(t)}, \tag{4}$$

we have

$$W(t^+) = W(t^-) - g(t) \cdot C(t) \tag{5}$$

$$U_{\text{attained}}(t) = g(t) \cdot U(t). \tag{6}$$

(We note that we set $a_{\text{thresh}} = 0.5$ for the experiments in this paper, since 0.5 is the midpoint of $[0,1]$, the range for $a_g(t)$.)

The resulting wealth, $W(t^+)$, is used to compute the state input for the investment portfolio agent (PortfolioAgent). PortfolioAgent returns the action $a_p(t) \in [0,1]$, which defines $p(t)$, the investment portfolio decision, by setting $p(t)$ equal to the value of the integer p where

$$\frac{p}{P} \leq a_p(t) < \frac{p+1}{P} \tag{7}$$

(and setting $p = P - 1$ if $a_p(t) = 1$). So, for example, if p increasing corresponds to progressively aggressive portfolios (potentially along the efficient frontier, although that doesn't need to be the case), then the magnitude of $a_p(t)$ corresponds to how aggressive the chosen investment portfolio is. We then assume that the wealth between time t and time $t+1$ will evolve via geometric Brownian motion (GBM) with the chosen portfolio p. That is,

$$W((t+1)^-) = W(t^+) \cdot \exp\left[(\mu_p - \frac{1}{2}\sigma_p^2) \cdot h + \sigma_p \cdot Z \cdot \sqrt{h}\right] + I(t+1). \tag{8}$$

Note that our model here assumes that the timing and magnitude of the infusions, $I(t)$, can either be predicted or known in advance. Of course, unlike the initial wealth, which can be applied to any goal, an infusion at time t can only be applied to goals available at time t or later.

The objective of both the agents combined is to maximize $E\left[\sum_{\tau=t}^{T} U_{\text{attained}}(\tau)\right]$, the expected value of the utility accrued over the remaining time horizon of the problem. Since the evolution equations (5), (6), and (8), as well as the optimal objective, are dependent only on the time, t, the current wealth, $W(t)$, and the action choices, we conclude that the problem is Markovian, and hence suitable for a RL formulation.

A.3 States

In principle, the information specified in Table 1 is sufficient for computation of $a_g(t)$ and $a_p(t)$. However, we require some manipulation of this information in order to obtain more effective state variables for the MetaRL agents. The objectives of this manipulation are, (i) to define a constant-dimension input for the MetaRL agent regardless of the size of the scenario, and (ii) to provide rich information that helps the MetaRL agent identify patterns during training. This subsection describes these more effective state variables.

We look to have state variables that are unitless and normalized to either only take values between 0 and 1 or have most of their interesting behavior occur between 0 and 1. This makes the gradient ascent algorithm in PPO more effective, since gradient ascent is well known to work poorly with independent variables that have significantly different scales.

The first three state variables are the scalars t_{norm}, $W_{\min}$, and $W_{\max}$, which are unitless, normalized versions of the time, t, and wealth, $W(t)$. The definition of t_{norm} is simply

$$t_{\text{norm}} = \frac{t}{T}. \tag{9}$$

The important aspect about the wealth at time t is its ability to obtain present (time $= t$) and future (time $> t$) goals, so we look to normalize the wealth in relation to the sum of the costs of all present and future goals. To do this, we must approximate how to discount the costs of future goals to time t dollars, which $W(t)$ is obviously expressed in. Equation (8), the GBM equation, gives the expression for discounting, which depends on the future portfolios, p, chosen, which can shift over time, and the future stochastic behavior, given by Z. The algorithm $\textsc{DiscountSum}(\mathbf{C}[t:], p, z)$, given in Algorithm 1, uses equation (8) to compute the discounted sum of present and future goal costs, assuming a given, fixed portfolio, p, for all times $\geq t$ and a given, fixed value z in place of Z for all times $\geq t$.

For the purposes of normalizing the wealth, we will approximate discounting the future goal costs in two different ways by using $\textsc{DiscountSum}(\mathbf{C}[t:], p, z)$ with two different (p, z) pairs. For the first pair, we set p to be 0 (the most conservative investment portfolio) and $z = -1$ (a pessimistic scenario). For the second pair, we set $p = P - 1$ (the most aggressive investment portfolio) and $z = 1$ (an optimistic scenario). This gives us $W_{\min}$ and $W_{\max}$, our two state variables for the normalized wealth:

$$W_{\min} = \frac{W(t)}{\textsc{DiscountSum}(\mathbf{C}[t:], p = 0, z = -1)} \tag{10}$$

$$W_{\max} = \frac{W(t)}{\textsc{DiscountSum}(\mathbf{C}[t:], p = P - 1, z = 1)}. \tag{11}$$

The next three state variables are the vectors $\mathbf{U}_{\text{agg}}$, $\mathbf{C}_{\min}$, and $\mathbf{C}_{\max}$, which describe the relative timing of the utilities and costs of goals at times $\geq t$. It matters if we are looking to attain a goal that occurs one year from now versus, say, 23 years from now. These state variable vectors look to contain the key aspects of this timing information, while retaining a key property for RL computation of having a fixed length, regardless of the time t at which they are computed or the time horizon T.

We begin with the utilities. At first, one might simply think to use the vector $\mathbf{U}[t:]$, which contains the utility of each present (time t) and future goal, and then divide this vector by its sum to normalize its components to be between 0 and 1. However, because the length of $\mathbf{U}[t:]$ is $T - t + 1$, which varies depending on t and T, and we need our state variable utility vector to have the same length regardless of t and T, we instead look to aggregate $\mathbf{U}[t:]$ into a fixed number, K, of time blocks. As an example that we will use in all of our experiments, we consider the following list $\mathbf{L}$ that contains $K = 7$ aggregated time blocks: $\mathbf{L} = [[0], [1], [2], [3], [4, 5], [6, 7, 8, 9], [10:]]$. The penultimate time

Algorithm 1 Discounting future goal costs and returning each of them (DISCOUNTVEC) or their sum (DISCOUNTSUM).

procedure DISCOUNTVEC($\mathbf{C}[t:], p, z$)
  $\mathbf{C}_{\text{disc}} \leftarrow$ initialize vector of zeros of same length as $\mathbf{C}[t:]$, which is $T - t + 1$
  for τ in [0 : length($\mathbf{C}[t:]$) **do**
    $C_{\text{disc}}(\tau) = C(\tau) \exp\left[-(\mu_p - \frac{1}{2}\sigma_p^2)h\tau - \sigma_p z\sqrt{h\tau}\right]$
  end for
  return $\mathbf{C}_{\text{disc}}$
end procedure

procedure DISCOUNTSUM($\mathbf{C}[t:], p, z$)
  return sum$\big($DISCOUNTVEC$(\mathbf{C}[t:], p, z)\big)$
end procedure

block for this $\mathbf{L}$ specifies aggregating all the utilities from goals occurring at times $t + 6$ through $t + 9$, while the final time block specifies aggregating all the utilities from goals occurring at time $t + 10$ or later.

To define $\mathbf{U}_{\text{agg}}$, our state variable utility vector, we apply the aggregation algorithm AGGREGATE$(\mathbf{V}, \mathbf{L})$, defined in Algorithm 2, to the vector $\mathbf{V} = \mathbf{U}[t:]$ and then divide by the sum of the components in $\mathbf{U}[t:]$:

$$\mathbf{U}_{\text{agg}}[t:] = \frac{\text{AGGREGATE}(\mathbf{U}[t:], \mathbf{L})}{\sum_{\tau=t}^{T} U(\tau)}. \tag{12}$$

Note that AGGREGATE$(\mathbf{U}[t:], \mathbf{L})$ is a vector of length $K = 7$ where, for example, the final component contains the sum of the utilities from the goals that occur at time $t + 10$ or later.

Our approach to the state variables for the goals' costs is parallel to our approach for $\mathbf{U}_{\text{agg}}$, the goals' utilities, but since we're looking to compare relative costs, we need to discount the costs to time t dollars for a fair comparison. To do this we use the same approach to discounting (along with the same two (p, z) pairs) that we used to define $W_{\min}$ and $W_{\max}$ to obtain

$$\mathbf{C}_{\min}[t:] = \frac{\text{AGGREGATE}\big(\text{DISCOUNTVEC}(\mathbf{C}[t:], p = P - 1, z = 1), \mathbf{L}\big)}{\text{DISCOUNTSUM}(\mathbf{C}[t:], p = P - 1, z = 1)} \tag{13}$$

$$\mathbf{C}_{\max}[t:] = \frac{\text{AGGREGATE}\big(\text{DISCOUNTVEC}(\mathbf{C}[t:], p = 0, z = -1), \mathbf{L}\big)}{\text{DISCOUNTSUM}(\mathbf{C}[t:], p = 0, z = -1)}, \tag{14}$$

where DISCOUNTVEC$(\mathbf{C}[t:], p, z)$ is defined in Algorithm 1. We note that because the discounted sum is in the denominator when defining $W_{\min}$ and $W_{\max}$, while the key discounting is in the numerator when defining $\mathbf{C}_{\min}$ and $\mathbf{C}_{\max}$, the case where $(p, z) = (0, -1)$ corresponds to $W_{\min}$ and $C_{\max}$, while the case where $(p, z) = (P - 1, 1)$ corresponds to $W_{\max}$ and $C_{\min}$.

Algorithm 2 Aggregating $\mathbf{V}$, which can represent a vector of goal utilities or a vector of goal costs, over the K aggregated present and future time blocks given in $\mathbf{L}$.

procedure AGGREGATE(Vector $\mathbf{V}$, List of lists $\mathbf{L}$)
  $K \leftarrow$ length of $\mathbf{L}$
  $\mathbf{A} \leftarrow$ vector of zeros of length K
  for ℓ in [0 : K] **do**
    currentlist $\leftarrow \mathbf{L}(\ell)$
    for i in [0 : length(**currentlist**)] **do**
      $A(\ell) \leftarrow A(\ell) + V(\textbf{currentlist}(i))$
    end for
  end for
  return $\mathbf{A}$
end procedure

The final two state variables are the scalars g_{sim} and p_{sim}, which, as we will demonstrate later, are the most useful of our state variables for optimizing results, with g_{sim} aimed to be particularly useful

for the GoalAgent and p_{sim} for the PortfolioAgent. These two state variables provide the MetaRL agents with richer information about the dynamics of the environment by running a handful of deterministic forward simulations, using some considerable approximations for the future dynamics. The procedure is inspired by literature on Neural Monte-Carlo Tree Search (Silver et al., 2016), but in simplified form.

To compute either g_{sim} or p_{sim}, we start by selecting a positive integer n and then determining a list of the n values of Z that are the midpoints of a partition of the cumulative standard normal distribution into n equal parts. That is, the list's components, z_i where $i = 0, 1, ..., n-1$, are defined so that $P(Z < z_i) = \frac{i+\frac{1}{2}}{n}$. Since we find that $n = 11$ is sufficiently large for our results, we use this in all our experiments.

We then calculate DISCOUNTVEC($\mathbf{C}[t:], p, z_i$) and DISCOUNTVEC($\mathbf{I}[t:], p, z_i$) for each investment portfolio p and each z_i. This gives the goal costs and infusions in time $= t$ dollars, given the investment portfolio is fixed to be p and $Z = z_i$ in equation (8) for all times $\geq t$. For each (p, z_i) pair, we can then take the wealth, $W(t^-)$, and the discounted infusions, which are all in time t dollars, and use them to purchase goals in order of decreasing utility (using chronological order for goals with the same utility). In purchasing these goals, we pay for the discounted goal prices DISCOUNTVEC($\mathbf{C}[t:], p, z_i$) by first applying any available discounted infusions DISCOUNTVEC($\mathbf{I}[t:], p, z_i$) (using reverse chronological order, meaning using the most recently available infusion that can be applied to the goal) and then, if all the available infusions are drained, applying $W(t^-)$. We stop when the discounted infusions and $W(t^-)$ remaining are too small to purchase any more goals or all the goals have been purchased. This gives the accumulated utility as a function of p and z_i. For each fixed p, we then take the average of the accumulated utilities over the n values of z_i to give a loose approximation of the expected accumulated utility sums for each investment portfolio p.

To determine g_{sim}, we use the method just described to determine the approximated expected utility sums (for each value of p) for two different sets of goals: one where we first take the current (time $= t$) goal, if it is possible, before taking the future goals in descending order of their utilities as described above, the other where we remove the current goal completely before taking the future goals, again as described above. Since we are looking to optimize these two approximated expected utility sums, we only retain the largest approximated expected utility sum over the P values of p.[3] This approximates using the best (fixed) portfolio choice, which may, of course, correspond to different values of p for the two approximated expected utility sums. We then subtract the second optimized approximated expected utility sum (which skips the time t goal) from the first sum (which gives the time t goal first priority), then divide this difference by the first sum to normalize it, and finally define g_{sim} to be the logistic function of this normalized difference, which keeps $g_{\text{sim}} \in [0, 1]$. Note that the larger g_{sim} is, the more evidence there is to take the goal at time t, which is exactly what GoalAgent is looking to decide. This process that we have just described for computing g_{sim} is summarized in Algorithm 3. Given the nature of g_{sim}, we may consider it akin to a binary classifier, choosing between one of two goal choices.

To determine p_{sim}, we use a similar process. As before, we compute the approximated expected utility sums for each p, but we no longer give first preference to the goal at time t, nor do we remove it. We then find the p that gives the highest approximated expected utility sum, and normalize by dividing it by $P - 1$ to determine the value for $p_{\text{sim}} \in [0, 1]$. If the investment portfolios get more aggressive as p increases, as we will have in our experiments where the investment portfolios are chosen over an efficient frontier, the higher p_{sim} is, the more evidence to choose a more aggressive investment portfolio. This process for computing p_{sim} is summarized in in Algorithm 4. Unlike the binary classifier nature of g_{sim}, the value of p_{sim} is used as a regressor that chooses one portfolio from an ordered set of portfolios.

We summarize our list of state variables in Table 7.

[3] We note that this is a significant approximation because we have fixed p for all current and future times, when our final results will allow the investment portfolio p to vary with time. Similarly, our expected accumulated utility sums only used fixed values $Z = z_i$ as a significant approximation for potentially complex Brownian motion. These approximations work well in practice, as we will see, and they can be computed deterministically and quickly, which is key for our state variable calculations.

Algorithm 3 Forward simulation to compute g_{sim}

procedure GSIM($W(t^-)$, $\mathbf{C}[t:]$, $\mathbf{U}[t:]$, $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, n)
  listZ ← the n midpoint Z values after partitioning Z's CDF into n equal sections
  sortU ← components of $\mathbf{U}[t:]$ in descending order (with ties in chronological order)
  Force the goal at time $= t$ to be the first entry in **sortU**
  $\mathbf{E}_{\text{take}}$ ← vector of zeros of length P
  for p in $[0, \ldots, P-1]$ **do**
    $\mathbf{C}_{\text{disc}}, \mathbf{I}_{\text{disc}}$, and $\mathbf{U}_{\text{tot}}$ ← vectors of zeros, each of length n
    for z in **listZ do**
      $\mathbf{C}_{\text{disc}}$ ← DISCOUNTVEC(**C**[t:],p,z)
      $\mathbf{I}_{\text{disc}}$ ← DISCOUNTVEC(**I**[t:],p,z)
      $\mathbf{sortC}$ ← $\mathbf{C}_{\text{disc}}$ reordered to be in the same goal order as **sortU**
      $\mathbf{U}_{\text{tot}}[z]$ ← accrued utility by spending $\mathbf{I}_{\text{disc}}$ and $W(t^-)$ in the order of goals in $\mathbf{sortU}$, paying amounts specified in $\mathbf{sortC}$, until bankrupt or attaining all goals
    end for
    $\mathbf{E}_{\text{take}}[p]$ ← average of $\mathbf{U}_{\text{tot}}$ over its n components
  end for
  Remove first goal from **sortU**
  $\mathbf{E}_{\text{skip}}$ ← vector of zeros of length P
  for p in $[0, \ldots, P-1]$ **do**
    $\mathbf{C}_{\text{disc}}, \mathbf{I}_{\text{disc}}$, and $\mathbf{U}_{\text{tot}}$ ← vectors of zeros, each of length n
    for z in **listZ do**
      $\mathbf{C}_{\text{disc}}$ ← DISCOUNTVEC(**C**[t:],p,z)
      $\mathbf{I}_{\text{disc}}$ ← DISCOUNTVEC(**I**[t:],p,z)
      $\mathbf{sortC}$ ← $\mathbf{C}_{\text{disc}}$ reordered to be in the same goal order as **sortU**
      $\mathbf{U}_{\text{tot}}[z]$ ← accrued utility by spending $\mathbf{I}_{\text{disc}}$ and $W(t^-)$ in the order of goals in $\mathbf{sortU}$, paying amounts specified in $\mathbf{sortC}$, until bankrupt or attaining all goals
    end for
    $\mathbf{E}_{\text{skip}}[p]$ ← average of $\mathbf{U}_{\text{tot}}$ over its n components
  end for
  $\max_{\text{take}} \leftarrow \max(\mathbf{E}_{\text{take}})$ and $\max_{\text{skip}} \leftarrow \max(\mathbf{E}_{\text{skip}})$
  return logistic$((\max_{\text{take}} - \max_{\text{skip}})/\max_{\text{take}})$
end procedure

Algorithm 4 Forward simulation to compute p_{sim}

procedure PSIM($W(t^-)$, $\mathbf{C}[t:]$, $\mathbf{U}[t:]$, $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, n)
  listZ ← the n midpoint Z values after partitioning Z's CDF into n equal sections
  sortU ← components of $\mathbf{U}[t:]$ in descending order (with ties in chronological order)
  $\mathbf{E}$ ← vector of zeros of length P
  for p in $[0, \ldots, P-1]$ **do**
    $\mathbf{C}_{\text{disc}}, \mathbf{I}_{\text{disc}}$, and $\mathbf{U}_{\text{tot}}$ ← vectors of zeros, each of length n
    for z in **listZ do**
      $\mathbf{C}_{\text{disc}}$ ← DISCOUNTVEC(**C**[t:],p,z)
      $\mathbf{I}_{\text{disc}}$ ← DISCOUNTVEC(**I**[t:],p,z)
      $\mathbf{sortC}$ ← $\mathbf{C}_{\text{disc}}$ reordered to be in the same goal order as **sortU**
      $\mathbf{U}_{\text{tot}}[z]$ ← accrued utility by spending $\mathbf{I}_{\text{disc}}$ and $W(t^-)$ in the order of goals in $\mathbf{sortU}$, paying amounts specified in $\mathbf{sortC}$, until bankrupt or attaining all goals
    end for
    $\mathbf{E}[p]$ ← average of $\mathbf{U}_{\text{tot}}$ over its n components
  end for
  return $\arg\max(\mathbf{E})/(P-1)$
end procedure

A.4 Rewards

We define two types of rewards associated with each time step for training the two MetaRL agents. The first type of time step reward, the extrinsic reward, $r_e(t)$, corresponds to the underlying objective function of the problem. We define this at all time steps, except the final time step at T, by

$$r_e(t) = \frac{g(t) \cdot U(t)}{\sum_{\tau=0}^{T} U(\tau)} \text{ for } t < T. \tag{15}$$

Table 7: Summary of state variable inputs for the two action producing agents, GoalAgent and PortfolioAgent. For all experiments in this paper, we use a value of $K = 7$ aggregated current and future time blocks, which are given by the list $\mathbf{L} = [[0], [1], [2], [3], [4, 5], [6, 7, 8, 9], [10 :]]$, where numbers represent time steps in the future.

Symbol	Length	Explanation
t_{norm}	1 (float)	Time step normalized by the final time step horizon T
$W_{\min}$	1 (float)	Current wealth normalized by the sum of future goal costs discounted using a pessimistic, conservative scenario
$W_{\max}$	1 (float)	Current wealth normalized by the sum of future goal costs discounted using an optimistic, aggressive scenario
$\mathbf{U}_{\text{agg}}$	K (float)	Vector of utilities aggregated into K time blocks and normalized by their sum
$\mathbf{C}_{\min}$	K (float)	Vector of optimistically, aggressively discounted goal costs aggregated into K time blocks and normalized by their sum
$\mathbf{C}_{\max}$	K (float)	Vector of pessimistically, conservatively discounted goal costs aggregated into K time blocks and normalized by their sum
g_{sim}	1 (float)	Forward-simulation-based evidence for taking the current goal
p_{sim}	1 (float)	Forward-simulation-based evidence for selecting a more aggressive investment portfolio at the current time

Recall from the definition of $g(t)$ in equation (4), this means the reward is accrued if and only if (i) sufficient wealth is available to take the goal, and (ii) the GoalAgent action, $a_g(t)$, is higher than a_{thresh}, where we set $a_{\text{thresh}} = 0.5$ for the experiments in this paper. At the final time step $t = T$, we augment the above definition by adding a small reward to the agent for the amount of money remaining if it is not enough to take the final goal. This signals to the MetaRL algorithm that finishing with more money is better than less money, which creates an otherwise missing push towards attaining the final goal as the MetaRL model evolves over epochs. Specifically, we define the extrinsic reward at the final step T by

$$r_e(T) = \frac{\mathbb{1}_{a_g(T) \geq a_{\text{thresh}}} \cdot U(T)}{\sum_{\tau=0}^{T} U(\tau)} \left[\mathbb{1}_{W(T^-) \geq C(T)} + \mathbb{1}_{W(T^-) < C(T)} \cdot \frac{1}{4} \left(\frac{W(T^-)}{C(T)} \right) \right]. \tag{16}$$

Even with this modification at $t = T$, the extrinsic reward provides a relatively weak signal to the MetaRL agents because (i) the rewards are driven by a number of factors, such as initial wealth and the stochasticity in GBM, that are disconnected from the MetaRL agents' decisions, (ii) the rewards are often delayed a number of time steps after MetaRL agent decisions are made, and (iii) the dual agent architecture makes it difficult to accomplish credit assignment (Zhou et al., 2020). We therefore provide a second reward, called the intrinsic reward, at each time step for each of the two MetaRL agents, following literature on intrinsic motivation in reinforcement learning (Barto, 2013). The intrinsic rewards are penalties (that is, negative rewards) for actions being far away from their corresponding forward simulated state variables.

More specifically, for GoalAgent and PortfolioAgent, the intrinsic rewards are respectively given by

$$r_{i,g}(t) = -\frac{1}{2}\rho \, |g_{\text{sim}} - a_g(t)| \text{ and} \tag{17}$$

$$r_{i,p}(t) = -\frac{1}{2}\rho \, |p_{\text{sim}} - a_p(t)|, \tag{18}$$

where the coefficient ρ is a scalar multiple that we anneal from 1 to 0.25 over the course of the MetaRL training to emphasize the extrinsic reward more by the end of the training. With $r_e(t), r_{i,g}(t)$, and $r_{i,p}(t)$ defined, we define $R_g(t)$ and $R_p(t)$, the total rewards used at time t for respectively

training GoalAgent and PortfolioAgent, by[4]

$$R_g(t) = \sum_{\tau=t}^{T} \big(r_e(\tau) + r_{i,g}(\tau)\big) \text{ and}$$

$$R_p(t) = \sum_{\tau=t+1}^{T} r_e(\tau) + \sum_{\tau=t}^{T} r_{i,p}(\tau),$$

noting that the value of $r_e(t)$, the extrinsic reward at time t, is excluded from the definition of $R_p(t)$ because PortfolioAgent is called *after* the goal-taking decision has been executed at time t.

This completes the description of the formulation of the MetaRL environment. We next discuss implementation details of this formulation.

B MetaRL Model Training And Inference

B.1 Network Architecture And Hyperparameters

RL formulates the decision-making at each time step to consist simultaneously of the goal-taking choice and the investment portfolio selection. We note that the goal-taking choice at the beginning of any time step directly affects the subsequent investment portfolio selection during that time step, because taking a goal affects the available remaining wealth, which affects the optimal investment portfolio choice. Therefore, the RL methodology depicted in Figure 5 splits the decision into two steps for goal-taking and investment portfolio selection (shown in the "Agents" section of the Figure).

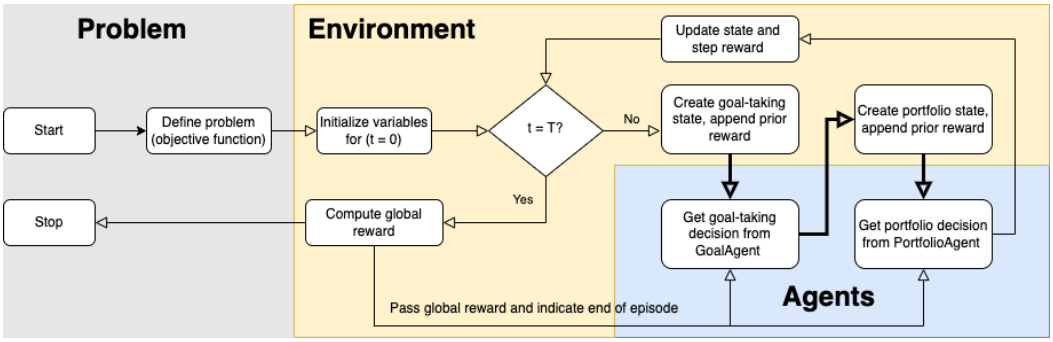


Figure 5: Logical flow of the RL methodology for a single "episode," during which we have time steps $t = 0, 1, ..., T$. Typically, RL runs through a large number of such episodes during training in order to generalize learning to arbitrary problems. The same logical flow is followed during inference on any new problem.

Because the Proximal Policy Optimization (PPO) algorithm (see Schulman et al. (2015, 2017); Haarnoja et al. (2018)) is based on the actor-critic architecture, it contains two neural networks (actor and critic) for GoalAgent and also for PortfolioAgent. The dimension of the input into the actor and critic networks is the dimension of the state space, which, from Table 7, is $5 + 3K = 26$, when $K = 7$, as we assume in our experiments. While GoalAgent and PortfolioAgent are trained as if they are independent PPO agents, the logical flow is similar to Hierarchical RL (Barto and Mahadevan, 2003).

The output for both actors and both critics are scalars, corresponding to a_g and a_p for the two actors, and RL value functions for the two critics. The three intermediate layers for the actor networks have 256, then 64, and finally 16 nodes, while the two for the critic networks have 64 and then 16 nodes. These numbers of nodes and layers, as well as the other hyperparameter values given in the next paragraph, were determined on the basis of an experimental hyperparameter sweep. We do not use any special initializers for the parameters of the networks.

[4]Note that for the sums over time steps in the definitions of both $R_g(t)$ and $R_p(t)$, we have implicitly used a discount factor of 1 (i.e., no discounting) to reflect the fact that the importance of a goal (i.e., its utility) is not diminished by when it occurs in the future.

Both the actor and critic networks use the `tanh` activation function for their intermediate layers. The critics have a `linear` output neuron to ensure the full range of predictions of returns-to-go (that is, RL value function predictions), while the actors have a `sigmoid` output neuron to ensure the output is in the range $[0, 1]$. For hyperparameter values, we used a learning rate of 10^{-4}, a clip parameter of 0.2, and a discount rate is 1.0 (meaning no discounting, since future utility values are not of lesser importance than present utility values).

We trained the MetaRL model using 1000 epochs, where each epoch runs for 500 episodes. All networks were updated at the end of each epoch. Training the MetaRL model using 1000 epochs took approximately 4 hours.

B.2 Training Curriculum

In order to maximize the generalizability of the agents, we provided the MetaRL algorithm with a diverse set of scenarios during training. These scenarios were generated using the procedure GENERATESCENARIO described in Algorithm 5, which is designed to explore a wide range of time horizons and numbers of goals, along with varying the goals' timing, costs, and utilities. A fresh scenario is generated at the start of every epoch, which is solved by a set of 500 random episodes. The randomness between these 500 episodes has three sources: (1) The values generated for Z, the standard normal random variables that determine the portfolio returns in equation (8), (2) the randomness used to determine exploration for the action variables a_g and a_p, (3) a small randomization in the initial wealth, which is selected from a uniform distribution between 80% and 120% of $W(0)$, the initial wealth for the epoch given by the GENERATESCENARIO procedure. The collected experience is used for training the critic and actor networks at the end of each epoch. Note that the intrinsic reward from (17) and (18) is being annealed over the course of training.

The above process is done in parallel using five different seeds for the randomness.[5] That is, in some sense we are running the MetaRL model five times, but the scenarios of all five are the same in each epoch, only the randomness for the 500 episodes within each epoch is different due to the different seeds. The actions produced during inference are determined by the median action produced by the five models.

Figure 6 was compiled to test whether 1000 epochs is sufficient. It shows that it is more than enough, with a large fraction of the algorithm's learning being completed within the first 100 epochs. The figure uses the "RL-Efficiency" determined from all five models. The RL-Efficiency is defined in Subsection 4.4, except that the denominator here is the DP value function (also discussed in that subsection). The bold black line in the graph represents the average RL-Efficiency from the five models. The gray area corresponds to adding or subtracting a standard deviation of the five RL-Efficiencies from the average RL-Efficiency.

B.3 Implementation Details

The RL algorithm (and the baseline dynamic programming logic) are implemented in `Python 3.11` and primarily executed on an AWS `c7i.24xlarge` instance.[6] The instance provides 96 virtual CPUs and 192 GB of memory and is optimized for CPU compute. (No GPUs were required, though using GPUs may offer further efficiencies.) Computational experiments on other hardware types are included in Section 4. Given the compute-intensive nature of state computation, the procedures GSIM and PSIM are compiled using the `numba 0.60` library for parallelization and faster execution. Neural networks for the PPO algorithm are implemented using `torch 2.1`. The PPO algorithm itself is custom-implemented, based on Schulman et al. (2017). The environment in which the agents operate is custom built as an extension of the `Env` environment in the `gym` package from OpenAI (see `https://github.com/openai/gym`).

B.4 Inference During Testing

With our MetaRL model complete, we use a hand-designed set of 66 scenarios to test it, as described in Section 4 and Appendix C. These scenarios fall both within and outside the training scenarios' distribution. During this testing, we only have access to the pre-trained actor models for GoalAgent

[5] We use seeds 0, 15, 722, 1021, 5069 in this paper for all reported results.

[6] https://aws.amazon.com/ec2/instance-types/

Algorithm 5 Generating a training scenario

procedure GENERATESCENARIO($\boldsymbol{\mu}$, $\boldsymbol{\sigma}$)
  Generate T (uniformly) randomly from $\{5, 6, ..., 50\}$
  $\mathbf{C}, \mathbf{U} \leftarrow$ arrays of zeros of length $T + 1$
  Generate N_G, the number of goals, from the distribution $p(1) = 0.22, p(2) = 0.15, p(3) = 0.12, p(4) = 0.10, p(5) = 0.06, p(6) = 0.05, p(7) = 0.04, p(8) = 0.03, p(9) = 0.02, p(10) = 0.01, p(T) = 0.20$
  if $N_G > T$ **then**
    $N_G \leftarrow T$
  end if
  One of the N_G goal times is T. Randomly choose the other $N_G - 1$ goal times from $\{1, 2, ..., T - 1\}$
  for each non-zero goal time $t \in [1, T]$ **do**
    Draw numbers u_1 and u_2 from a uniform distribution on $[0, 1]$
    $C(t) \leftarrow 100 \cdot u_1 \cdot 1.03^t$
    $U(t) \leftarrow 0.3 \cdot C(t)/1.03^t + 25 \cdot u_2$
  end for
  Choose the initial wealth $W(0)$ from a uniform distribution between DISCOUNTSUM($\mathbf{C}[:], P - 1, 2$) and DISCOUNTSUM($\mathbf{C}[:], 0, -2$), where DISCOUNTSUM is definied in Algorithm 1
  return $\mathbf{C}[1:], \mathbf{U}[1:], W(0)$
end procedure

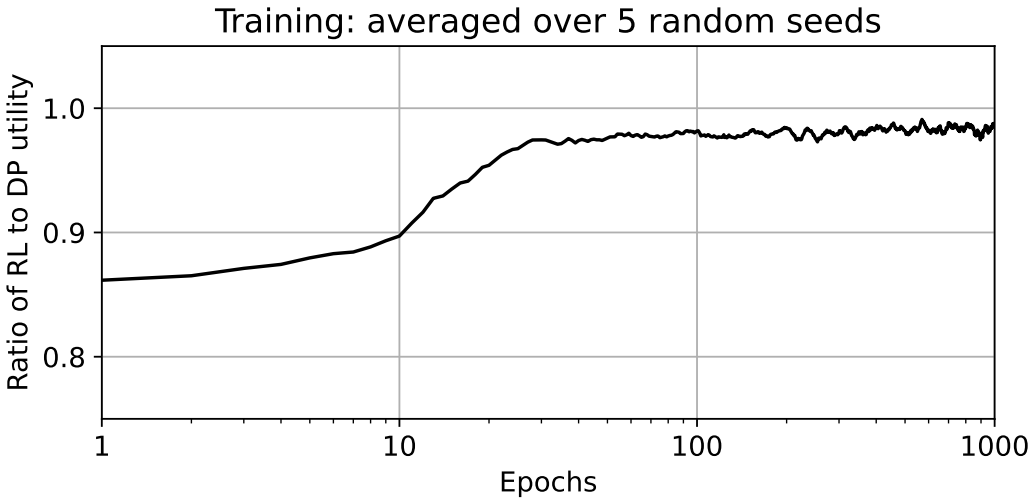


Figure 6: Training efficiency of the RL algorithm over the course of training (using the RL-Efficiency for each of the 1000 epochs' scenarios). While the training episodes each have randomised initial wealth, the quantities plotted here are for the baseline values of initial wealth. The black line represents the average of the five RL-Efficiencies determined using five different seeds for randomness. The shaded region corresponds to being one standard deviation away from the average, where the standard deviation is computed from the five RL-Efficiencies.

and PortfolioAgent from the MetaRL algorithm. The state variable inputs are generated at each time step as described in Section A.3. The actor networks produce goal-taking and investment portfolio selection actions, which are implemented by the environment. As described in Appendix B.2, we mitigate the risk of getting either unusually good or unusually bad results due to random initialization by training 5 models from scratch using identical scenarios in each epoch but different randomness for the 500 episodes within each epoch. The actions reported by RL inference using the MetaRL model are computed by sending the state space values to all 5 models within MetaRL and reporting the median of the 5 actions generated.

C Description Of The 66 Test Suite Cases

The first 33 of the 66 test suite cases are given in Table 8. These cases have no infusions. The other 33 cases are given in Table 9. These are the same as the first 33 cases, except that they have infusions.

D Extending The Model To Concurrent Goals And/Or Partial Goals

The MetaRL model described in Appendix A is designed to handle multiple goals, so long as there is at most one goal per time step. We now describe how to alter this model when we do have multiple competing goals at the same time step and/or allow partial fulfillment of a goal (e.g., buying a less than ideal car, which generates less utility but also costs less than buying the ideal car).

Many of the updates to the concurrent and/or partial goals case are relatively straightforward. For example, for the computation of state inputs $W_{\min}(t)$, $W_{\max}(t)$, $\mathbf{C}_{\min}[t:]$, and $\mathbf{C}_{\max}[t:]$ from Table 7, we just replace $C(t)$, the cost of the single goal at time t, with the highest possible cost at time t, meaning the cost of taking all of the concurrent goals at time t at their full (i.e., not partial) levels. Similarly, to compute $\mathbf{U}_{\text{agg}}[t:]$, we replace $U(t)$, the utility of the single goal at time t, with the highest possible utility at time t from attaining all the concurrent goals at time t at their full levels.

This leaves the computation of g_{sim} and p_{sim}, which is a bit more complicated. In the concurrent/partial goals case, at each time t, there can be a number of possible costs and corresponding utilities corresponding to each possible combination of forgoing, (each available way of) partially fulfilling, or fully fulfilling each of the concurrent goals. We form a pareto front of these possibilities by removing combinations that are dominated by any other combination; for example, we would remove a combination that generates a utility of 50 at a cost of $40,000 if there is another combination that generates a utility of 52 at the same cost. The combinations that remain, we will call $\mathbf{G}(t)$, (for goals) which contains the cost and corresponding utilities of these combinations in increasing order. Note that in this pareto front of combinations, the utility must increase as the cost increases. See Das et al. (2022) for more details.

In the single all-or-nothing goal context of Appendix A, $a_g(t) \in [0,1]$ is a measure of the confidence in taking ($a_g(t) \approx 1$) or not taking ($a_g(t) \approx 0$) the goal at time t. In the current context, $a_g(t)$ is a measure of the confidence in taking the goal combination at time t that corresponds to the highest approximated expected accumulated utility from time t onwards versus the combination at time t that corresponds to the second highest approximated expected accumulated utility from time t onwards. The determination of the which two combinations at time t have the two highest approximated expected accumulated utility from time t onwards comes from our computation for determining g_{sim}.

To compute g_{sim}, we first extend the computation of the P-vectors $\mathbf{E}_{\text{take}}$ and $\mathbf{E}_{\text{skip}}$ in Algorithm 3 to all the combinations in $\mathbf{G}(t)$. This results in a matrix $\mathbf{E}$ of dimension $P \times |\mathbf{G}(t)|$, where $|\mathbf{G}(t)|$ is the number of goal combinations in $\mathbf{G}(t)$. Each column in $\mathbf{E}$ is determined by following Algorithm 3 for computing $\mathbf{E}_{\text{take}}$, but forcing taking the goal combination corresponding to the column to be the first goal taken (i.e., the first entry in $\mathbf{sort}\mathbf{U}$). The highest approximated expected utility for every goal combination is given by taking the maximum of each column of $\mathbf{E}$, and the logistic in GSIM is computed between the highest and second-highest values of the column-maxima. The goal combinations that correspond to these two columns are the two goal combinations that $a_g(t)$ chooses between. Note that our approach here conforms to the single all-or-nothing goals case in Appendix A where $|\mathbf{G}| = 2$.

For computing p_{sim}, we start with the same matrix $\mathbf{E}$ calculated for g_{sim}. However, instead of computing column maxima, we determine the index of the row that contains the largest component in $\mathbf{E}$. This corresponds to the approximating the best portfolio p. As in Algorithm 4, we define p_{sim} to be this index divided by $P-1$ to normalize p_{sim} to be between 0 and 1.

E Extending The Model To Stochastic Inflation

In this section we show how our MetaRL model can quickly be extended when inflation is treated as a stochastic process. We assume that the evolution of $I_{\text{infl}}(t)$, the instantaneous rate of inflation over time t, is governed by the Vasicek (1977) model, whose dynamics are governed by the stochastic

Table 8: Description of test cases without infusions.

Case	T	$W(0)$	Non-zero goals $[t, C(t), U(t)]$	Non-zero infusions $[t, I(t)]$
1	10	100	$[10, 150, 1]$	–
2	10	100	$[10, 200, 1]$	–
3	10	100	$[10, 400, 1]$	–
4	40	100	$[40, 600, 1]$	–
5	40	100	$[40, 1200, 1]$	–
6	40	100	$[40, 2400, 1]$	–
7	100	100	$[100, 50000, 1]$	–
8	100	100	$[100, 500000, 1]$	–
9	100	100	$[100, 2000000, 1]$	–
10	3	100	$[2, 75, 0.9]$, $[3, 75, 1]$	–
11	20	100	$[10, 200, 1.3]$, $[20, 500, 1]$	–
12	20	100	$[15, 200, 1.3]$, $[20, 300, 1]$	–
13	35	100	$[5, 50, 1]$, $[25, 500, .5]$, $[35, 1000, 1]$	–
14	60	100	$[15, 300, .7]$, $[30, 6000, 1.2]$, $[45, 5000, .2]$, $[60, 20000, 1]$	–
15	25	100	$[3, 30, .2]$, $[5, 70, .3]$, $[8, 70, .3]$, $[25, 1000, 1]$	–
16	40	100	$[10, 150, 1.5]$, $[30, 400, 1]$, $[35, 500, 1]$, $[40, 600, 1]$	–
17	20	100	$[t, 15, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	–
18	20	100	$[t, 25, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	–
19	20	100	$[t, 50, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	–
20	20	100	$[t, 75, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	–
21	60	100	$[t, 20, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	–
22	60	100	$[t, t, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	–
23	60	100	$[t, t, 100 + t]$ for $t \in \{1, 2, 3, \ldots, 60\}$	–
24	60	100	$[t, t, 100 - t]$ for $t \in \{1, 2, 3, \ldots, 60\}$	–
25	60	100	$[t, 60 - \frac{t}{2}, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	–
26	60	100	$[t, 60 - \frac{t}{2}, 100 + t]$ for $t \in \{1, 2, \ldots, 60\}$	–
27	60	100	$[t, 60 - \frac{t}{2}, 100 - t]$ for $t \in \{1, 2, \ldots, 60\}$	–
28	30	100	$[3, 35, 1.5]$, $[6, 35, 1.3]$, $[9, 5, 0.4]$, $[12, 50, 1]$, $[15, 15, 0.7]$, $[18, 5, 0.3]$, $[21, 45, 0.6]$, $[24, 120, 0.9]$, $[27, 170, 1.1]$, $[30, 160, 1]$	–
29	16	12	$[16, 34.25, 26]$	–
30	16	21.63	$[8, 18.50, 18]$, $[16, 34.25, 26]$	–
31	16	38.99	$[4, 13.60, 14]$, $[8, 18.50, 18]$, $[12, 25.18, 22]$, $[16, 34.25, 26]$	–
32	16	70.27	$[2, 11, 66, 12]$, $[4, 13.60, 14]$, $[6, 15.87, 16]$, $[8, 18.50, 18]$, $[10, 21.59, 20]$, $[12, 25.18, 22]$, $[14, 29.37, 24]$, $[16, 34.25, 26]$	–
33	16	126.67	$[1, 10.8, 11]$, $[2, 11, 66, 12]$, $[3, 12.60, 13]$, $[4, 13.60, 14]$, $[5, 14.69, 15]$, $[6, 15.87, 16]$, $[7, 17.14, 17]$, $[8, 18.50, 18]$, $[9, 19.99, 19]$, $[10, 21.59, 20]$, $[11, 23, 32, 21]$, $[12, 25.18, 22]$, $[13, 27.20, 23]$, $[14, 29.37, 24]$, $[15, 31.72, 25]$, $[16, 34.25, 26]$	–

Table 9: Description of test cases with infusions. The base amount of infusion for odd-numbered test cases is $\tilde{I} = \frac{W(0)}{10(T-1)}$, rounded up to the nearest integer.

Case	T	$W(0)$	Non-zero goals $[t, C(t), U(t)]$	Non-zero infusions $[t, I(t)]$
34	10	100	$[10, 150, 1]$	$[1, 10]$
35	10	100	$[10, 200, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
36	10	100	$[10, 400, 1]$	$[1, 10]$
37	40	100	$[40, 600, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
38	40	100	$[40, 1200, 1]$	$[6, 12]$
39	40	100	$[40, 2400, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
40	100	100	$[100, 50000, 1]$	$[21, 19]$
41	100	100	$[100, 500000, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
42	100	100	$[100, 2000000, 1]$	$[27, 22]$
43	3	100	$[2, 75, 0.9]$, $[3, 75, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
44	20	100	$[10, 200, 1.3]$, $[20, 500, 1]$	$[6, 12]$
45	20	100	$[15, 200, 1.3]$, $[20, 300, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
46	35	100	$[5, 50, 1]$, $[25, 500, .5]$, $[35, 1000, 1]$	$[13, 15]$
47	60	100	$[15, 300, .7]$, $[30, 6000, 1.2]$, $[45, 5000, .2]$, $[60, 20000, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
48	25	100	$[3, 30, .2]$, $[5, 70, .3]$, $[8, 70, .3]$, $[25, 1000, 1]$	$[11, 14]$
49	40	100	$[10, 150, 1.5]$, $[30, 400, 1]$, $[35, 500, 1]$, $[40, 600, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
50	20	100	$[t, 15, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	$[10, 13]$
51	20	100	$[t, 25, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
52	20	100	$[t, 50, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	$[11, 14]$
53	20	100	$[t, 75, 1]$ for $t \in \{2, 4, 6, \ldots, 20\}$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
54	60	100	$[t, 20, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	$[38, 31]$
55	60	100	$[t, t, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
56	60	100	$[t, t, 100 + t]$ for $t \in \{1, 2, 3, \ldots, 60\}$	$[41, 34]$
57	60	100	$[t, t, 100 - t]$ for $t \in \{1, 2, 3, \ldots, 60\}$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
58	60	100	$[t, 60 - \frac{t}{2}, 1]$ for $t \in \{1, 2, 3, \ldots, 60\}$	$[45, 38]$
59	60	100	$[t, 60 - \frac{t}{2}, 100 + t]$ for $t \in \{1, 2, \ldots, 60\}$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
60	60	100	$[t, 60 - \frac{t}{2}, 100 - t]$ for $t \in \{1, 2, \ldots, 60\}$	$[49, 43]$
61	30	100	$[3, 35, 1.5]$, $[6, 35, 1.3]$, $[9, 5, 0.4]$, $[12, 50, 1]$, $[15, 15, 0.7]$, $[18, 5, 0.3]$, $[21, 45, 0.6]$, $[24, 120, 0.9]$, $[27, 170, 1.1]$, $[30, 160, 1]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
62	16	12	$[16, 34.25, 26]$	$[14, 3]$
63	16	21.63	$[8, 18.50, 18]$, $[16, 34.25, 26]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
64	16	38.99	$[4, 13.60, 14]$, $[8, 18.50, 18]$, $[12, 25.18, 22]$, $[16, 34.25, 26]$	$[15, 6]$
65	16	70.27	$[2, 11, 66, 12]$, $[4, 13.60, 14]$, $[6, 15.87, 16]$, $[8, 18.50, 18]$, $[10, 21.59, 20]$, $[12, 25.18, 22]$, $[14, 29.37, 24]$, $[16, 34.25, 26]$	$[t, 1.03^t \tilde{I}]$ for $t \in \{1, .., T-1\}$
66	16	126.67	$[1, 10.8, 11]$, $[2, 11, 66, 12]$, $[3, 12.60, 13]$, $[4, 13.60, 14]$, $[5, 14.69, 15]$, $[6, 15.87, 16]$, $[7, 17.14, 17]$, $[8, 18.50, 18]$, $[9, 19.99, 19]$, $[10, 21.59, 20]$, $[11, 23, 32, 21]$, $[12, 25.18, 22]$, $[13, 27.20, 23]$, $[14, 29.37, 24]$, $[15, 31.72, 25]$, $[16, 34.25, 26]$	$[16, 21]$

differential equation

$$dI_{\text{infl}} = -\kappa_{\text{infl}}(I_{\text{infl}} - \theta_{\text{infl}})dt + \sigma_{\text{infl}}dW, \tag{19}$$

where W is a Wiener process. The Vasicek model is an Ornstein–Uhlenbeck process, where θ_{infl} is the mean value of I_{infl}, κ_{infl} is the constant strength of the reversion to this mean value, and σ_{infl} is a volatility constant representing the strength of the randomness. This process admits negative values for inflation and hence, deflationary environments are possible in this model. The solution to equation (19) between times t and τ is

$$I_{\text{infl}}(\tau) = I_{\text{infl}}(t)e^{-\kappa_{\text{infl}}(\tau-t)} + \theta_{\text{infl}}(1 - e^{-\kappa_{\text{infl}}(\tau-t)}) + \sigma_{\text{infl}}\sqrt{\frac{1 - e^{-2\kappa_{\text{infl}}(\tau-t)}}{2\kappa_{\text{infl}}}}Z, \tag{20}$$

where Z is a standard normal random variable. During training for the metaRL model, we use this equation to update the inflation from the previous year to the current year by replacing t with $t-1$ and substituting $\tau = t$ in equation (20).

We must multiply all future real infusions and real goal costs by the cumulative effect of the inflation. To calculate this cumulative effect between time 0 to the current time t, we simply use the inflation that has been generated using equation (20) up to the current time t:

$$I_{\text{cum,past}}(t) = e^{\int_0^t I_{\text{infl}}(s)ds} = e^{I_{\text{infl}}(1)+I_{\text{infl}}(2)+\cdots+I_{\text{infl}}(t-1)}.$$

To adjust real infusions and real goal costs for projected inflation in the future (i.e., times greater than t), we will also need to multiply these infusions and costs by $\bar{I}_{\text{cum}}(\tau)$, the expected value of the cumulative effect of inflation between times t and τ. That is,

$$\bar{I}_{\text{cum}}(\tau) = E\left[e^{\int_t^\tau I_{\text{infl}}(s)ds}\right]. \tag{21}$$

The explicit formula for $\bar{I}_{\text{cum}}(\tau)$ is known (see, for instance, Vasicek (1977)). It is

$$\bar{I}_{\text{cum}}(\tau) = e^{I_{\text{infl}}(t)A(\tau)-B(\tau)}, \tag{22}$$

where

$$A(\tau) = \frac{1}{\kappa_{\text{infl}}}\left(1 - e^{-\kappa_{\text{infl}}(\tau-t)}\right) \text{ and} \tag{23}$$

$$B(\tau) = \left(\theta_{\text{infl}} - \frac{\sigma_{\text{infl}}^2}{2\kappa_{\text{infl}}^2}\right)\left(A(\tau) - (\tau - t)\right) - \frac{\sigma_{\text{infl}}^2}{4\kappa_{\text{infl}}}A^2(\tau). \tag{24}$$

As an example of using this formula, if the cost of a goal at time $= \tau$ (years) adjusts with inflation and its time $= t$ cost is \$120,000, then \$120,000$\times\bar{I}_{\text{cum}}(\tau)$ is the expected value at time $= t$ of the goal's nominal cost at time $= \tau$.

To account for the stochastic inflation in the MetaRL model, we make two straightforward changes. The first change is to introduce the new state variable I_{norm}, which is the current instantaneous inflation, $I_{\text{infl}}(t)$, normalized by the mean inflation in the Vasicek process, θ_{infl}:

$$I_{\text{norm}} = \frac{I_{\text{infl}}(t)}{\theta_{\text{infl}}}.$$

We add I_{norm} to the list of the other $3K + 5$ state variables in Table 7. The calculations for the state variables t_{norm} and $\mathbf{U}_{\text{agg}}[t:]$ remain the same. The second change is to the calculations of the remaining state variables, $W_{\min}(t)$, $W_{\max}(t)$, $\mathbf{C}_{\min}[t:]$, $\mathbf{C}_{\max}[t:]$, g_{sim}, and p_{sim} from Table 7. In the calculations of these $2K + 4$ state variables, we simply replace DISCOUNTSUM and DISCOUNTVEC defined in Algorithm 1 with DISCOUNTSUMINFL and DISCOUNTVECINFL defined in Algorithm 6. The only difference between Algorithm 1 and Algorithm 6 is that Algorithm 6 takes the average projected cumulative inflation, $\bar{I}_{\text{cum}}$, into account.

Algorithm 6 Discounting future goal costs with stochastic inflation

procedure DISCOUNTVECINFL($\mathbf{C}[t:]$, p, z, I_{init}, θ_{infl}, κ_{infl}, σ_{infl})
  $\mathbf{C}_{\text{disc}} \leftarrow$ initialize vector of zeros of same length as $\mathbf{C}[t:]$, which is $T - t + 1$
  for τ in $[0 : \text{length}(\mathbf{C}[t:])$ **do**
    $I_{\text{cum,avg}} \leftarrow \exp\left(\frac{I_{\text{init}}}{\kappa_{\text{infl}}}\left(1 - e^{-\kappa_{\text{infl}}\tau}\right) - \left(\theta_{\text{infl}} - \frac{\sigma_{\text{infl}}^2}{2\kappa_{\text{infl}}^2}\right)(A - \tau) - \frac{\sigma_{\text{infl}}^2}{4\kappa_{\text{infl}}}A^2\right)$
    $C_{\text{disc}}(\tau) \leftarrow C(\tau) \cdot \exp\left[-(\mu_p - \frac{1}{2}\sigma_p^2)h\tau - \sigma_p z\sqrt{h\tau}\right] \cdot I_{\text{cum,avg}}$
  end for
  return $\mathbf{C}_{\text{disc}}$
end procedure

procedure DISCOUNTSUMINFL($\mathbf{C}[t:]$, p, z, I_{init}, θ_{infl}, κ_{infl}, σ_{infl})
  return sum$\big($DISCOUNTVECINFL($\mathbf{C}[t:]$, p, z, I_{init}, θ_{infl}, κ_{infl}, σ_{infl})$\big)$
end procedure

References

Ang, A. and G. Bekaert (2002). International Asset Allocation With Regime Shifts. *Review of Financial Studies 15*(4), 1137–1187.

Ang, A. and G. Bekaert (2004). How Regimes Affect Asset Allocation. *Financial Analysts Journal 60*(2), 86–99.

Bae, S., Y. Lee, W. C. Kim, J. H. Kim, and F. J. Fabozzi (2024, August). Goal-based investing with goal postponement: multistage stochastic mixed-integer programming approach. *Annals of Operations Research*.

Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. *Intrinsically motivated learning in natural and artificial systems*, 17–47.

Barto, A. G. and S. Mahadevan (2003). Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems 13*, 341–379.

Beck, J., R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson (2023, January). A Survey of Meta-Reinforcement Learning. arXiv:2301.08028 [cs].

Bellalah, M., A. Hakim, K. Si, and D. Zhang (2020, June). Long term optimal investment with regime switching: inflation, information and short sales. *Annals of Operations Research*.

Bellman, R. (1952, August). On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences 38*(8), 716–719.

Bellman, R. (1966). Dynamic programming. *science 153*(3731), 34–37.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009, June). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, New York, NY, USA, pp. 41–48. Association for Computing Machinery.

Bernhart, G., S. Höcht, M. Neugebauer, M. Neumann, and R. Zagst (2011, February). Asset correlations in turbulent markets and the impact of different regimes on asset management. *Asia-Pacific Journal of Operational Research 28*(01), 1–23. Publisher: World Scientific Publishing Co.

Brandt, M. W., A. Goyal, P. Santa-Clara, and J. R. Stroud (2005, October). A Simulation Approach to Dynamic Portfolio Choice with an Application to Learning About Return Predictability. *The Review of Financial Studies 18*(3), 831–873.

Brandt, M. W. and P. Santa-Clara (2006). Dynamic Portfolio Selection by Augmenting the Asset Space. *The Journal of Finance 61*(5), 2187–2217. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.2006.01055.x.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020, July). Language Models are Few-Shot Learners. Number: arXiv:2005.14165 arXiv:2005.14165 [cs].

Browne, S. (1999, July). The Risk and Rewards of Minimizing Shortfall Probability. *The Journal of Portfolio Management 25*(4), 76–85.

Brunel, J. (2015). *Goals-Based Wealth Management: An Integrated and Practical Approach to Changing the Structure of Wealth Advisory Practices*. New York: Wiley.

Bulla, J., S. Mergner, I. Bulla, A. Sesboüé, and C. Chesneau (2011, November). Markov-switching asset allocation: Do profitable strategies exist? *Journal of Asset Management 12*(5), 310–321.

Capponi, A. and Y. Zhang (2023, October). A Continuous Time Framework for Sequential Goal-Based Wealth Management. *Management Science forthcoming*.

Chang, V., Q. A. Xu, S. H. Akinloye, V. Benson, and K. Hall (2024, July). Prediction of bank credit worthiness through credit risk analysis: an explainable machine learning study. *Annals of Operations Research*.

Chhabra, A. B. (2005, January). Beyond Markowitz: A Comprehensive Wealth Allocation Framework for Individual Investors. *The Journal of Wealth Management 7*(4), 8–34.

Consigli, G. and M. A. H. Dempster (1998, June). Dynamic stochastic programmingfor asset-liability management. *Annals of Operations Research 81*(0), 131–162.

Dapena, J. P., J. A. Serur, and J. R. Siri (2019, December). Risk on-Risk off: A regime switching model for active portfolio management. Technical Report 706, Universidad del CEMA. Publication Title: CEMA Working Papers: Serie Documentos de Trabajo.

Das, S., X. Huang, S. Adeshina, P. Yang, and L. Bachega (2023, October). Credit Risk Modeling with Graph Machine Learning. *INFORMS Journal on Data Science 2*(2), 197–217. Publisher: INFORMS.

Das, S., D. Ostrov, A. Radhakrishnan, and D. Srivastav (2023). Efficient Goal Probabilities: A New Frontier. *Journal of Investment Management 21*(3), 1–25.

Das, S. and S. Varma (2020). Dynamic Goals-Based Wealth Management Using Reinforcement Learning. *Journal of Investment Management 18*(2), 37–56.

Das, S. R., D. Ostrov, A. Casanova, A. Radhakrishnan, and D. Srivastav (2022). Optimal Goals-Based Investment Strategies for Switching between Bull and Bear Markets. *Journal of Wealth Management 24*(4), 8–36.

Das, S. R., D. Ostrov, A. Radhakrishnan, and D. Srivastav (2018). Goals-Based Wealth Management: A New Approach. *Journal of Investment Management 16*(3), 1–27.

Das, S. R., D. Ostrov, A. Radhakrishnan, and D. Srivastav (2020). Dynamic Portfolio Allocation in Goals-Based Wealth Management. *Computational Management Science 17*(June), 613–640.

Das, S. R., D. Ostrov, A. Radhakrishnan, and D. Srivastav (2022, July). Dynamic optimization for multi-goals wealth management. *Journal of Banking & Finance 140*, 106192.

Deguest, R., L. Martellini, V. Milhau, A. Suri, and H. Wang (2015). Introducing a Comprehensive Allocation Framework for Goals-Based Wealth Management. *Working paper, EDHEC Business School*.

Dempster, M. and G. Consigli (1998, January). The CALM stochastic programming model for dynamic asset-liability management. In *Worldwide Asset and Liability Modeling*, Publications of the Newton Institute, pp. 464–500. University of Cambridge. Editors: William Ziemba and John Mulvey.

Dempster, M. and E. Medova (2011). Planning for Retirement: Asset Liability Management for Individuals. In G. Mitra and K. Schwaiger (Eds.), *Asset and Liability Management Handbook*, pp. 409–432. London: Palgrave Macmillan UK.

Dixon, M. and I. Halperin (2020, February). G-Learner and GIRL: Goal Based Wealth Management with Reinforcement Learning. arXiv:2002.10990 [cs, q-fin, stat].

Duarte, V., J. Fonseca, A. S. Goodman, and J. A. Parker (2021, December). Simple Allocation Rules and Optimal Portfolio Choice Over the Lifecycle. Working Paper 29559, National Bureau of Economic Research. Series: Working Paper Series.

Finn, C., P. Abbeel, and S. Levine (2017, July). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. arXiv. arXiv:1703.03400 [cs].

Finn, C. and S. Levine (2018, February). Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm. arXiv. arXiv:1710.11622 [cs].

Grobys, K. (2012). Active PortofolioManagement in the Presence of Regime Switching: What Are the Benefits of Defensive Asset Allocation Strategies If the Investor Faces Bear Markets? *The Review of Finance and Banking 4*(1), 15–31. Publisher: EDITURA ASE.

Guidolin, M. and A. Timmermann (2007). Asset allocation under multivariate regime switching. *Journal of Economic Dynamics and Control 31*(11), 3503–3544.

Gupta, A., R. Mendonca, Y. Liu, P. Abbeel, and S. Levine (2018, February). Meta-Reinforcement Learning of Structured Exploration Strategies. arXiv:1802.07245 [cs].

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018, August). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs, stat].

Halperin, I. (2021, April). Distributional Offline Continuous-Time Reinforcement Learning with Neural Physics-Informed PDEs (SciPhy RL for DOCTR-L). arXiv:2104.01040 [physics, q-fin].

Hambly, B. M., R. Xu, and H. Yang (2021, November). Recent Advances in Reinforcement Learning in Finance. SSRN Scholarly Paper ID 3971071, Social Science Research Network, Rochester, NY.

Humplik, J., A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess (2019, October). Meta reinforcement learning as task inference. arXiv:1905.06424 [cs, stat].

Infanger, G. (2008, January). Chapter 5 - Dynamic asset allocation strategies using a stochastic dynamic programming aproach. In S. A. Zenios and W. T. Ziemba (Eds.), *Handbook of Asset and Liability Management*, pp. 199–251. San Diego: North-Holland.

Izacard, G., P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave (2022, November). Atlas: Few-shot Learning with Retrieval Augmented Language Models. arXiv:2208.03299 [cs].

Jiang, P., Q. Liu, and Y. Tse (2015). International Asset Allocation with Regime Switching: Evidence from the ETFs. *Asia-Pacific Journal of Financial Studies 44*(5), 661–687. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ajfs.12109.

Jiang, Z., D. Xu, and J. Liang (2017, July). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. arXiv:1706.10059 [cs, q-fin].

Kim, W. C., D.-G. Kwon, Y. Lee, J. H. Kim, and C. Lin (2020, March). Personalized goal-based investing via multi-stage stochastic goal programming. *Quantitative Finance 20*(3), 515–526. Publisher: Routledge _eprint: https://doi.org/10.1080/14697688.2019.1662079.

Kojima, T., S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa (2023, January). Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs].

Lewin, M. and C. H. Campani (2020). Portfolio Management under Multiple Regimes: Strategies that Outperform the Market. *RAC - Revista de Administração Contemporânea (Journal of Contemporary Administration) 24*(4), 300–316. Publisher: ANPAD - Associação Nacional de Pós-Graduação e Pesquisa em Administração.

Lin, S., J. Wan, T. Xu, Y. Liang, and J. Zhang (2022, July). Model-Based Offline Meta-Reinforcement Learning with Regularization. arXiv. arXiv:2202.02929 [cs].

Martellini, L., V. Milhau, and J. Mulvey (2020, April). Securing Replacement Income with Goal-Based Retirement Investing Strategies. *The Journal of Retirement 7*(4), 8–26. Publisher: Institutional Investor Journals Umbrella.

Merton, R. (1969). Lifetime Portfolio Selection under Uncertainty: The Continuous-Time Case. *The Review of Economics and Statistics 51*(3), 247–57.

Merton, R. (1971). Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory 3*(4), 373–413.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016, June). Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs].

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*. Publisher: [object Object] Version Number: 1.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015, February). Human-level control through deep reinforcement learning. *Nature 518*(7540), 529–533. Publisher: Nature Publishing Group.

Mohammed, S., R. Bealer, and J. Cohen (2021, October). Embracing advanced AI/ML to help investors achieve success: Vanguard Reinforcement Learning for Financial Goal Planning. arXiv:2110.12003 [cs, q-fin].

Mulvey, J. M. and B. Shetty (2004, January). Financial planning via multi-stage stochastic optimization. *Computers & Operations Research 31*(1), 1–20.

Mulvey, J. M. and H. Vladimirou (1992). Stochastic Network Programming for Financial Planning Problems. *Management Science 38*(11), 1642–1664. Publisher: INFORMS.

Nevmyvaka, Y., Y. Feng, and M. Kearns (2006, June). Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, New York, NY, USA, pp. 673–680. Association for Computing Machinery.

Osterrieder, J. and C. Gpt (2023, January). A Primer on Deep Reinforcement Learning for Finance.

Pakizer, S. (2017, July). Goals Based Investing for the Middle Class: Can Behavioral and Traditional Financial Policies Simplify Investing and Maximize Wealth? SSRN Scholarly Paper ID 3036231, Social Science Research Network, Rochester, NY.

Parker, F. J. (2020, April). Allocation of Wealth Both Within and Across Goals: *A Practitioner's Guide*. *The Journal of Wealth Management 23*(1), 8–21.

Parker, F. J. (2021, January). A Goals-Based Theory of Utility. *Journal of Behavioral Finance 22*(1), 10–25. Publisher: Routledge _eprint: https://doi.org/10.1080/15427560.2020.1716359.

Pourpanah, F., M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. J. Wu (2022). A review of generalized zero-shot learning methods. *IEEE transactions on pattern analysis and machine intelligence 45*(4), 4051–4070.

Sabharwal, R., S. J. Miah, S. F. Wamba, and P. Cook (2024, January). Extending application of explainable artificial intelligence for managers in financial organizations. *Annals of Operations Research*.

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015, June). Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, pp. 1889–1897. PMLR. ISSN: 1938-7228.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017, August). Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs].

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016, January). Mastering the game of Go with deep neural networks and tree search. *Nature 529*(7587), 484–489. Publisher: Nature Publishing Group.

Singhal, S. and P. C. Biswal (2019, January). Dynamic Commodity Portfolio Management: A Regime-switching VAR Model. *Global Business Review*, 0972150918811705. Publisher: SAGE Publications India.

Sutton, R. S. (1988, August). Learning to predict by the methods of temporal differences. *Machine Learning 3*(1), 9–44.

Sutton, R. S. and A. G. Barto (2018, November). *Reinforcement Learning, second edition: An Introduction* (2nd edition ed.). Cambridge, Massachusetts London, England: Bradford Books.

Team, G., T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, A. Tacchetti, A. Bulanova, A. Paterson, B. Tsai, B. Shahriari, C. L. Lan, C. A. Choquette-Choo, C. Crepy, D. Cer, D. Ippolito, D. Reid, E. Buchatskaya, E. Ni, E. Noland, G. Yan, G. Tucker, G.-C. Muraru, G. Rozhdestvenskiy, H. Michalewski, I. Tenney, I. Grishchenko, J. Austin, J. Keeling, J. Labanowski, J.-B. Lespiau, J. Stanway, J. Brennan, J. Chen, J. Ferret, J. Chiu, J. Mao-Jones, K. Lee, K. Yu, K. Millican, L. L. Sjoesund, L. Lee, L. Dixon, M. Reid, M. Mikuła, M. Wirth, M. Sharman, N. Chinaev, N. Thain, O. Bachem, O. Chang, O. Wahltinez, P. Bailey, P. Michel, P. Yotov, P. G. Sessa, R. Chaabouni, R. Comanescu, R. Jana, R. Anil, R. McIlroy, R. Liu, R. Mullins, S. L. Smith, S. Borgeaud, S. Girgin, S. Douglas, S. Pandya, S. Shakeri, S. De, T. Klimenko, T. Hennigan, V. Feinberg, W. Stokowiec, Y.-h. Chen, Z. Ahmed, Z. Gong, T. Warkentin, L. Peran, M. Giang, C. Farabet, O. Vinyals, J. Dean, K. Kavukcuoglu, D. Hassabis, Z. Ghahramani, D. Eck, J. Barral, F. Pereira, E. Collins, A. Joulin, N. Fiedel, E. Senter, A. Andreev, and K. Kenealy (2024, March). Gemma: Open Models Based on Gemini Research and Technology. arXiv:2403.08295 [cs].

Topaloglou, N., H. Vladimirou, and S. A. Zenios (2008, March). A dynamic stochastic programming model for international portfolio management. *European Journal of Operational Research 185*(3), 1501–1524.

Vasicek, O. (1977, November). An equilibrium characterization of the term structure. *Journal of Financial Economics 5*(2), 177–188.

Vo, H. T. and R. Maurer (2013, March). Dynamic Asset Allocation with Regime Shifts and Long Horizon CVaR-Constraints. SSRN Scholarly Paper ID 2191286, Social Science Research Network, Rochester, NY.

Wang, H., A. Suri, D. Laster, and H. Almadi (2011, April). Portfolio Selection in Goals-Based WealthManagement. *The Journal of Wealth Management 14*(1), 55–65.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD Thesis, King's College, Oxford.

Wei, J., M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le (2022, February). Finetuned Language Models Are Zero-Shot Learners. arXiv:2109.01652 [cs].

Zhang, W., J. Shi, X. Wang, and H. Wynn (2023, October). AI-powered decision-making in facilitating insurance claim dispute resolution. *Annals of Operations Research*.

Zheng, C., J. He, and C. Yang (2023, June). Optimal Execution Using Reinforcement Learning. arXiv:2306.17178 [cs, q-fin].

Zhou, M., Z. Liu, P. Sui, Y. Li, and Y. Y. Chung (2020). Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in neural information processing systems 33*, 11853–11864.

Zhou, X. Y. and G. Yin (2003, January). Markowitz's Mean-Variance Portfolio Selection with Regime Switching: A Continuous-Time Model. *SIAM Journal on Control and Optimization 42*(4), 1466–1482.