Reinforcement learning for Multiple Goals in Goals-Based Wealth Management

Sanjiv R. Das
Department of Finance
Santa Clara University
Santa Clara, USA
srdas@scu.edu

Sukrit Mittal
AI & Digital Transformation
Franklin Templeton Investments
Hyderabad, India
sukrit.mittal@franklintempleton.com

Daniel Ostrov
Mathematics and Computer Science
Santa Clara University
Santa Clara, USA
dostrov@scu.edu

Anand Radhakrishnan
AI & Digital Transformation
Franklin Templeton Investments
Hyderabad, India
andy.radhakrishnan@franklintempleton.com

Deep Ratna Srivastav
AI & Digital Transformation
Franklin Templeton Investments
San Ramon, USA
deepratna.srivastav@franklintempleton.com

Hungjen Wang
AI & Digital Transformation
Franklin Templeton Investments
New York, USA
hungjen.wang@franklintempleton.com

***Abstract*—We formulate the goals-based wealth management (GBWM) problem as a Markov Decision Process (MDP) and assess deep reinforcement learning (RL) and dynamic programming (DP) algorithms for dynamic solutions. These algorithms seek to (i) optimally choose from multiple goals over time and (ii) optimally switch between several model investment portfolios over time. In RL, we implement a proximal policy optimization (PPO) algorithm with a customized environment in which the goal-taking choices and investment portfolio selections are implemented with a discrete two-dimensional action space using a deep neural network. We show this RL approach performs almost as well as exact DP solutions. By showing that RL works well for GBWM in a comparative setting with DP, we see that RL may be a suitable method in other GBWM models whose larger state spaces render DP methods computationally infeasible.**

***Index Terms*—Goals-based wealth management, reinforcement learning, proximal policy optimization**

I. Introduction

This paper develops a custom environment for a reinforcement learning (RL) algorithm that dynamically optimizes an investor's actions in a goals-based wealth management (GBWM) context and compares it to existing dynamic programming (DP) algorithms used to solve this problem. We establish that RL is effective for GBWM.

In our GBWM context, the investor looks to maximize the expected value of the total utility/reward they accumulate over time from each financial goal they attain, like purchasing a car or paying for a child's college tuition. They may opt to forgo goals even if they have enough money to attain them, because it might sufficiently increase the chance of obtaining future goals that are more important to the investor. Therefore, our action space is two-dimensional in that it requires both making the binary decision of whether or not to take a goal when one is available, as well as selecting the best investment portfolio each rebalancing period. GBWM focuses on maximizing the probability that investors achieve their goals, taking into account each investor's individual circumstances. It expands the scope of traditional retirement planning [1] to cover the entire accumulation and decumulation phases of long-term wealth management.

In this paper, we compare the results from RL to the exact optimal expected utility, which can be determined using DP. We find for various sets of investor goals that the actions determined by using RL achieve 94–98% of the optimal expected utility determined by using DP.

We use a multi-objective genetic algorithm to help select hyperparameters that optimize performance balanced by keeping the runtime short. Performance is measured by the "RL Efficiency," meaning the percentage of the optimal expected utility (determined from DP) that is attained by the RL algorithm. On a simple Mac mini with an M2 chip, the RL algorithm runs in just over a half of a minute without even using a GPU, which is remarkable for such large-scale RL problems. Short runtimes are important in many practical applications such as robo-advising, where investors are generally watching a screen while waiting for results to be generated.

In traditional wealth management, risk is usually defined as the volatility of portfolio returns. Risk in GBWM, on the other hand, is associated with the probability of not achieving either a single goal [2], [3] or multiple goals weighted by their importance to the investor [4]. These notions of risk are often inconsistent with each other. For example, a 100% bond, 0% stock portfolio in an account looking to attain a stretch goal is a low risk portfolio in traditional wealth management, but a high risk portfolio in GBWM.

Early work on formulating GBWM was developed in a series of qualitative papers [5], [6], [7], summarized and expanded in subsequent papers, like [8] and [9]. The connection between GBWM and traditional mean-variance optimization was developed in [2], which gave a static, quantitative approach to GBWM for a single goal. Dynamic approaches to GBWM for a single goal were developed in [10], [11], [3]. This was extended to multiple goals in [4], which only requires

investors to state their relative goal preferences; see also [12] and [13]. Related early works by Browne look at meeting a series of liabilities [14] and also meeting goals by a deadline [15]. Papers like [16] look to examine GBWM with minimum replacement income constraints.

RL for goals-based optimization has been considered in [17], using Tabular-Q learning to optimize the probability of attaining a single future goal, and [18], using Q-learning and Deep RL to identify optimal savings rates across multiple goals and sources of income. The paper [19] extends the work of [20] to multiple targeted cashflows using G-learning, which was introduced in [21]. In [19], G-learning is applied in model-free RL where environment detection is noisy.

This paper uses Proximal Policy Optimization (PPO), an actor-critic method, which is model-free and on-policy, though off-policy variants have also been developed (e.g., [22]). PPO was introduced in [23] as an extension of Trust Region Policy Optimization (TRPO) in [24].[1] We believe this is the first paper to implement RL for GBWM in a way that attains solutions both very accurately and very quickly computationally.

The rest of this paper proceeds as follows: Section II describes our GBWM problem and the RL algorithm we deploy to solve it. Section III presents various numerical GBWM examples, comparing their RL and DP solutions and discussing the effect on the RL solutions of varying the hyperparameter choices. We offer a concluding discussion in Section IV.

II. Reinforcement Learning (RL) for GBWM

A. GBWM Model

We define a trajectory (a.k.a., a path) to be an investor's financial circumstances along with both their investment portfolio decisions and goal-taking decisions over the course of time. Time, t, over the course of a trajectory takes integer values measured in years, so $t = 0, 1, 2, ..., T$ years. Along a trajectory, there are times, t, when financial goals may be fulfilled, such as choosing to buy a vacation at, say, $t = 5$ or buying a specific annuity to retire at the final time horizon, $t = T$. At each time t where there is a goal, the cost of fulfilling the goal (resulting in a portfolio drawdown) will be denoted by $C(t)$, and the utility of the goal, which is the reward if the goal is fulfilled, will be denoted by $U(t)$.

The utility values in our GBWM context are just relative weights across goals that correspond to the relative importance of the goals to the investor. The overall aim of the investor is to optimize the expected value of the reward total, that is the sum of the utilities from fulfilled goals over $t \in [0, T]$.

The investor starts with wealth $W = W_0$ at $t = 0$. We denote the evolution of wealth over a trajectory by $\{t, W(t)\}$ where $t = 0, 1, 2, ..., T$. This evolution depends, of course, on which goals the investor decides to fulfill and the investment portfolio chosen each year.

[1]For a simple introduction to PPO, see https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b. See also https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12 for exposition of various technicalities.

At each time t where there is a goal, the investor must make a binary decision, denoted by i, either not to take the goal ($i = 0$) or to take the goal ($i = 1$). When $i = 1$ (that is, the goal at time t is fulfilled), the cost of the goal, $C(t)$, is subtracted from the wealth $W(t)$, and the utility, $U(t)$, attained from taking the goal is added to the investor's accumulated reward total. Of course, if $W(t) < C(t)$, then the investor does not have enough money to be able to fulfill the goal, so $i = 0$. An investor with $W(t) \geq C(t)$ on the other hand, may *choose* not to fulfill a goal (i.e., choose $i = 0$) so as to preserve cash in the hopes of fulfilling future goals that correspond to attaining a higher overall utility sum.

We will assume there are n investment portfolios to choose from, with annualized return means and standard deviations $\{\mu_j, \sigma_j\}$, where $j = 1, 2, ..., n$.[2] The values of $\{\mu_j, \sigma_j\}$ may correspond to locations on the Markowitz mean-variance efficient frontier [25] (that is, the best return portfolios for different levels of volatility), but they don't need to.

The investment portfolio—that is, the value of j—can change each year. Over the course of any year, we will assume that the investments evolve by the most common, basic model for stock evolution, which is geometric Brownian motion:

$$W(t+1) = W(t) \exp\left[\left(\mu_j - \frac{1}{2}\sigma_j^2\right) + \sigma_j Z\right], \qquad (1)$$

where Z has a standard normal distribution. We could choose another Markovian model in place of geometric Brownian motion if desired. As a simple example, we could replace the Z distribution in equation (1) with a t-distribution.

B. RL Setup

State Space: The "state space" $\mathcal{S}$ in our problem is two-dimensional, comprising time and wealth. We adopt the notation $s_t = \{t, W(t)\}$, to denote a specific location in the state space, meaning a specific time $t = 0, 1, ..., T$ and wealth at that time.

Action Space: The "action space" $\mathcal{A}$ comprises two different sets of actions: the value of i, meaning the binary choice of whether or not to take a goal if one is available at the beginning of year t, and the value of j, meaning the choice for year t from the n possible investment portfolios. We define $a_t = \{i(s_t), j(s_t)\}$, noting that $a_t \in \mathcal{A}$, a $2 \times n$ array of possible choices for the actions. At the final time, $t = T$, only i is used (that is, j is masked), since there is no portfolio evolution after the time horizon, T, is reached. Similarly, in years where there is no goal, i, as opposed to j, is masked.

Policy Function when t < T: When $t < T$, the action will not always be a deterministic function of the location in the state space. Indeed, in our RL algorithm, it will generally be a stochastic function. We define the "policy function" $\pi(a_t|s_t)$ to give the probability of taking action a_t if we are at the point s_t in the state space. Our RL algorithm using PPO looks to generate an optimal policy function. This policy function

[2]These investment portfolios are determined by the investor's preferences. For example, an aggressive investor would enable access to more volatile investment portfolios than a conservative investor.

is then used to obtain a deterministic function $\pi(s_t)$ defined by $\pi(s_t) = a_t = \text{argmax}_{a \in \mathcal{A}} \pi(a|s_t)$. This deterministic function, $\pi(s_t)$, is our final recommended optimized strategy, associating a specific action to each point in the sample space.

Algorithm over a Single Time Step when $t < T$: At the beginning of the year $t < T$, we are at a location $s_t = \{t, W(t)\}$ in the state space. We then choose an action a_t either stochastically or deterministically based on the policy function applied to this location. If there is an available goal at time t, the value from i specified by a_t determines whether or not we take this goal and therefore reduce $W(t)$ by $C(t)$ and add $U(t)$ to our reward total. Next, the value of j specified by a_t determines our investment portfolio for the year, which gives our probability distribution for $W(t+1)$ via equation (1) above, where $W(t)$ in equation (1) is the value of the wealth after purchasing the goal if $i = 1$. We then determine the value of $W(t+1)$ by drawing a random value for Z in equation (1) from a standard normal distribution.

State Transitions: Putting all of this together for a year $t < T$ allows us to define the "state transition probability density function" $f(s_{t+1}|s_t, a_t)$. This is the probability density function for reaching state s_{t+1} (defined by $W(t+1)$ at time $t+1$), given that we start at state s_t and choose action a_t.

Policy Function and Algorithm when $t = T$: At $t = T$, we only take or do not take the final goal, and this action, a_T, is deterministic: if $W(T) < C(T)$, then $i = 0$, and if $W(T) \geq C(T)$, then $i = 1$. In other words, the final goal at time T is fulfilled if possible, since there is no reason not to take the goal if the investor is able to. This defines a (deterministic) policy function, which is used at all stages of RL when $t = T$.

Rewards: We define $r_t(a_t)$, the rewards function, by the following: $r_t(a_t) = U(t)$ if a_t specifies that $i = 1$, and $r_t(a_t) = 0$ otherwise. Our overall goal is to find the deterministic policy function $\pi^*(s_t)$ that maximizes $E\left[\sum_{t=0}^{T} r_t(a_t)\right]$, the expected value of the accumulated utilities from fulfilled goals. The value function, $v^*(s_t)$, is defined to be this optimal expected value, conditioned that we are currently at the point $s_t = \{t, W(t)\}$ in the state space. That is,

$$v^*(s_t) = E_{\pi_t^*}\left[\sum_{u=t}^{T} r_u(a_u) \,\middle|\, s_t\right]. \tag{2}$$

DP is able to directly generate the optimal policy, $\pi_t^*(s_t)$, and the value function, $v^*(s_t)$, by using backwards-in-time recursion to implement the well-known Bellman equation [26].

C. Proximal Policy Optimization (PPO)

PPO is a deep-RL learning algorithm that uses deep neural networks to approximate $\pi_t^*(s_t)$ and $v^*(s_t)$, the optimal policy function and the value function, defined over a continuous state space. We will use the notation π_θ to emphasize the dependence of the policy function, π, on θ, the weights for the connections between the nodes (i.e., the neurons) of the network. We evolve the θ parameters so as to optimize π_θ.

In PPO, the approximated gradient of the total rewards with respect to θ is computed from batch data. Parameter updates use the gradient modulated by the learning rate, whose initial value is denoted by η, to move the current θ to θ_{new}. However, because the jump between θ and θ_{new} suggested by the gradient can create a change in the policy function that is too big for stability, PPO also uses a clip parameter, $\epsilon > 0$, to restrict the size of the jump in the policy function. More specifically, the smaller ϵ is, the closer the ratio $\frac{\pi_{\theta_{new}}}{\pi_\theta}$ must stay near one. This indirectly prevents large movements between the current θ and θ_{new}.

D. A Custom Environment for GBWM

This section describes our custom environment for GBWM that leverages OpenAI's environments,[3] and it describes the hyperparameters used in the RL PPO algorithm. Our algorithm uses the Stable Baselines library [27] to implement RL using PPO[4]. We created a new custom environment, which inherits the `Env` class from the `gymnasium` library. The main components of the environment are as follows:

1) Initialize the action space to be `MultiDiscrete`, where our discrete, two-dimensional action space is $2 \times n$, reflecting the binary goal-taking choice and the n investment portfolio choices.
2) Initialize the state space to be continuous in two-dimensions for time $t \in [0, T]$ and wealth $W(t) \in [0, W_t^{max}]$. While the state space is continuous in t, we will only be using t at the discrete values $0, 1, 2, ..., T$. The action space is a stochastic function of the state space, as defined by the policy function $\pi_\theta(a_t|s_t)$. We initialize the state space at $t = 0$ with the initial wealth, $W(0) = W_0$.
3) Define a function to take one time step by following the "Algorithm over a Single Time Step" for $t < T$ and for $t = T$, which are both given in Subsection II-B.
4) A trajectory (also called an "episode" in this context) sequentially uses this single time step function at $t = 0$, then $t = 1$, then $t = 2$, etc., until finishing at $t = T$. For each trajectory, τ, the total accumulated rewards, $R(\tau) = \sum_{t=0}^{T} r_t(a_t)$, is determined.

For each epoch (equivalent of "iteration") in RL, we run a batch of M trajectories (or episodes) using the same parameter set θ. The PPO algorithm then determines the next parameter set θ_{new}, which is used in the next epoch. The overall process flow of an RL run is depicted in Figure 1. This RL algorithm depends on five hyperparameters (or variables):

1) N_{traj} is the total number of trajectories in an RL run. Given T time steps in each trajectory, the number of time steps that will be simulated in an RL run is $N_{\text{traj}} \times T$.
2) M is the batch size, meaning the number of trajectories taken between updates to the parameter set θ. The number

[3]OpenAI's `gym` has now been converted into `Gymnasium`. See: https://gymnasium.farama.org. This GitHub repo offers a simple introduction to programming PPO: https://github.com/ericyangyu/PPO-for-Beginners.

[4]The Stable Baselines 3 library is here: https://github.com/DLR-RM/stable-baselines3. Since we have a multi-discrete action space, we are restricted to implementing PPO, as other RL algorithms in this library do not support a multi-discrete action space

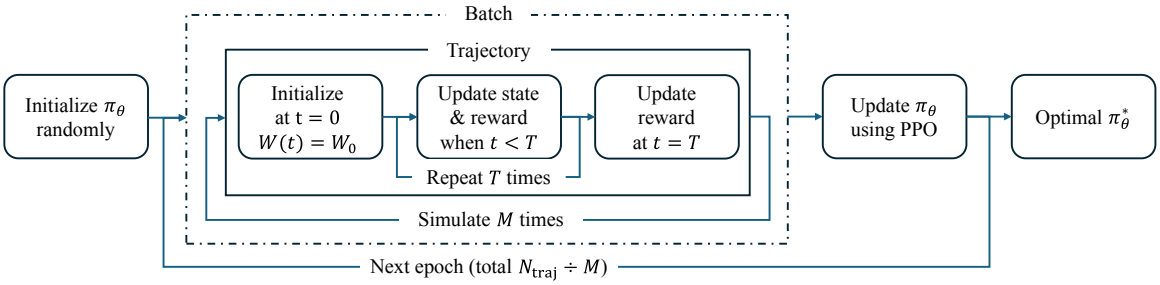

Fig. 1. Process flow of an RL run, starting with a randomly initialized policy, π_θ, to obtaining the optimal policy π^*_θ.

of time steps per batch is therefore $M \times T$, and the number of epochs in an RL run is $N_{\text{traj}} \div M$.

3) η is the initial learning rate used in PPO. This initial learning rate decays linearly to zero across the N_{traj} trajectories simulated by the algorithm.
4) ϵ is the clip parameter used in PPO.
5) N_{neur} is the number of neurons in each hidden layer of the neural networks for both the policy function and the estimated value function. Both neural networks have two inputs (the state variables t and $W(t)$) and have two hidden layers[5] of N_{neur} neurons each. The output for the policy neural network is a two-dimensional $(2 \times n)$ layer, comprising the goal-taking and the investment portfolio choices. The output for the estimated value function is an approximation of the optimal expected reward for the remainder of the trajectory after time t, for a given $W(t)$.

E. Measuring RL's Efficiency

Once an RL run is finished (having run all N_{traj} trajectories), it produces a final stochastic policy function, $\pi_{\theta_{\text{final}}}(a|s_t)$. From this, we obtain our desired deterministic policy function, $\pi_{RL}(s_t)$, using the formula

$$\pi_{RL}(s_t) = a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \pi_{\theta_{\text{final}}}(a|s_t).$$

We want to see how close in performance $\pi_{RL}(s_t)$ is to $\pi_{DP}(s_t) = \pi^*(s_t)$, the optimal policy function that we determine from DP using the algorithm detailed in [4]. We therefore run 100,000 Monte Carlo trajectories using π_{RL} and then run 100,000 Monte Carlo trajectories using π_{DP}, making sure that the $T \times 100,000$ standard normal samples needed for the geometric Brownian motion model in equation (1) are the same sequence for RL and for DP. We then average the total rewards for the RL trajectories: $R_{RL,avg} = \frac{1}{100,000} \sum_{j=1}^{100,000} R_{RL}(\tau_j)$, and for the DP trajectories: $R_{DP,avg} = \frac{1}{100,000} \sum_{j=1}^{100,000} R_{DP}(\tau_j)$, where τ_j is the j^{th} trajectory. Their ratio is the RL Efficiency metric:[6]

$$\text{RL Efficiency} = \frac{R_{RL,avg}}{R_{DP,avg}} \leq 1. \tag{3}$$

[5]We note that performance diminished when we changed the number of hidden layers away from two.

[6]In practice, using 100,000 Monte Carlo trajectories is easily a large enough number to prevent the numerical possibility of a pathological case where the calculated RL Efficiency is greater than one.

For the results in Section III, we run the entire RL algorithm 30 times (unless otherwise specified) and average the results of these 30 runs to find the "Mean RL Efficiency." That is,

$$\text{Mean RL Efficiency} = \frac{\sum_{k=1}^{30} (\text{RL Efficiency})_k}{30}. \tag{4}$$

Obviously, the closer the Mean RL Efficiency is to one, the better our RL algorithm's approximation is to the optimal policy function.

F. Tuning the Hyperparameters

To determine the optimal hyperparameter values, we first found rough ranges for the hyperparameters in which we have a reasonable RL Efficiency, as well as runtime. These rough ranges were $N_{\text{traj}} \in$ [25,000, 5,000,000], $M \in$ [3200, 8000], $\eta \in$ [0.001, 0.100], $\epsilon \in$ [0.05, 0.75], and $N_{\text{neur}} \in$ [32, 128].

Executing the RL algorithm for each combination of hyperparameters in such large rough ranges is computationally expensive. To reduce these ranges to be computationally feasible, we used a surrogate-assisted approach. In this approach, we first trained a metamodel to capture the relationships between the hyperparameters and our two objectives of maximizing RL Efficiency and minimizing runtime. We then ran the multi-objective genetic algorithm NSGA-II [28] using the trained metamodel for function evaluations. The specifics of this process are listed below:

1) Within the given rough bounds, we sampled about 2000 combinations of the five hyperparameters and executed an RL run for each combination. This created a dataset with hyperparameter combinations as the inputs, and the corresponding RL Efficiency and runtime as the outputs.
2) We tried nine metamodels found within four different machine learning methods.[7] From those, we selected the extra-trees regression [29], because it achieved the highest coefficient of determination (that is, R^2 score).
3) Next, we ran the NSGA-II algorithm using the `pymoo` framework[8] for 500 iterations. This was used to produce 100 points on the Pareto frontier where the hyperparameter combinations produce RL Efficiency and runtime

[7]Specifically, we tried *linear methods*: standard linear regression, ridge regression, and elastic-net regression; *tree-based methods*: extra-trees regression and random forest regression; *boosting methods*: XGBoost and AdaBoost; and *non-linear methods*: kNN and support vector regression.

[8]https://pymoo.org/algorithms/moo/nsga2.html

results that cannot be dominated by other hyperparameter combinations. From this frontier, we selected the two most extreme points, which correspond to the highest RL Efficiency and to the lowest runtime. At the highest RL Efficiency, the hyperparameter values were N_{traj} = 100,000, M = 4800, η = 0.01, ϵ = 0.50, and N_{neur} = 128. At the lowest runtime, the hyperparameter values were N_{traj} = 25,000, M = 4800, η = 0.01, ϵ = 0.75, and N_{neur} = 32. We then tightened our hyperparameter bounds considerably by only considering values within the ranges between these two extreme cases.

Analyzing combinations of hyperparameters within these reduced bounds led to selecting the following base case hyperparameter values: N_{traj} = 50,000; $M = 4800$; $\eta = 0.01$; $\epsilon = 0.50$; $N_{\text{neur}} = 64$. Tables II and III, which will later be presented in Subsection III-E, explore the sensitivity of these base case choices, confirming that they do indeed implement a satisfactory trade-off between RL efficiency and runtime.

III. Goals and Solutions

Section II described a class of GBWM problems and our RL approach to solving them. This section defines examples from this class of GBWM problems by specifying (i) initial wealth values, (ii) the times, costs, and utility values of various sets of goals, and (iii) n possible investment portfolios that the investor can switch between each year. As discussed in Subsection II-E, we note that the RL or DP results stated in this section are based on averaging 100,000 Monte Carlo trajectories using the deterministic policy function produced from an RL run or from DP.

A. Model Setup and Goals

We consider a 16-year time horizon for meeting a number of goals, $NG = 1, 2, 4, 8,$ or 16 goals. The goals are equally spaced over these 16 years, so, for example, in the case of 2 goals, the goals occur at $t = 8$ years and $t = 16$ years. In the case of 4 goals, they occur at $t = 4, 8, 12,$ and 16 years.

We have selected the cost of each goal in our model to be $C(t) = 10 \times 1.08^t$ dollars (noting that other currency units can be used in place of dollars, of course). When a goal occurs, the investor can either forgo taking the goal, meaning they pay nothing and accrue no utility, or they can fulfill the goal, in which case they pay the cost of the goal and accrue a utility that we have selected for our model to be $U(t) = 10+t$. While both $C(t)$ and $U(t)$ increase with time here, $U(t)$ obviously grows linearly, while $C(t)$ grows exponentially.

The investor is looking to optimize the sum of the utility from all their fulfilled goals. We let the investor start with $W_0 = 12 \cdot (NG)^{0.85}$ dollars, meaning the investor starts with more money when they have more goals. This scaling for the initial investment generally leads to the optimal expected value of the sum of the utility from fulfilled goals being about 70-80% of the total utility sum that would be obtained if it were possible to attain all the goals.

TABLE I
RL performance relative to DP as the number of goals varies. The metric for performance is the RL Efficiency defined in equation (3) of Subsection II-E. The RL results reported here use the base case hyperparameter values given in Subsection II-F.

Number of Goals	RL Efficiency Mean	Median	Standard Deviation	Skewness	Average RL runtime (minutes)	30 run noise confidence interval
1	0.974	0.973	0.0080	-0.653	0.540	±0.0041
2	0.948	0.950	0.0100	-0.378	0.543	±0.0052
4	0.961	0.963	0.0088	-0.451	0.541	±0.0045
8	0.978	0.978	0.0051	-0.296	0.541	±0.0026
16	0.978	0.979	0.0058	-0.371	0.543	±0.0030

B. Model Investment Portfolios

Our investment portfolio policy is discrete. More specifically, we select from among $n = 15$ investment portfolios on the efficient frontier.[9] The efficient frontier is generated using Markowitz portfolio optimization [25] with the following mean return vector (μ) and return covariance matrix ($\mathbf{\Sigma}$):

$$\mu = \begin{bmatrix} 0.0493 \\ 0.0770 \\ 0.0886 \end{bmatrix}, \quad \mathbf{\Sigma} = \begin{bmatrix} 0.0017 & -0.0017 & -0.0021 \\ -0.0017 & 0.0396 & 0.0309 \\ -0.0021 & 0.0309 & 0.0391 \end{bmatrix}.$$

These numbers correspond to historical returns for U.S. bonds, U.S. stock, and international stock.

We have chosen the 15 model portfolios on the efficient frontier to have mean returns that are equally spaced over the range $[0.052632, 0.088636]$. This corresponds to return standard deviations in the range $[0.037351, 0.195437]$. The portfolios are labeled (from #1 to #15) in order of being increasingly aggressive, as is used in [30], [3], [31], [4].

C. Analysis for RL using Base Case Hyperparameters

We obtain solutions for the setup, goals, and investment portfolios just described in Subsections III-A and III-B, using RL and DP, as described in Section II. For RL, we use the base case hyperparameters specified in Subsection II-F.

Each row of Table I represents a specific case for the number of goals (NG). For each of these cases, we ran our RL algorithm 30 times. The mean and the median RL Efficiency (defined in equation (3)) from these 30 runs are shown in Table I. We note the median strays no more than 0.002 from the mean, and the average expected utility produced by RL stays within 94–98% of DP's optimal expected utility. Further, the RL Efficiency from any single run strayed no more than 0.02 from the mean RL Efficiency given in the table.

The standard deviation and the skewness of the efficiency metric given in Table I are determined using 200 runs, instead

[9] "The efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return. Portfolios that lie below the efficient frontier are sub-optimal because they do not provide enough return for the level of risk. Portfolios that cluster to the right of the efficient frontier are sub-optimal because they have a higher level of risk for the defined rate of return."– https://www.investopedia.com/terms/e/efficientfrontier.asp.

of 30. Given how close the mean (and median) are to one, which is their theoretical maximum, it's no surprise that 1) the standard deviation is small (0.01, at most), 2) the closer the mean/median is to one, the smaller the standard deviation, and 3) the distributions are skewed to the left, although this skewness is mild, ranging from -0.3 to -0.7.

The average runtime of a single run for our RL algorithm stays in the range of 0.540–0.543 minutes; that is, 32–33 seconds. These runs were performed on a Mac Mini with an M2 chip. Since these runs only use the Mac Mini's CPU, not the GPU, they could be performed even faster with RL programs that can take advantage of the GPU.

We note the mean RL Efficiency presented in Table I's "Mean" column corresponds to the definition of the "Mean RL Efficiency" given in equation (4). The last column in Table I gives a 95% confidence interval for the difference between two independently determined Mean RL Efficiencies. Since the Mean RL Efficiency is, by the central limit theorem, approximately normally distributed and the difference of normal random variables is normal, the 95% confidence interval for the difference between two Mean RL Efficiencies is well approximated by $\pm 2 \cdot SD\left[\overline{X_1} - \overline{X_2}\right] = \pm 2 \cdot \sqrt{V\left[\overline{X_1} - \overline{X_2}\right]} = \pm 2 \cdot \sqrt{V\left[\overline{X_1}\right] + V\left[\overline{X_2}\right]} \approx \pm 2 \cdot \sqrt{\frac{s^2}{30} + \frac{s^2}{30}} = \pm 2 \cdot \frac{s}{\sqrt{15}}$, where s is the (sample) standard deviation reported in Table I. This formula corresponds to the last column of Table I, which provides a range for differences in reported Mean RL Efficiencies that can be attributed just to noise, which we will use in Subsection III-E.

D. Comparison to Benchmarks and Comparing Efficiency Loss from RL's Investment and Goal-taking Strategies

The results in Table I's "Mean" column are shown graphically in the top plot in Figure 2 by the blue path for the Mean RL Efficiency and the straight orange line at 1 for the DP efficiency. But how effective is RL versus other benchmark strategies? The top plot in Figure 2 also shows the mean efficiency of three benchmark strategies compared to DP:

- *Greedy goal-taking* (in purple): Greedy goal-taking uses the optimal investment portfolio policy from RL, but its goal-taking policy is simply to take any goal if the portfolio has sufficient funds (i.e., the portfolio wealth is greater than or equal to the cost of the goal).
- *Buy and hold* (in red): Buy and hold maintains a single investment portfolio throughout all 16 years by rebalancing back to it each year. For illustrative purposes, we chose investment portfolio #8, because it is in the middle of our 15 possible investment portfolios, striking a balance between being conservative and aggressive. We randomize goal-taking for $t < T$ with a half and half chance of taking or not taking a goal when there is sufficient wealth to be able to take the goal.
- *Random* (in green): We follow the same randomized goal-taking strategy as in the buy and hold case, and we also randomly select from one of the 15 possible investment portfolios each year.

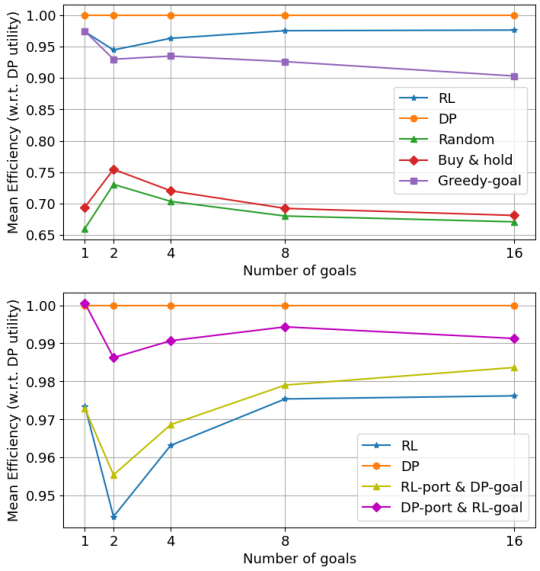

Fig. 2. A comparison of the efficiency of various portfolio strategies. The top plot compares the optimal solution (DP) for our five different numbers of goals with (i) the RL solution, as well as with (ii) random investment portfolios and random goal choices, (iii) a buy and hold investment portfolio strategy, and (iv) the greedy strategy for goal-taking. The lower plot shows the comparison of DP and RL with two intermediate strategies: (i) the DP investment portfolio strategy combined with the RL goal-taking strategy and (ii) the DP goal-taking strategy combined with the RL investment portfolio strategy.

RL outperforms all three of these benchmarks. The largest drop in efficiency occurs when we go from the greedy goal-taking strategy to the buy and hold strategy.

This large drop suggests that obtaining the optimal investment portfolio choice is more important than obtaining the optimal goal-taking choice. Further evidence for this is demonstrated in the lower plot in Figure 2 where we compare DP (which is the same orange line as in the top plot) and RL (which is the same blue path as in the top plot) with two intermediate strategies:

- the DP strategy for investment portfolio choice combined with the RL strategy for goal-taking (in magenta), and
- the DP strategy for goal-taking combined with the RL strategy for investment portfolio choice (in olive).

Since the magenta path stays well above the olive path, it is clear that RL's investment portfolio strategy moves the investor further from the ideal DP strategy than RL's goal-taking strategy does.

E. Sensitivity to Changing the Hyperparameter Values

We next explore the sensitivity of the Mean RL Efficiency to changing each hyperparameter from its base case value given in Subsection II-F. These sensitivity results are shown in Tables II and III. For each individual hyperparameter sensitivity, 30 runs are performed, and the resulting Mean RL Efficiency, as defined in equation (4) of Subsection II-E, is reported. Table II also shows the effects on the mean runtime of changing two of our five hyperparameters, N_{traj} and N_{neur}. Table III shows

TABLE II

RL ALGORITHM PERFORMANCE FOR A GIVEN NUMBER OF GOALS AS EITHER THE TOTAL NUMBER OF TRAJECTORIES, N_{traj}, OR THE NUMBER OF NEURONS IN BOTH POLICY AND VALUE FUNCTIONS, N_{neur}, ARE DECREASED (TOP ROW), LEFT ALONE (MIDDLE ROW), OR INCREASED (BOTTOM ROW) FROM THEIR BASE CASE VALUES. ALL OTHER HYPERPARAMETERS REMAIN AT THEIR BASE CASE VALUES GIVEN IN SUBSECTION II-F. RESULTS ARE GIVEN FOR THE MEAN RL EFFICIENCY (DEFINED IN EQUATION (4) OF SUBSECTION II-E) AND THE MEAN RL RUNTIME.

Number of Goals	Total number of trajectories, N_{traj}	Mean RL Efficiency	RL runtime (minutes)	Number of neurons, N_{neur}	Mean RL Efficiency	RL runtime (minutes)
1	25000	0.952	0.295	32	0.974	0.497
	50000	0.974	0.540	64	0.974	0.540
	100000	0.982	1.018	128	0.975	0.623
2	25000	0.928	0.295	32	0.947	0.497
	50000	0.948	0.543	64	0.948	0.543
	100000	0.962	1.010	128	0.948	0.619
4	25000	0.939	0.297	32	0.960	0.496
	50000	0.961	0.541	64	0.961	0.541
	100000	0.960	1.008	128	0.965	0.614
8	25000	0.938	0.300	32	0.975	0.497
	50000	0.978	0.541	64	0.978	0.541
	100000	0.972	1.010	128	0.975	0.612
16	25000	0.947	0.301	32	0.977	0.499
	50000	0.978	0.543	64	0.978	0.543
	100000	0.971	1.019	128	0.979	0.612

TABLE III

ALGORITHM PERFORMANCE AS EITHER THE BATCH SIZE, M, THE INITIAL LEARNING RATE, η, OR THE PPO CLIP PARAMETER, ϵ, ARE DECREASED (TOP ROW), LEFT ALONE (MIDDLE ROW), OR INCREASED (BOTTOM ROW) FROM THEIR BASE CASE VALUES. ALL OTHER HYPERPARAMETERS REMAIN AT THEIR BASE CASE VALUES GIVEN IN SUBSECTION II-F. RESULTS ARE GIVEN FOR THE MEAN RL EFFICIENCY (DEFINED IN EQUATION (4) OF SUBSECTION II-E).

# of Goals	Batch size, M	Mean RL Efficiency	Initial learning rate, η	Mean RL Efficiency	PPO clip parameter, ϵ	Mean RL Efficiency
1	2400	0.977	0.005	0.972	0.25	0.969
	4800	0.974	0.010	0.974	0.50	0.974
	7200	0.971	0.050	0.791	0.75	0.969
2	2400	0.954	0.005	0.944	0.25	0.945
	4800	0.948	0.010	0.948	0.50	0.948
	7200	0.943	0.050	0.859	0.75	0.950
4	2400	0.960	0.005	0.961	0.25	0.960
	4800	0.961	0.010	0.961	0.50	0.961
	7200	0.958	0.050	0.911	0.75	0.964
8	2400	0.969	0.005	0.976	0.25	0.967
	4800	0.978	0.010	0.978	0.50	0.978
	7200	0.967	0.050	0.928	0.75	0.977
16	2400	0.967	0.005	0.977	0.25	0.972
	4800	0.978	0.010	0.978	0.50	0.978
	7200	0.968	0.050	0.934	0.75	0.973

the effects of changing our three other hyperparameters, M, η, and ϵ, none of which affect runtime.

Table II shows that the runtime is linear in N_{traj}, the total number of trajectories per run, as would be expected. While the Mean RL Efficiency is always worse at N_{traj} = 25,000, it tends to be better at N_{traj} = 50,000 than at N_{traj} = 100,000 when there are many goals, but this reverses when there are fewer goals. Additional increases to N_{traj}, even going as high as 5,000,000, does not improve the Mean RL Efficiency. Indeed, it becomes slightly worse, which cannot be explained by the noise given in the final column of Table I.

Table II shows that changing N_{neur}, the number of neurons per hidden layer in both the policy function and the value function, makes no statistically meaningful difference. That is, the variation shown in Table II is within the noise calculated in the last column of Table I. The results are worse when one or three hidden layers are used instead of two in the base case.

In Table III we see that the effect of varying M, the batch size, parallels the effect of varying N_{traj} in Table II. That is, while the Mean RL Efficiency is almost always worse at $M = 7200$, it tends to be better at $M = 4800$ than at $M = 2400$ when there are many goals, but this reverses when there are fewer goals. And again, this effect cannot be explained by the noise given in the final column of Table I.

Both η, the initial learning rate, and ϵ, the PPO clip parameter, limit how much change there can be between successive batch run conditions. When they are too small, they will slow down the algorithm's convergence, but when they are too large, instability can occur, giving worse results. Table III demonstrates this well. The results are slightly worse when η or ϵ are below their base case values (although often by an amount small enough to be explained by the noise given in the final column of Table I). However, when η is above its base case value, we see that instability sets in, making the results much worse, and when ϵ is above its base case value, we begin to see instability affecting some, although not all, of these cases as well, although, again, often by an amount small enough to be explained by the noise.

IV. CONCLUDING DISCUSSION

This paper solves a variety of GBWM problems using RL (PPO) on a custom environment, leveraging the OpenAI `gym` library. Optimal hyperparameters are determined with the aid of a multi-objective genetic algorithm (NSGA-II). RL is efficient, coming to within 94–98% of the optimal DP solution, and, further, RL easily beats other benchmarks such as greedy goal-taking, buy and hold investing strategies, and random policies. The underperformance of RL relative to DP may be ascribed more to RL's marginally weaker investment portfolio policy than to RL's weaker goal-taking policy. Runtimes for RL in this paper were quite fast, requiring only 33 seconds on an Apple Mac Mini with an M2 chip, and only using the CPU, not the GPU.

This paper's GBWM problems were purposely chosen to be solvable using DP, so the optimal solution can be obtained with the Bellman equation [26]. This allowed us to measure our RL solution's accuracy. With its high accuracy now shown, we can explore extending the RL algorithm (in this paper) to additional dimensions in the state space, where DP becomes completely impractical. Some examples include new state

variables for stochastic inflation or stochastic interest rates. Further, because the Bellman equation is computed backwards in time, DP cannot be applied to many forwards-in-time phenomena that RL can be applied to. Examples include tracking the history of stock purchases for determining capital gains taxes, or how best to defer taking goals until later. Further research is predicated on the interpretability of policy and value function neural networks to support their understanding and wider adoption.

REFERENCES

[1] M. Dempster and E. Medova, "Planning for Retirement: Asset Liability Management for Individuals," in *Asset and Liability Management Handbook*, G. Mitra and K. Schwaiger, Eds. London: Palgrave Macmillan UK, 2011, pp. 409–432. [Online]. Available: https://doi.org/10.1057/9780230307230_16

[2] S. R. Das, D. Ostrov, A. Radhakrishnan, and D. Srivastav, "Goals-Based Wealth Management: A New Approach," *Journal of Investment Management*, vol. 16, no. 3, pp. 1–27, 2018.

[3] ——, "Dynamic Portfolio Allocation in Goals-Based Wealth Management," *Computational Management Science*, vol. 17, no. June, pp. 613–640, 2020.

[4] ——, "Dynamic optimization for multi-goals wealth management," *Journal of Banking & Finance*, vol. 140, p. 106192, Jul. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378426621001515

[5] D. Nevins, "Goals-Based Investing: Integrating Traditional and Behavioral Finance," *The Journal of Wealth Management*, vol. 6, no. 4, pp. 8–23, Jan. 2004. [Online]. Available: http://jwm.iijournals.com/lookup/doi/10.3905/jwm.2004.391053

[6] A. B. Chhabra, "Beyond Markowitz: A Comprehensive Wealth Allocation Framework for Individual Investors," *The Journal of Wealth Management*, vol. 7, no. 4, pp. 8–34, Jan. 2005. [Online]. Available: http://jwm.iijournals.com/lookup/doi/10.3905/jwm.2005.470606

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

[8] J. Brunel, "Goals-based wealth management: An integrated and practical approach to changing the structure of wealth advisory practices." New York: Wiley, 2015. [Online]. Available: https://www.wiley.com/en-us/Goals+Based+Wealth+Management%3A+An+Integrated+and+Practical+Approach+to+Changing+the+Structure+of+Wealth+Advisory+Practices-p-9781118995907

[9] F. J. Parker, "Allocation of Wealth Both Within and Across Goals: *A Practitioner's Guide*," *The Journal of Wealth Management*, vol. 23, no. 1, pp. 8–21, Apr. 2020. [Online]. Available: http://jwm.pm-research.com/lookup/doi/10.3905/jwm.2020.1.102

[10] A. Consiglio, F. Cocco, and S. Zenios, "Scenario optimization asset and liability modelling for individual investors," *Annals of Operations Research*, vol. 152, no. 1, pp. 167–191, 2007, publisher: Springer. [Online]. Available: https://ideas.repec.org/a/spr/annopr/v152y2007i1p167-19110.1007-s10479-006-0133-5.html

[11] H. Wang, A. Suri, D. Laster, and H. Almadi, "Portfolio Selection in Goals-Based WealthManagement," *The Journal of Wealth Management*, vol. 14, no. 1, pp. 55–65, Apr. 2011. [Online]. Available: http://jwm.iijournals.com/lookup/doi/10.3905/jwm.2011.14.1.055

[12] A. Capponi and Y. Zhang, "A Continuous Time Framework for Sequential Goal-Based Wealth Management," *Management Science*, vol. forthcoming, Oct. 2023. [Online]. Available: https://papers.ssrn.com/abstract=4121931

[13] W. C. Kim, D.-G. Kwon, Y. Lee, J. H. Kim, and C. Lin, "Personalized goal-based investing via multi-stage stochastic goal programming," *Quantitative Finance*, vol. 20, no. 3, pp. 515–526, Mar. 2020, publisher: Routledge _eprint: https://doi.org/10.1080/14697688.2019.1662079. [Online]. Available: https://doi.org/10.1080/14697688.2019.1662079

[14] S. Browne, "Survival and Growth with a Liability: Optimal Portfolio Strategies in Continuous Time," *Mathematics of Operations Research*, vol. 22, no. 2, pp. 468–493, May 1997. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/moor.22.2.468

[15] ——, "Reaching Goals by a Deadline: Digital Options and Continuous-Time Active Portfolio Management," *Advances in Applied Probability*, vol. 31, no. 2, pp. 551–577, 1999. [Online]. Available: http://www.jstor.org/stable/1428122

[16] L. Martellini, V. Milhau, and J. Mulvey, "Securing Replacement Income with Goal-Based Retirement Investing Strategies," *The Journal of Retirement*, vol. 7, no. 4, pp. 8–26, Apr. 2020, publisher: Institutional Investor Journals Umbrella. [Online]. Available: https://jor.pm-research.com/content/7/4/8

[17] S. Das and S. Varma, "Dynamic Goals-Based Wealth Management Using Reinforcement Learning," *Journal of Investment Management*, vol. 18, no. 2, pp. 37–56, 2020. [Online]. Available: https://joim.com/article/dynamic-goals-based-wealth-management-using-reinforcement-learning/

[18] S. Mohammed, R. Bealer, and J. Cohen, "Embracing advanced AI/ML to help investors achieve success: Vanguard Reinforcement Learning for Financial Goal Planning," Oct. 2021, arXiv:2110.12003 [cs, q-fin]. [Online]. Available: http://arxiv.org/abs/2110.12003

[19] M. Dixon and I. Halperin, "G-Learner and GIRL: Goal Based Wealth Management with Reinforcement Learning," Feb. 2020, arXiv:2002.10990 [cs, q-fin, stat]. [Online]. Available: http://arxiv.org/abs/2002.10990

[20] I. Halperin and I. Feldshteyn, "Market Self-Learning of Signals, Impact and Optimal Trading: Invisible Hand Inference with Free Energy (Or, How We Learned to Stop Worrying and Love Bounded Rationality)," Rochester, NY, May 2018. [Online]. Available: https://papers.ssrn.com/abstract=3174498

[21] R. Fox, A. Pakman, and N. Tishby, "Taming the noise in reinforcement learning via soft updates," in *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, ser. UAI'16. Arlington, Virginia, USA: AUAI Press, Jun. 2016, pp. 202–211.

[22] W. Meng, Q. Zheng, G. Pan, and Y. Yin, "Off-Policy Proximal Policy Optimization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 8, pp. 9162–9170, Jun. 2023, number: 8. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26099

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," in *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, Jun. 2015, pp. 1889–1897, iSSN: 1938-7228. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html

[25] H. H. Markowitz, "Portfolio Selection," *Journal of Finance*, vol. 6, pp. 77–91, 1952.

[26] R. Bellman, "On the Theory of Dynamic Programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, Aug. 1952. [Online]. Available: https://www.pnas.org/content/38/8/716

[27] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, conference Name: IEEE Transactions on Evolutionary Computation. [Online]. Available: https://ieeexplore.ieee.org/document/996017/authors#authors

[29] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr. 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

[30] S. R. Das, D. Ostrov, A. Casanova, A. Radhakrishnan, and D. Srivastav, "Combining Investment and Tax Strategies for Optimizing Lifetime Solvency under Uncertain Returns and Mortality," *Journal of Risk and Financial Management*, vol. 14, no. 7, p. 285, Jul. 2021, number: 7 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1911-8074/14/7/285

[31] S. Das, D. Ostrov, A. Radhakrishnan, and D. Srivastav, "Efficient Goal Probabilities: A New Frontier," *Journal of Investment Management*, vol. 21, no. 3, pp. 1–25, 2023. [Online]. Available: https://joim.com/efficient-goal-probabilities-a-new-frontier/