monteplan: A Vectorized Monte Carlo Framework for Multi-Account Retirement Planning with Tax-Aware Decumulation

Engineer Investor
egr.investor@gmail.com
https://github.com/engineerinvestor/monteplan

February 2026 — Version 0.5.0

Abstract

We present MONTEPLAN, an open-source Python framework for Monte Carlo simulation of multi-account retirement plans under realistic tax treatment. The framework addresses a critical gap in open-source financial planning tools: most existing solutions either lack multi-account modeling, ignore progressive tax brackets, or restrict returns to historical replay. MONTEPLAN provides four return models—multivariate normal, multivariate Student-t, Markov regime-switching, and historical block bootstrap—coupled with an Ornstein–Uhlenbeck inflation process. The engine operates on a fully vectorized three-dimensional tensor representation of positions (paths × accounts × assets), eliminating per-path Python loops to achieve high throughput. Tax-aware decumulation incorporates US federal progressive brackets with long-term capital gains rates, the Net Investment Income Tax surtax, state tax overlays, Required Minimum Distributions, and Roth conversion strategies (fixed-amount and fill-bracket). Five dynamic spending policies—constant real, percent-of-portfolio, Guyton–Klinger guardrails, variable percentage withdrawal, and floor-ceiling—provide flexible drawdown modeling. Antithetic variates offer variance reduction, and a sensitivity analysis framework enables one-at-a-time perturbation and two-dimensional grid searches. The framework is validated through golden test suites with exact bit-level reproducibility via the PCG64DXSM pseudorandom generator. All configuration is type-safe via Pydantic v2, and a Streamlit web application provides interactive exploration.

Contents

1 Introduction

Retirement planning under uncertainty is one of the most consequential financial decisions individuals face. The fundamental challenge lies in projecting portfolio performance, inflation, tax liability, and spending needs over horizons that may span 30–40 years, subject to stochastic market returns and evolving tax policy.

Deterministic planning tools—which assume fixed rates of return and inflation—produce point estimates that ignore the distribution of outcomes. Since Bengen's seminal work on safe withdrawal rates [2] and the Trinity Study [8], Monte Carlo simulation has become the standard methodology for quantifying the probability of portfolio survival under uncertainty.

However, most open-source Monte Carlo retirement tools suffer from one or more of the following limitations:

(i) **Single-account models** that ignore the tax-differential treatment of taxable, traditional (tax-deferred), and Roth (tax-free) accounts;

(ii) **Historical replay only**, lacking parametric models for regime-dependent behavior or fat-tailed returns;

(iii) **Flat-rate tax approximations** that miss the significant impact of progressive brackets, long-term capital gains rates, and surtaxes such as the Net Investment Income Tax;

(iv) **Static spending rules**, typically the "4% rule," without dynamic policies that adapt to portfolio performance.

This paper describes MONTEPLAN, an open-source Python framework (v0.5.0) that addresses all four limitations. The key contributions are:

1. A **vectorized multi-account simulation engine** operating on a three-dimensional numpy tensor of positions (paths × accounts × assets), with no per-path Python loops in the main simulation step;

2. **Four asset return models**: multivariate normal (MVN), multivariate Student-t, Markov regime-switching with $K \in [2, 5]$ regimes, and historical block bootstrap;

3. **Tax-aware decumulation** with US federal progressive brackets, long-term capital gains rates, the 3.8% Net Investment Income Tax, configurable state income tax, Required Minimum Distributions (RMDs), and Roth conversion strategies;

4. **Five dynamic spending policies**: constant real, percent-of-portfolio, Guyton–Klinger guardrails [11], variable percentage withdrawal (VPW), and floor-ceiling;

5. A **sensitivity analysis framework** supporting one-at-a-time perturbation with parallel execution and two-dimensional grid search;

6. A **safe withdrawal rate finder** using bisection search on the success probability function.

The remainder of this paper is organized as follows. Section 2 develops the mathematical framework for return models, inflation dynamics, tax computation, and spending policies. Section 3 describes the computational architecture, vectorization strategy, and variance reduction. Section 4 covers the configuration and validation system. Section 5 presents the analytics framework. Section 6 provides empirical validation and convergence analysis. Section 7 surveys related work, and Section 8 concludes.

2 Mathematical Framework

2.1 Asset Return Models

Let n denote the number of asset classes, T the number of monthly time steps, and M the number of Monte Carlo paths. The simulation pre-generates a return tensor $\boldsymbol{R} \in \mathbb{R}^{M \times T \times n}$ before entering the main loop. Each element $R_{m,t,i}$ represents the simple monthly return for path m, step t, and asset i.

2.1.1 Multivariate Normal Returns

Given annual expected returns $\boldsymbol{\mu}^{(a)} \in \mathbb{R}^n$ and annual volatilities $\boldsymbol{\sigma}^{(a)} \in \mathbb{R}^n_+$, the monthly parameters are:

$$\mu_i = \frac{\mu_i^{(a)}}{12}, \tag{1}$$

$$\sigma_i = \frac{\sigma_i^{(a)}}{\sqrt{12}}. \tag{2}$$

Given the $n \times n$ correlation matrix $\boldsymbol{\rho}$, the monthly covariance matrix is:

$$\boldsymbol{\Sigma} = \operatorname{diag}(\boldsymbol{\sigma})\, \boldsymbol{\rho}\, \operatorname{diag}(\boldsymbol{\sigma}). \tag{3}$$

Returns are drawn as:

$$\boldsymbol{R}_{m,t} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad m = 1, \ldots, M,\ t = 1, \ldots, T. \tag{4}$$

In practice, the implementation draws $M \times T$ samples from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and reshapes to (M, T, n).

2.1.2 Multivariate Student-t Returns

To capture the fat tails observed in empirical return distributions [9], we implement a multivariate Student-t model with $\nu > 2$ degrees of freedom.

Let $\boldsymbol{L}$ be the Cholesky factor of the correlation matrix $\boldsymbol{\rho} = \boldsymbol{L}\boldsymbol{L}^\top$. The sampling procedure is:

1. Draw $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_n)$;

2. Draw $W \sim \chi^2(\nu)$, independent of $\boldsymbol{Z}$;

3. Compute $\boldsymbol{X} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot (\boldsymbol{L}\boldsymbol{Z} \cdot \sqrt{\nu/W})$.

The resulting $\boldsymbol{X}$ follows a multivariate Student-t distribution with mean $\boldsymbol{\mu}$, scale $\boldsymbol{\sigma}$, correlation $\boldsymbol{\rho}$, and ν degrees of freedom. The scaling factor $\sqrt{\nu/W}$ inflates the tails: as $\nu \to \infty$, $\sqrt{\nu/W} \to 1$ and the distribution converges to the multivariate normal.

2.1.3 Markov Regime-Switching Returns

Following Hamilton [12], we model market regimes as a discrete-time Markov chain with K states ($K \in [2, 5]$). Each regime $k \in \{1, \ldots, K\}$ has its own return parameters:

$$\text{Regime } k: \quad \boldsymbol{\mu}_k \in \mathbb{R}^n,\ \boldsymbol{\sigma}_k \in \mathbb{R}^n_+,\ \boldsymbol{\rho}_k \in \mathbb{R}^{n \times n}. \tag{5}$$

The transition dynamics are governed by the row-stochastic matrix $\boldsymbol{P} \in \mathbb{R}^{K \times K}$, where $P_{ij} = \mathbb{P}(S_{t+1} = j \,|\, S_t = i)$ and $\sum_j P_{ij} = 1$ for all i.

Definition 2.1 (Regime-Switching Return Process)**.** Let S_t denote the regime at time t, with S_0 given. At each step t:

1. Observe regime $S_t = k$;

2. Draw $\boldsymbol{R}_t \sim \mathcal{N}(\boldsymbol{\mu}_k/12,\ \boldsymbol{\Sigma}_k)$ where $\boldsymbol{\Sigma}_k = \operatorname{diag}(\boldsymbol{\sigma}_k/\sqrt{12})\, \boldsymbol{\rho}_k\, \operatorname{diag}(\boldsymbol{\sigma}_k/\sqrt{12})$;

3. Transition: draw $U \sim \text{Uniform}(0, 1)$ and set $S_{t+1} = \min\{j : \sum_{l=1}^{j} P_{k,l} > U\}$.

The regime transition is implemented via cumulative probability thresholds:

$$S_{t+1} = \arg\min_j \left\{ \sum_{l=1}^{j} P_{S_t, l} > U_t \right\}, \quad U_t \sim \text{Uniform}(0, 1). \tag{6}$$

The implementation vectorizes across paths within each regime at each time step: for regime k, all paths where $S_t = k$ are gathered, their returns sampled simultaneously using $\boldsymbol{L}_k$ (the Cholesky factor of $\boldsymbol{\rho}_k$), and scattered back.

2.1.4 Historical Block Bootstrap

The block bootstrap [6, 18] provides a non-parametric alternative that preserves short-range autocorrelation. Given a historical return matrix $\boldsymbol{H} \in \mathbb{R}^{N \times n}$ of N monthly observations, and a block size B (default $B = 12$), the procedure is:

1. Compute the number of blocks needed: $n_b = \lceil T/B \rceil$;

2. For each path m, draw n_b starting indices independently: $s_{m,j} \sim \text{Uniform}\{0, 1, \ldots, N - B\}$, $j = 1, \ldots, n_b$;

3. Concatenate blocks: $\boldsymbol{R}_m = [\boldsymbol{H}_{s_{m,1}:s_{m,1}+B},\ \boldsymbol{H}_{s_{m,2}:s_{m,2}+B},\ \ldots]$;

4. Truncate to T steps.

The implementation is fully vectorized via NumPy fancy indexing: block start indices are expanded into a (M, n_b, B) index tensor, which is flattened and used to gather from $\boldsymbol{H}$ in a single operation.

Remark 2.2. Block size B controls the trade-off between preserving autocorrelation (larger B) and sampling diversity (smaller B). With $B = 12$, one-year seasonal patterns are preserved, which is appropriate for monthly financial data.

2.2 Inflation Dynamics

2.2.1 Ornstein–Uhlenbeck Process

Inflation is modeled as a mean-reverting Ornstein–Uhlenbeck (OU) process [20, 22]:

$$dI_t = \kappa(\theta - I_t)\, dt + \sigma_I\, dW_t, \tag{7}$$

where I_t is the annualized inflation rate, θ is the long-run mean (default 3%), κ is the mean-reversion speed (default 0.5), σ_I is the volatility (default 1%), and W_t is a standard Brownian motion.

The Euler–Maruyama discretization with monthly time step $\Delta t = 1/12$ is:

$$I_{t+1} = I_t + \kappa(\theta - I_t)\Delta t + \sigma_I \sqrt{\Delta t}\, Z_t, \quad Z_t \sim \mathcal{N}(0, 1). \tag{8}$$

The monthly inflation rate applied to nominal values is $I_t/12$. The cumulative inflation factor at step t is:

$$\Pi_t = \prod_{s=0}^{t-1} (1 + I_s/12). \tag{9}$$

2.2.2 Regime-Coupled Inflation

When regime-switching returns (Section 2.1.3) are active, inflation parameters are coupled to the market regime. Each regime k specifies its own $(\theta_k, \sigma_{I,k})$, while κ is shared:

$$I_{t+1} = I_t + \kappa(\theta_{S_t} - I_t)\Delta t + \sigma_{I,S_t} \sqrt{\Delta t}\, Z_t. \tag{10}$$

This couples inflation to market conditions: a "bear" regime may feature higher inflation mean and volatility, reflecting stagflation scenarios [1, 10].

2.3 Tax Models

The framework provides two tax model implementations: a flat-rate approximation and a full US federal progressive model.

2.3.1 Flat-Rate Model

For quick estimation:

$$\text{Tax} = r \cdot (\text{ordinary income} + \text{LTCG}), \tag{11}$$

where r is a configurable effective rate (default 22%).

2.3.2 US Federal Progressive Tax

The US federal model implements the 2024 tax year brackets from IRS Revenue Procedure 2023-34 [15].

Ordinary Income. Let I be gross ordinary income and D the standard deduction. Taxable ordinary income is $\bar{I} = \max(I - D, 0)$. Given B brackets with thresholds $\{L_1 = 0 < L_2 < \cdots < L_B < L_{B+1} = \infty\}$ and marginal rates $\{r_1, r_2, \ldots, r_B\}$, the ordinary tax is:

$$\text{Tax}_{\text{ord}}(\bar{I}) = \sum_{b=1}^{B} r_b \cdot \min\big(\max(\bar{I} - L_b, 0),\ L_{b+1} - L_b\big). \tag{12}$$

For the 2024 single filer, the brackets and standard deduction are given in Table 6.

Long-Term Capital Gains. LTCG receives preferential rates via a separate bracket schedule:

$$\text{Tax}_{\text{LTCG}}(G) = \sum_{b=1}^{B'} r'_b \cdot \min\big(\max(G - L'_b, 0),\ L'_{b+1} - L'_b\big). \tag{13}$$

For single filers (2024): 0% up to \$47,025; 15% up to \$518,900; 20% above.

Net Investment Income Tax (NIIT). The 3.8% surtax on net investment income applies when modified adjusted gross income (MAGI) exceeds a threshold τ:

$$\text{NIIT} = 0.038 \cdot \min\big(G,\ \max(\text{MAGI} - \tau, 0)\big), \tag{14}$$

where $\text{MAGI} = I + G$ and τ depends on filing status (\$200,000 single, \$250,000 married filing jointly).

State Income Tax. A flat state overlay is applied on total income:

$$\text{Tax}_{\text{state}} = r_s \cdot (I + G), \quad r_s \in [0, 0.15]. \tag{15}$$

Total Annual Tax.

$$\text{Tax}_{\text{total}} = \text{Tax}_{\text{ord}}(\bar{I}) + \text{Tax}_{\text{LTCG}}(G) + \text{NIIT} + \text{Tax}_{\text{state}}. \tag{16}$$

Vectorized Implementation. The bracket computation is vectorized across all M paths simultaneously. For each bracket b, the taxable amount in that bracket is computed as a single NumPy array operation on the $(M,)$-dimensional income vector, avoiding any per-path Python loop.

Tax Application to Positions. Year-end tax liability is deducted proportionally from all positions:

$$\boldsymbol{P}_{m,a,i} \leftarrow \boldsymbol{P}_{m,a,i} \cdot \Big(1 - \frac{\text{Tax}_m}{\max(W_m, 1)}\Big), \tag{17}$$

where $W_m = \sum_{a,i} P_{m,a,i}$ is total wealth for path m.

2.4 Required Minimum Distributions

For tax-deferred accounts (traditional IRA/401k), the IRS mandates RMDs beginning at age 73 [16]. The annual RMD is:

$$\text{RMD}(\text{age}) = \frac{B_{\text{prior year}}}{\delta(\text{age})}, \tag{18}$$

where $B_{\text{prior year}}$ is the December 31 balance of the prior year and $\delta(\text{age})$ is the divisor from the IRS Uniform Lifetime Table (Table 8).

In the engine, RMDs are enforced at the December step (month 12). Any traditional account withdrawals made during the year for spending count toward RMD satisfaction. If a shortfall remains, forced withdrawals are executed proportionally from traditional account positions.

2.5 Spending Policies

All spending policies implement a common protocol:

$$s_t = f(\text{state}_t) \in \mathbb{R}_+^M, \tag{19}$$

returning a vector of monthly spending amounts across all M paths.

2.5.1 Constant Real

The simplest policy maintains purchasing power:

$$s_t = s_0 \cdot \Pi_t, \tag{20}$$

where s_0 is the base monthly spending in today's dollars and Π_t is the cumulative inflation factor from Eq. (9).

2.5.2 Percent of Portfolio

Spending is a fixed fraction of current wealth:

$$s_t = \frac{r}{12} \cdot \max(W_t, 0), \tag{21}$$

where r is the annual withdrawal rate (e.g., 0.04).

2.5.3 Guyton–Klinger Guardrails

The guardrail policy [11] adjusts spending based on the current withdrawal rate relative to the initial rate r_0:

Definition 2.3 (Guardrail Decision Rules)**.** Let $\hat{r}_t = 12s_t/W_t$ be the current annualized withdrawal rate.

- **Prosperity Rule:** If $\hat{r}_t < r_0(1 - \alpha_u)$, set $s_{t+1} = s_t(1 + \gamma_u)$.
- **Capital Preservation Rule:** If $\hat{r}_t > r_0(1 + \alpha_l)$, set $s_{t+1} = s_t(1 - \gamma_l)$.

where α_u, α_l are threshold parameters (default 20%) and γ_u, γ_l are adjustment magnitudes (default 10%).

The initial spending at the start of retirement is:

$$s_{\text{retire}} = \frac{r_0}{12} \cdot W_{\text{retire}}. \tag{22}$$

2.5.4 Variable Percentage Withdrawal (VPW)

VPW [5] sets the withdrawal rate inversely proportional to the remaining planning horizon:

$$r_t = \text{clamp}\left(\frac{1}{T_{\text{end}} - a_t},\ r_{\min},\ r_{\max}\right), \quad s_t = \frac{r_t}{12} \cdot \max(W_t, 0), \tag{23}$$

where a_t is the current age, T_{end} is the planning end age, and $r_{\min} = 0.03$, $r_{\max} = 0.15$ are bounds.

2.5.5 Floor-Ceiling

The floor-ceiling policy [21] clamps portfolio-proportional spending between inflation-adjusted bounds:

$$s_t = \text{clamp}\left(\frac{r}{12} \cdot W_t,\ F \cdot \Pi_t,\ C \cdot \Pi_t\right), \tag{24}$$

where F is the monthly floor (default \$3,000), C is the monthly ceiling (default \$10,000), and both are in today's dollars.

2.6 Roth Conversion Strategies

Roth conversions move balances from traditional (tax-deferred) accounts to Roth (tax-free) accounts, triggering ordinary income tax on the converted amount. Two strategies are supported:

Fixed Amount. Convert a constant annual amount A during the conversion window $[a_{\text{start}}, a_{\text{end}})$:

$$C_t = \min(A,\ B_{\text{trad},t}), \tag{25}$$

where $B_{\text{trad},t}$ is the available traditional balance. This strategy is straightforward but does not adapt to the individual's current tax situation.

Fill-Bracket. Convert up to the ceiling of a target tax bracket, minus current ordinary income:

$$C_t = \min\big(\max(\text{ceiling}(r^*) - I_t, 0),\ B_{\text{trad},t}\big), \tag{26}$$

where $\text{ceiling}(r^*)$ is the income level at which the marginal rate exceeds the target rate r^*, and I_t is the year-to-date ordinary income (including RMDs and other withdrawals).

The bracket ceiling function maps a target marginal rate to the corresponding income threshold. For the 2024 single filer with $r^* = 0.22$, the ceiling is the top of the 22% bracket ($100,525) plus the standard deduction ($14,600), yielding $115,125 in gross income.

Conversion Mechanics. Conversions are executed at year-end (step where month = 12), after RMD enforcement and before annual tax computation. The converted amount is added to that year's taxable ordinary income. The transfer is performed pro-rata across assets within each traditional account:

$$P_{m,\text{trad},i} \leftarrow P_{m,\text{trad},i} \cdot \left(1 - \frac{C_m}{B_{m,\text{trad}}}\right), \quad P_{m,\text{roth},i} \mathrel{+}= P_{m,\text{trad},i} \cdot \frac{C_m}{B_{m,\text{trad}}}, \tag{27}$$

preserving the asset allocation of the converted positions.

Tax Implications. Roth conversions create a short-term tax cost (higher ordinary income in the conversion year) in exchange for a long-term benefit (tax-free growth and withdrawals). The fill-bracket strategy is particularly effective during the "gap years" between retirement and RMD onset (ages ∼60–72), when ordinary income may be low, allowing conversions at favorable marginal rates.

Interaction with RMDs. Since conversions execute after RMD satisfaction, the year-to-date income I_t already includes any RMD withdrawals. This ensures the fill-bracket strategy correctly accounts for the RMD's tax impact when computing the remaining bracket headroom.

3 Computational Architecture

3.1 Tensor Representation

The engine maintains portfolio state as a three-dimensional NumPy [14] array:

$$\boldsymbol{P} \in \mathbb{R}^{M \times A \times n}, \tag{28}$$

where M is the number of Monte Carlo paths, A is the number of accounts (e.g., taxable, traditional, Roth), and n is the number of asset classes. Each element $P_{m,a,i}$ represents the dollar value of asset i in account a for path m.

Derived quantities are computed via axis reductions:

$$\text{Balance}_{m,a} = \sum_{i=1}^{n} P_{m,a,i} \quad \text{(account balances, shape } (M, A)\text{)}, \tag{29}$$

$$W_m = \sum_{a=1}^{A} \sum_{i=1}^{n} P_{m,a,i} \quad \text{(total wealth, shape } (M,)\text{)}. \tag{30}$$

Returns are pre-generated as:

$$\boldsymbol{R} \in \mathbb{R}^{M \times T \times n}, \tag{31}$$

and inflation rates as $\boldsymbol{\pi} \in \mathbb{R}^{M \times T}$.

Remark 3.1 (Memory Considerations)**.** For $M = 20{,}000$ paths, $A = 3$ accounts, $n = 10$ assets, and $T = 480$ steps (40 years), the position tensor requires $20{,}000 \times 3 \times 10 \times 8 \approx 4.6$ MB, and the return tensor requires $20{,}000 \times 480 \times 10 \times 8 \approx 732$ MB. The return tensor dominates memory usage.

3.2 Engine Pipeline

The simulation loop executes the following twelve-step pipeline at each monthly time step $t = 0, 1, \ldots, T-1$.

Algorithm 1: Monthly Simulation Step

Input: State $(\boldsymbol{P}, \Pi, \boldsymbol{d})$, returns $\boldsymbol{R}_{:,t,:}$, inflation $\boldsymbol{\pi}_{:,t}$, timeline, policies

▷ `Step 1: Apply asset returns`

$\boldsymbol{P}_{m,a,i} \leftarrow \boldsymbol{P}_{m,a,i} \cdot (1 + R_{m,t,i}) \quad \forall m, a, i$

$\boldsymbol{P} \leftarrow \max(\boldsymbol{P}, 0)$

▷ `Step 1b: Investment fee drag`

$f \leftarrow (\text{expense\_ratio} + \text{aum\_fee} + \text{advisory\_fee})/12$

$\boldsymbol{P} \leftarrow \boldsymbol{P} \cdot (1 - f)$

▷ `Step 2: Accumulate inflation`

$\Pi_m \leftarrow \Pi_m \cdot (1 + \pi_{m,t}) \quad \forall m$

▷ `Step 2a: Compute target weights (static or glide path)`

$\boldsymbol{w} \leftarrow \text{target\_weights}(t)$

▷ `Step 3: Contributions (pre-retirement)`

if $t < t_{income\_end}$ **then**

  $\boldsymbol{P}_{m,a,i} \mathrel{+}= c_a \cdot g_t \cdot w_i \quad \forall m, a, i$

▷ `Step 4: Discrete events`

foreach *event* e *at step* t **do**

  Apply inflow or withdraw outflow

▷ `Step 5: Rebalancing`

if *threshold triggered* ***or*** *calendar month* **then**

  Rebalance positions to target weights $\boldsymbol{w}$

▷ `Step 6: Retirement spending & withdrawals`

if $t \geq t_{retire}$ **then**

  $\boldsymbol{s} \leftarrow \text{spending\_policy.compute(state)}$

  Subtract guaranteed income from $\boldsymbol{s}$

  Execute withdrawals in account priority order

  Track traditional withdrawals for tax

▷ `Step 6b: RMD enforcement (December, age` ≥ 73`)`

if *month* $= 12$ ***and*** *age* ≥ 73 **then**

  Force withdraw any RMD shortfall from traditional accounts

▷ `Step 6c: Roth conversions (December, within window)`

if *Roth enabled* ***and*** *month* $= 12$ ***and*** *in window* **then**

  Compute conversion amount (fixed or fill-bracket)

  Move positions from traditional to Roth, pro-rata

  Add to ordinary income for tax

▷ `Step 7: Year-end tax computation (December)`

if *month* $= 12$ **then**

  $\boldsymbol{\tau} \leftarrow \text{tax\_model}(\text{income}, \text{LTCG})$

  $\boldsymbol{\tau} \mathrel{+}= \text{state\_tax} + \text{NIIT}$

Key vectorization patterns used throughout the pipeline:

Broadcasting returns across accounts. Asset returns have shape (M, n) and positions have shape (M, A, n). Broadcasting is achieved by inserting a size-1 axis:

$$\boldsymbol{P} \leftarrow \boldsymbol{P} \odot (1 + \boldsymbol{R}_{:,t,:})[:, \texttt{newaxis}, :]. \tag{32}$$

Proportional tax deduction. Tax has shape $(M,)$, requiring two axis insertions:

$$\boldsymbol{P} \leftarrow \boldsymbol{P} \odot (1 - \boldsymbol{\tau} / \max(\boldsymbol{W}, 1))[:, \texttt{newaxis}, \texttt{newaxis}]. \tag{33}$$

Masked withdrawal. For withdrawal from a single account a by amount $\boldsymbol{v} \in \mathbb{R}^M$:

$$\text{frac}_m = \min(v_m / \max(B_{m,a}, 1), 1), \quad P_{m,a,i} \leftarrow P_{m,a,i} \cdot (1 - \text{frac}_m) \quad \forall i. \tag{34}$$

3.3 Variance Reduction via Antithetic Variates

Antithetic variates [9, 13] reduce variance by pairing each random draw with its reflection about the mean.

Proposition 3.2 (Antithetic Variance Reduction)**.** *Let $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and define $\tilde{\boldsymbol{X}} = 2\boldsymbol{\mu} - \boldsymbol{X}$. Then $\tilde{\boldsymbol{X}}$ has the same distribution as $\boldsymbol{X}$, and for any function g such that $\mathbb{E}[g(\boldsymbol{X})^2] < \infty$:*

$$\text{Var}\left(\frac{g(\boldsymbol{X}) + g(\tilde{\boldsymbol{X}})}{2}\right) \leq \text{Var}(g(\boldsymbol{X})). \tag{35}$$

Equality holds if and only if $\text{Cov}(g(\boldsymbol{X}), g(\tilde{\boldsymbol{X}})) = 0$. For monotone g, the covariance is negative, yielding strict improvement.

Proof. Since $\boldsymbol{X} - \boldsymbol{\mu}$ and $\boldsymbol{\mu} - \tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{\mu}$, the pair $(\boldsymbol{X}, \tilde{\boldsymbol{X}})$ is identically distributed, so $\mathbb{E}[g(\boldsymbol{X})] = \mathbb{E}[g(\tilde{\boldsymbol{X}})]$. For the average estimator:

$$\text{Var}\left(\frac{g(\boldsymbol{X}) + g(\tilde{\boldsymbol{X}})}{2}\right) = \frac{1}{4}\left[\text{Var}(g(\boldsymbol{X})) + \text{Var}(g(\tilde{\boldsymbol{X}})) + 2\,\text{Cov}(g(\boldsymbol{X}), g(\tilde{\boldsymbol{X}}))\right] \tag{36}$$

$$= \frac{1}{2}\,\text{Var}(g(\boldsymbol{X})) + \frac{1}{2}\,\text{Cov}(g(\boldsymbol{X}), g(\tilde{\boldsymbol{X}})). \tag{37}$$

Since $\text{Cov}(g(\boldsymbol{X}), g(\tilde{\boldsymbol{X}})) \leq 0$ for monotone g (because $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$ are negatively correlated), the result follows. □

Implementation. When antithetic mode is enabled, M must be even. The engine generates $M/2$ "base" return paths and constructs the antithetic paths by reflection:

$$\boldsymbol{R}^{\text{anti}}_{m,t,i} = 2\mu_i - \boldsymbol{R}^{\text{base}}_{m,t,i}, \quad m = 1, \ldots, M/2. \tag{38}$$

For the OU inflation process, independent antithetic noise is used: $Z^{\text{anti}} = -Z^{\text{base}}$. For regime-switching, the antithetic paths share the same regime sequence as their base counterparts.

3.4 Withdrawal Logic

Withdrawals follow a configurable account priority order (default: taxable → traditional → Roth), implementing the standard tax-efficient withdrawal sequencing strategy.

Definition 3.3 (Sequential Withdrawal)**.** Given a spending need v_m for path m and account priority order $(a_1, a_2, \ldots, a_A)$, the withdrawal from each account is:

$$w_{m,a_1} = \min(v_m,\ B_{m,a_1}), \tag{39}$$

$$w_{m,a_k} = \min\left(v_m - \sum_{j=1}^{k-1} w_{m,a_j},\ B_{m,a_k}\right), \quad k = 2, \ldots, A. \tag{40}$$

Each withdrawal is executed by reducing positions proportionally across all assets within the account, preserving the current asset allocation:

$$P_{m,a,i} \leftarrow P_{m,a,i} \cdot \left(1 - \frac{w_{m,a}}{\max(B_{m,a}, 1)}\right) \quad \forall i \in \{1, \ldots, n\}. \tag{41}$$

Tax Tracking. Withdrawals from traditional (tax-deferred) accounts generate ordinary income. The engine tracks cumulative traditional withdrawals during each calendar year for three purposes:

1. Year-end federal income tax computation;
2. RMD satisfaction tracking;
3. Roth conversion bracket headroom calculation.

Withdrawals from Roth accounts are tax-free, and withdrawals from taxable accounts generate long-term capital gains (tracked separately).

3.5 Rebalancing

Two rebalancing strategies are supported:

Calendar Rebalancing. Positions are rebalanced to target weights at fixed calendar months (default: January and July):

$$\text{Rebalance if } (t \bmod 12) + 1 \in \mathcal{M}_{\text{rebal}}, \tag{42}$$

where $\mathcal{M}_{\text{rebal}} = \{1, 7\}$ by default.

Threshold Rebalancing. Rebalancing is triggered only when the maximum absolute drift from target weights exceeds a threshold τ (default 5%):

$$\text{Rebalance path } m \text{ if } \max_{a,i} \left| \frac{P_{m,a,i}}{B_{m,a}} - w_i \right| > \tau. \tag{43}$$

In practice, the drift is computed at the portfolio level. Let $\hat{w}_i = \sum_a P_{m,a,i}/W_m$ be the current portfolio-level weight for asset i. The L^∞ drift is:

$$d_m = \max_i |\hat{w}_i - w_i|. \tag{44}$$

Threshold rebalancing reduces transaction costs and is empirically shown to produce comparable or superior results to frequent calendar rebalancing.

Rebalancing Execution. For each account a and each path m requiring rebalancing:

$$P_{m,a,i} \leftarrow B_{m,a} \cdot w_i \quad \forall i, \tag{45}$$

i.e., each account is set to hold the target weights proportional to its current balance.

3.6 Glide Path

A glide path shifts the target asset allocation over time, typically from equity-heavy (accumulation) to bond-heavy (decumulation). The implementation uses linear interpolation between start and end weights:

Definition 3.4 (Linear Glide Path)**.** Given start weights $\boldsymbol{w}_s$ at age a_s and end weights $\boldsymbol{w}_e$ at age a_e, the target weights at age a are:

$$\boldsymbol{w}(a) = \begin{cases} \boldsymbol{w}_s & \text{if } a \le a_s, \\ \boldsymbol{w}_s + \frac{a - a_s}{a_e - a_s}(\boldsymbol{w}_e - \boldsymbol{w}_s) & \text{if } a_s < a < a_e, \\ \boldsymbol{w}_e & \text{if } a \ge a_e. \end{cases} \tag{46}$$

A common glide path shifts from 80/20 stocks/bonds at age 40 to 40/60 at age 70, reducing equity exposure by approximately 1.3 percentage points per year.

3.7 Guaranteed Income

Guaranteed income streams (Social Security, pensions, annuities) reduce the portfolio withdrawal need during retirement.

Definition 3.5 (Guaranteed Income Stream)**.** Each stream j is characterized by:

- Monthly benefit g_j in today's dollars;

- Start age a_j^{start} and optional end age a_j^{end} (default: lifetime);
- Annual COLA rate c_j (default: 0%).

The COLA rate is converted to a monthly compounding factor:

$$c_j^{(\text{mo})} = (1 + c_j)^{1/12} - 1. \tag{47}$$

At each retirement step t, the net spending need after guaranteed income is:

$$s'_t = \max\left(s_t - \sum_{j \in \mathcal{G}_t} g_j \cdot (1 + c_j^{(\text{mo})})^{t - t_j^{\text{start}}} \cdot \Pi_t,\ 0 \right), \tag{48}$$

where $\mathcal{G}_t$ is the set of active streams at step t and t_j^{start} is the step corresponding to a_j^{start}.

Common Streams. Typical guaranteed income streams include:

- **Social Security**: starts at ages 62–70, with COLA typically 2–3%;
- **Pension**: employer-defined benefit, may or may not include COLA;
- **Annuity**: purchased income, often with no COLA (fixed nominal).

The presence of substantial guaranteed income significantly impacts portfolio withdrawal rates: a retiree with \$3,000/month in Social Security needs far less from the portfolio than one without, enabling more aggressive asset allocation and higher success probabilities.

3.8 Investment Fees

Three fee components are modeled as annual rates, converted to monthly drag on positions:

Definition 3.6 (Fee Structure)**.**

- **Expense Ratio** (f_e): Fund-level fees (e.g., 0.03% for index funds, 0.5–1% for active);
- **AUM Fee** (f_a): Platform or custodian fee on assets under management;
- **Advisory Fee** (f_v): Financial advisor fee.

The combined monthly drag is:

$$f_{\text{mo}} = \frac{f_e + f_a + f_v}{12}, \quad \boldsymbol{P} \leftarrow \boldsymbol{P} \cdot (1 - f_{\text{mo}}). \tag{49}$$

Remark 3.7 (Impact of Fees)**.** A seemingly small 1% annual fee compounds significantly over a 30-year retirement. With no fees, a \$1M portfolio earning 7% grows to \$7.6M over 30 years. With 1% in total fees (net 6%), it reaches only \$5.7M—a 25% reduction in terminal wealth. The engine applies fees monthly to capture this compounding effect accurately.

3.9 Complexity Analysis

Table 1: Computational complexity of major operations.

Operation	Complexity per step
Return application	$O(M \cdot A \cdot n)$
Inflation update	$O(M)$
Rebalancing	$O(M \cdot A \cdot n)$
Spending computation	$O(M)$
Withdrawal execution	$O(M \cdot A \cdot n)$
Tax computation	$O(M \cdot B)$ where B = brackets
Full simulation	$O(T \cdot M \cdot A \cdot n)$
OAT sensitivity	$O(2P \cdot T \cdot M \cdot A \cdot n)$
SWR bisection	$O(\log(\Delta/\epsilon) \cdot T \cdot M \cdot A \cdot n)$

where P is the number of perturbable parameters, Δ is the search range, and ϵ is the convergence tolerance.

4 Configuration and Validation System

4.1 Pydantic v2 Config Models

All configuration is managed through Pydantic v2 [7] `BaseModel` classes with strict validation:

- **PlanConfig**: Demographics (ages), account definitions, income, spending goals, discrete events, guaranteed income streams.
- **MarketAssumptions**: Asset classes, returns, volatilities, correlation matrix, inflation parameters, return model selection, fee structure, glide path.
- **PolicyBundle**: Spending policy, rebalancing strategy, withdrawal order, tax model, state tax rate, NIIT flag, Roth conversion settings.
- **SimulationConfig**: Path count, seed, antithetic flag, preset selection, stress scenarios.

Cross-Field Validation. Pydantic model validators enforce:

1. Correlation matrix symmetry: $\rho_{ij} = \rho_{ji}$, with tolerance 10^{-8};
2. Diagonal unity: $\rho_{ii} = 1$ for all i;
3. Positive semi-definiteness (implicitly via Cholesky in return models);
4. Transition matrix row-stochasticity: $|\sum_j P_{ij} - 1| < 10^{-6}$;

5. Age ordering: current_age < retirement_age < end_age;

6. Target weight normalization: $|\sum_i w_i - 1| < 10^{-6}$.

4.2 Reproducibility

Deterministic RNG. The engine uses NumPy's PCG64DXSM [17] generator, seeded from the user-provided seed:

```
rng = numpy.random.Generator(numpy.random.PCG64DXSM(seed))
```

PCG64DXSM provides a period of 2^{128}, excellent equidistribution, and fast generation. The same seed produces bit-identical return tensors across runs, ensuring perfect reproducibility.

Config Hashing. Every simulation result embeds a SHA-256 hash of the serialized configuration, enabling provenance tracking:

$$h = \text{SHA-256}\big(\text{JSON}(\text{plan}, \text{market}, \text{policies}, \text{sim\_config})\big). \tag{50}$$

5 Analytics Framework

5.1 Sensitivity Analysis

5.1.1 One-at-a-Time (OAT) Perturbation

The OAT method varies one parameter at a time while holding all others at their baseline values. For each parameter p_i with baseline value v_i:

1. Compute perturbed values:

$$v_i^- = v_i(1-\delta), \quad v_i^+ = v_i(1+\delta), \tag{51}$$

   where $\delta = 0.10$ (10%) for multiplicative parameters. Additive parameters (retirement age, state tax rate) use fixed deltas.

2. Run simulations at v_i^- and v_i^+;

3. Compute impact: $\Delta_i = p_{\text{success}}(v_i^+) - p_{\text{success}}(v_i^-)$.

   The results are sorted by $|\Delta_i|$ to produce a tornado chart.

Parallel Execution. Perturbation runs are independent and execute via Python's `ProcessPoolExecutor` with configurable `max_workers`. Path counts are capped at 2,000 for OAT analysis to balance speed and precision.

5.1.2 Two-Dimensional Grid Search

For interaction analysis, a 2D grid evaluates success probability over parameter pairs:

$$\text{grid}[i, j] = p_{\text{success}}(x_i, y_j), \quad x_i \in \{x_1, \ldots, x_{n_x}\},\ y_j \in \{y_1, \ldots, y_{n_y}\}. \tag{52}$$

Grid values are generated via `numpy.linspace`. Path counts are capped at 1,000 per cell.

5.2 Safe Withdrawal Rate Finder

The safe withdrawal rate (SWR) is the maximum monthly spending s^* such that the success probability meets a target p^* (default 95%):

$$s^* = \max\{s : p_{\text{success}}(s) \geq p^*\}. \tag{53}$$

Proposition 5.1 (Monotonicity)**.** *Under standard assumptions (non-negative returns and spending), the success probability $p_{success}(s)$ is monotonically non-increasing in the spending level s.*

This monotonicity guarantees that bisection converges. The algorithm searches $[s_{\min}, s_{\max}]$ with tolerance ϵ (default \$50):

Algorithm 2: Bisection SWR Finder

Input: $s_{\min}, s_{\max}, p^*, \epsilon, N_{\max}$
Output: s^* (safe monthly spending)

```
lo ← s_min;
hi ← s_max;
for k = 1 to N_max do
    mid ← (lo + hi)/2
    p ← simulate(spending = mid)
    if p ≥ p* then
        lo ← mid
    else
        hi ← mid
    if hi − lo < ε then break
return lo
```

The number of iterations is bounded by $\lceil \log_2((s_{\max} - s_{\min})/\epsilon) \rceil$. With default parameters ($s_{\max} = 2s_0$, $\epsilon = \$50$), this is at most $\lceil \log_2(2s_0/50) \rceil$ iterations.

Output. The SWR finder returns:

- Maximum safe monthly spending s^*;
- Implied annual withdrawal rate: $r^* = 12s^* / \sum_a B_a^{(0)}$;

- Achieved success probability at s^*;
- Number of bisection iterations.

5.3 Distributional Metrics

5.3.1 Success Probability and Standard Error

The primary output metric is the fraction of paths with non-zero wealth throughout retirement:

$$\hat{p} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}\left[\min_{t \geq t_{\text{retire}}} W_{m,t} > 0\right]. \tag{54}$$

The Monte Carlo standard error is:

$$\text{SE}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{M}}. \tag{55}$$

For $M = 5{,}000$ and $\hat{p} = 0.50$ (worst case), SE $\approx 0.71\%$. The "deep" preset ($M = 20{,}000$ with antithetic) achieves SE $\approx 0.25\%$ (effective due to variance reduction).

5.3.2 Maximum Drawdown Distribution

For each path m, the maximum drawdown is:

$$\text{MDD}_m = \max_{0 \leq t \leq T} \frac{\bar{W}_{m,t} - W_{m,t}}{\max(\bar{W}_{m,t}, 1)}, \tag{56}$$

where $\bar{W}_{m,t} = \max_{0 \leq s \leq t} W_{m,s}$ is the running maximum. The distribution $\{\text{MDD}_m\}_{m=1}^{M}$ is summarized by percentiles (5th, 25th, 50th, 75th, 95th) and mean.

5.3.3 Spending Volatility

Monthly spending volatility during retirement is measured by the coefficient of variation of spending changes:

$$\text{CV}_m = \frac{\text{sd}(\{s_{m,t+1} - s_{m,t}\}_{t \geq t_{\text{retire}}})}{\max(\bar{s}_m, 1)}, \tag{57}$$

where $\bar{s}_m$ is the mean monthly spending for path m.

5.3.4 Ruin-by-Age Curve

The cumulative ruin curve gives the fraction of paths depleted by each age:

$$F(\text{age}) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}[\exists\, t \leq t(\text{age}) : W_{m,t} \leq 0]. \tag{58}$$

This is computed efficiently via cumulative logical-OR on the depletion indicator.

6 Empirical Validation

6.1 Golden Test Suite

The framework maintains a suite of golden tests that verify exact numerical outputs for fixed seeds. These tests detect any unintentional changes to the simulation engine, serving as regression guards.

Table 2: Golden test values (seed=42, 5,000 paths).

Scenario	Return Model	Success Probability
Basic retirement (65–95)	MVN	47.94%
Regime-switching (3 regimes)	Markov	32.82%
Antithetic variates	MVN (antithetic)	49.90%

These values are verified to exact floating-point precision across Python 3.11, 3.12, and 3.13 on the CI matrix. Any change to the engine that alters these values requires explicit golden value updates with justification.

Interpretation of Golden Values. The basic retirement scenario (47.94% success) reflects a moderately aggressive plan: a 30-year horizon from age 65 to 95 with default market assumptions (stocks: 7% return, 15% volatility; bonds: 3% return, 5% volatility). The sub-50% success rate indicates the default spending level is slightly above the safe withdrawal rate for this asset allocation.

The regime-switching scenario (32.82%) shows substantially lower success due to the inclusion of a "bear" regime with negative expected returns and elevated volatility. This demonstrates the framework's ability to model adverse market environments that are not captured by MVN.

The antithetic result (49.90%) is close to the non-antithetic MVN result, as expected—antithetic variates reduce variance of the estimator but do not change its expected value.

6.2 Convergence Analysis

6.2.1 Path Count vs. Precision

The Monte Carlo standard error (Eq. (55)) provides a theoretical bound on the precision of $\hat{p}$. Table 3 shows the standard error at $\hat{p} = 0.50$ for each preset.

Table 3: Simulation presets and standard errors.

Preset	Paths	Antithetic	SE at $p = 0.50$	95% CI width
Fast	1,000	No	1.58%	±3.10%
Balanced	5,000	No	0.71%	±1.39%
Deep	20,000	Yes	$\leq 0.25\%$*	$\leq \pm 0.50\%$

*Effective SE is lower due to negative covariance from antithetic pairing. 95% CI $= \hat{p} \pm 1.96 \cdot \text{SE}$.

The "balanced" preset (5,000 paths) provides sub-1% standard error, which is sufficient for most planning decisions. The "deep" preset (20,000 paths with antithetic) achieves precision suitable for research-grade analysis.

6.2.2 Convergence Rate

The standard Monte Carlo convergence rate is $O(1/\sqrt{M})$. To halve the standard error, one must quadruple the number of paths. This fundamental relationship motivates the three presets:

$$\text{SE}(M) = \frac{c}{\sqrt{M}}, \quad c = \sqrt{p(1-p)} \leq 0.5. \tag{59}$$

For $p = 0.50$ (worst case): SE(1000) = 1.58%, SE(5000) = 0.71%, SE(20000) = 0.35% (before antithetic improvement).

6.2.3 Antithetic Variance Reduction Factor

For the basic retirement scenario, the empirical variance reduction factor of antithetic variates is approximately 1.8–2.1×, consistent with the theoretical prediction for monotone portfolio value functions (Proposition 3.2).

The variance reduction factor ρ is defined as:

$$\rho = \frac{\text{Var}(\hat{p}_{\text{standard}})}{\text{Var}(\hat{p}_{\text{antithetic}})}, \tag{60}$$

where both estimators use the same total number of simulation paths. A factor of 2.0 means the antithetic estimator with M paths achieves the same precision as a standard estimator with $2M$ paths.

Remark 6.1. The variance reduction is most effective when the portfolio value function is approximately monotone in the return draws. For highly non-linear strategies (e.g., guardrails with extreme thresholds), the reduction factor may be closer to 1.5×.

6.3 Tax Model Verification

The US federal tax model is verified against manual bracket computations derived from IRS Revenue Procedure 2023-34 [15]:

- For single filer with \$100,000 ordinary income (2024): taxable = \$100,000 − \$14,600 = \$85,400. Tax = $0.10 \times 11{,}600 + 0.12 \times (47{,}150 - 11{,}600) + 0.22 \times (85{,}400 - 47{,}150)$ = \$13,483.
- NIIT for single filer with \$150,000 ordinary + \$100,000 LTCG: MAGI = \$250,000; excess = \$50,000; base = $\min(100{,}000, 50{,}000)$ = \$50,000; NIIT = $0.038 \times 50{,}000$ = \$1,900.
- Bracket ceiling at 22% marginal rate (single): ceiling = \$100,525 + \$14,600 = \$115,125.

These computations are encoded as parameterized test cases in the test suite.
Additional tax verification includes:

- **Zero-income test**: Tax on \$0 income returns \$0 for both flat and progressive models;
- **Deduction-only test**: Income below the standard deduction produces zero ordinary tax;
- **Top-bracket test**: Income of \$1,000,000 exercises all seven brackets;
- **LTCG 0% bracket**: LTCG below \$47,025 (single) pays zero capital gains tax;
- **NIIT threshold boundary**: Income at exactly \$200,000 (single) produces zero NIIT; income at \$200,001 triggers the surtax;
- **Vectorization equivalence**: The vectorized implementation produces bit-identical results to the scalar implementation for all test inputs.

6.4 RMD Verification

The RMD model is verified against the IRS Uniform Lifetime Table [16]:

- Age 73 with \$1,000,000 prior-year balance: RMD = 1,000,000/26.5 = \$37,735.85;
- Age 80 with \$500,000: RMD = 500,000/20.2 = \$24,752.48;
- Age 72 (below start age): RMD = \$0;
- Age 120 (maximum table age): divisor = 2.0.

The engine enforces RMD satisfaction: if spending withdrawals from traditional accounts during the year are insufficient to meet the RMD, forced withdrawals are executed at the December step.

6.5 Spending Policy Invariants

Each spending policy is tested for the following invariants:

1. **Non-negativity**: $s_t \geq 0$ for all paths and steps;
2. **Zero on depletion**: $s_t = 0$ when $W_t \leq 0$;
3. **Shape correctness**: output shape is $(M,)$ matching the number of paths;
4. **Policy-specific bounds**:
   - Floor-ceiling: $F \cdot \Pi_t \leq s_t \leq C \cdot \Pi_t$ (when $W_t > 0$);
   - VPW: $r_{\min}/12 \cdot W_t \leq s_t \leq r_{\max}/12 \cdot W_t$;
   - Constant real: $s_t = s_0 \cdot \Pi_t$ exactly.

6.6 Test Coverage

The test suite comprises 263 tests organized across the following categories. Tests are executed via `pytest` with coverage reporting via `pytest-cov`.

Table 4: Test suite composition (v0.5.0, 263 tests).

Category	Count	Key Coverage
Engine (core simulation)	∼50	Pipeline steps, depletion
Return models	∼40	MVN, t, regime, bootstrap
Tax models	∼30	Brackets, NIIT, RMD
Spending policies	∼25	All 5 policies, invariants
Config validation	∼30	Cross-field, bounds
Analytics	∼25	Sensitivity, SWR, metrics
Serialization & I/O	∼15	JSON, CSV export
Integration & golden	∼20	End-to-end, reproducibility
Roth conversions	∼15	Fixed, fill-bracket
Other	∼13	Timeline, state, defaults

Testing Methodology. The test suite employs several complementary strategies:

- **Golden snapshot tests**: Exact numerical values for fixed seeds, catching any unintentional engine changes;
- **Table-driven parametric tests**: Tax bracket computations against IRS-published values across multiple income levels;
- **Invariant tests**: Mathematical properties that must hold regardless of input (non-negative wealth, weight normalization, monotonicity);
- **Boundary tests**: Edge cases such as zero income, maximum age, single-path simulation, and single-asset portfolios;
- **Integration tests**: End-to-end simulations with all features enabled (regime switching + antithetic + Roth + guardrails);
- **Benchmark tests**: Performance regression via `pytest-benchmark` on the critical simulation loop.

7 Related Work

7.1 Open-Source Tools

Several open-source retirement calculators exist, each with distinct limitations relative to MONTEPLAN:

FICalc and cFIREsim. Web-based calculators using historical backtesting (the "Trinity Study" approach). These replay actual market sequences rather than generating synthetic paths, limiting scenario analysis to the historical record. Neither supports multi-account tax modeling or parametric return distributions.

FireCalc. A longstanding historical-only calculator with spending rule variants (constant, percent-of-portfolio, Bernicke) but no multi-account treatment and no tax modeling.

7.2 Commercial Tools

eMoney Advisor and MoneyGuidePro. Professional financial planning platforms with sophisticated tax-aware modeling. However, they are closed-source, expensive ($3,000–5,000+ per year), and not auditable. Their Monte Carlo implementations are not publicly documented.

Boldin (formerly NewRetirement). Consumer-facing tool with some Monte Carlo capability and tax awareness. Closed-source with limited parametric flexibility.

7.3 Feature Comparison

Table 5 provides a systematic comparison of MONTEPLAN against the most widely used open-source and commercial tools.

Table 5: Feature comparison with existing tools.

Feature	MONTEPLAN	FICalc	cFIREsim	Commercial
Monte Carlo simulation	✓	–	–	✓
Historical backtesting	✓	✓	✓	✓
Parametric returns (MVN)	✓	–	–	?
Fat-tailed returns (t)	✓	–	–	–
Regime switching	✓	–	–	–
Multi-account (tax-aware)	✓	–	–	✓
Progressive tax brackets	✓	–	–	✓
LTCG preferential rates	✓	–	–	✓
NIIT surtax	✓	–	–	?
State income tax	✓	–	–	✓
RMD enforcement	✓	–	–	✓
Roth conversions	✓	–	–	✓
Dynamic spending (5+)	✓	P	P	✓
Antithetic variates	✓	–	–	–
Sensitivity analysis	✓	–	–	?
SWR finder	✓	–	–	?
Open source	✓	✓	✓	–
API / library access	✓	–	–	–
Reproducible (seeded)	✓	–	–	?

✓ = supported, – = not supported, P = partial, ? = unknown (closed source).

7.4 Academic Literature

The foundational work on safe withdrawal rates by Bengen [2] established the "4% rule" using historical backtesting over rolling 30-year periods. The Trinity Study [8] extended this analysis across stock/bond allocations.

Guyton and Klinger [11] introduced dynamic decision rules (guardrails) that adjust spending based on portfolio performance, finding that dynamic policies support higher initial withdrawal rates. Blanchett, Kowara, and Chen [3] developed optimal withdrawal strategies considering asset allocation and mortality. Blanchett, Finke, and Pfau [4] examined the impact of low-yield environments on safe withdrawal rates.

Hamilton [12] introduced the Markov regime-switching model for economic time series. Ang and Bekaert [1] and Guidolin and Timmermann [10] applied regime-switching to portfolio allocation.

Glasserman [9] provides the standard reference for Monte Carlo methods in financial engineering, including variance reduction techniques such as antithetic variates [13]. The block bootstrap methodology follows Politis and Romano [18].

The Ornstein–Uhlenbeck process for inflation modeling draws on the term structure literature

originating with Vasicek [22].

8 Conclusion

We have presented MONTEPLAN, an open-source Python framework (v0.5.0) for Monte Carlo simulation of multi-account retirement plans under realistic tax treatment. The framework addresses a critical gap in the landscape of financial planning tools: the absence of an open-source, auditable system that combines parametric return modeling, progressive tax brackets, multi-account treatment, and dynamic spending policies in a single coherent engine.

8.1 Summary of Contributions

The framework's key technical contributions are:

1. **Vectorized multi-account engine**: A three-dimensional tensor representation ($M \times A \times n$) with a 12-step monthly pipeline that eliminates per-path Python loops, achieving throughput suitable for interactive web-based exploration;

2. **Four return models**: Multivariate normal, Student-t (fat tails), Markov regime-switching (structural breaks), and historical block bootstrap (non-parametric), all sharing a common protocol for seamless substitution;

3. **Tax-aware decumulation**: US federal progressive brackets with preferential LTCG rates, the 3.8% NIIT surtax, configurable state tax overlays, RMD enforcement, and two Roth conversion strategies (fixed-amount and fill-bracket);

4. **Five dynamic spending policies**: From the simple (constant real) to the sophisticated (Guyton–Klinger guardrails), enabling exploration of the trade-off between spending stability and portfolio survival;

5. **Variance reduction**: Antithetic variates providing approximately $2\times$ variance reduction, with formal proof of effectiveness for monotone portfolio value functions;

6. **Analytics framework**: One-at-a-time sensitivity analysis with parallel execution, two-dimensional grid search for interaction effects, bisection-based safe withdrawal rate finder, and distributional metrics (max drawdown, spending volatility, ruin-by-age curves).

8.2 Design Principles

The framework's design adheres to four core principles:

1. **Vectorization over explicitness**: All operations across Monte Carlo paths use NumPy broadcasting rather than Python loops. This enables simulation of 20,000 paths over 40 years (480 monthly steps) in seconds on commodity hardware;

2. **Protocol-based extensibility**: Return models, spending policies, and tax models implement Python protocols (structural typing), enabling new implementations without modifying the engine;

3. **Reproducibility by construction**: Deterministic RNG (PCG64DXSM) with user-specified seeds, combined with SHA-256 config hashing, ensures every simulation result is exactly replicable and traceable to its inputs;

4. **Validation at every level**: Golden tests guard against regression, table-driven tests verify tax computations against IRS publications, and invariant tests ensure mathematical properties hold across all inputs.

8.3 Limitations

The current version (v0.5.0) has several limitations:

- **Estate planning**: No bequest optimization or estate tax modeling;
- **Social Security optimization**: Benefits are modeled as exogenous guaranteed income; the claiming age decision is not optimized;
- **Dynamic asset allocation**: Only static and linear glide path allocations are supported; no market-condition-responsive strategies;
- **Healthcare costs**: No age-dependent healthcare expenditure modeling (e.g., Medicare Part B, long-term care);
- **Mortality risk**: The simulation runs to a fixed end age rather than incorporating actuarial life tables;
- **Tax code evolution**: Tax brackets are static (2024 values); no modeling of legislative changes or TCJA sunset;
- **International taxes**: Only US federal and state taxes are implemented; no support for other jurisdictions.

8.4 Future Work

Planned extensions include:

1. **Mortality-weighted metrics**: Integrating actuarial life tables (SOA 2012 IAM) to compute mortality-weighted success probabilities and expected bequest values;

2. **Social Security optimization**: Brute-force or dynamic programming search over claiming ages (62–70) for individual and spousal benefits;

3. **Multi-objective optimization**: Jointly optimizing spending level, asset allocation, and Roth conversion strategy using Pareto-front methods;

4. **Tax-loss harvesting**: Modeling the tax benefit of realizing losses in taxable accounts to offset gains;

5. **Healthcare cost modeling**: Age-dependent healthcare expenditure curves calibrated to CMS data.

A US Federal Tax Brackets (2024)

Table 6: 2024 ordinary income brackets (single filer).

Taxable Income Up To	Marginal Rate	Taxable Income Up To	Marginal Rate
$11,600	10%	$23,200	10%
$47,150	12%	$94,300	12%
$100,525	22%	$201,050	22%
$191,950	24%	$383,900	24%
$243,725	32%	$487,450	32%
$609,350	35%	$731,200	35%
Above	37%	Above	37%

Left: Single. Right: Married Filing Jointly.
Standard deduction: $14,600 (single), $29,200 (MFJ).
Source: IRS Rev. Proc. 2023-34 [15].

Table 7: 2024 long-term capital gains brackets.

Filing Status	Threshold	Rate
	Up to $47,025	0%
Single	$47,025–$518,900	15%
	Above $518,900	20%
	Up to $94,050	0%
MFJ	$94,050–$583,750	15%
	Above $583,750	20%

B IRS Uniform Lifetime Table (RMD Divisors)

Table 8: IRS Uniform Lifetime Table divisors (selected ages). Full table covers ages 73–120.

Age	Divisor	Age	Divisor	Age	Divisor
73	26.5	83	17.7	93	10.1
74	25.5	84	16.8	94	9.5
75	24.6	85	16.0	95	8.9
76	23.7	86	15.2	96	8.4
77	22.9	87	14.4	97	7.8
78	22.0	88	13.7	98	7.3
79	21.1	89	12.9	99	6.8
80	20.2	90	12.2	100	6.4
81	19.4	91	11.5	105	4.6
82	18.5	92	10.8	110	3.5

Source: IRS Publication 590-B, Table III [16].

C Complete Parameter Reference

Table 9: Configuration parameter reference.

Parameter	Type	Default	Description
PlanConfig			
current_age	int	—	Current age (18–100)
retirement_age	int	—	Retirement age (> current)
end_age	int	—	Planning horizon (18–120)
monthly_spending	float	—	Monthly goal (today's $)
monthly_income	float	0	Earned income
income_growth_rate	float	0.0	Real income growth rate
income_end_age	int|None	retire	Age income stops
accounts	list	—	Account configs
discrete_events	list	[]	One-time cash events
guaranteed_income	list	[]	GI streams
MarketAssumptions			
expected_returns	list[float]	—	Per-asset annual returns
annual_volatilities	list[float]	—	Per-asset volatility
correlation_matrix	list[list]	—	$n \times n$ correlation
inflation_mean	float	0.03	Long-run inflation
inflation_vol	float	0.01	Inflation volatility
return_model	str	mvn	mvn, student_t, bootstrap, regime_switching

Parameter	Type	Default	Description
degrees_of_freedom	float|None	None	Student-t d.o.f. (> 2)
bootstrap_block_size	int	12	Block size (months)
expense_ratio	float	0.0	Annual fund fees
aum_fee	float	0.0	AUM fee
advisory_fee	float	0.0	Advisor fee
PolicyBundle			
spending.policy_type	str	const_real	Spending policy
rebalancing_strategy	str	calendar	calendar|threshold
rebalancing_months	list[int]	[1,7]	Calendar months
rebalancing_threshold	float	0.05	Drift threshold
withdrawal_order	list[str]	[tax,trad, roth]	Account priority
tax_model	str	flat	flat|us_federal
tax_rate	float	0.22	Flat effective rate
filing_status	str	single	single|married_jointly
state_tax_rate	float	0.0	State flat rate (0–15%)
include_niit	bool	False	NIIT 3.8% surtax
RothConversionConfig			
enabled	bool	False	Enable conversions
strategy	str	fixed_amt	fixed_amount, fill_bracket
annual_amount	float	0	Fixed conversion $
fill_to_bracket	float	0.22	Target marginal rate
start_age	float	55	Conversion start age
end_age	float	72	Conversion end age
SimulationConfig			
n_paths	int	5000	Monte Carlo paths
seed	int	42	Random seed
antithetic	bool	False	Antithetic variates
preset	str|None	None	fast|balanced|deep
store_paths	bool	False	Store full paths

D Streamlit Application

The web application [19] provides six pages for interactive exploration:

1. **Plan Setup**: Demographics, accounts, income, spending, guaranteed income streams, discrete events. Quick-start templates (Default, FIRE, Coast FIRE, Conservative Retiree).
2. **Portfolio**: Asset allocation, correlation matrix, return model selection (MVN, Student-t, regime-switching, bootstrap), fees, glide path.
3. **Run & Results**: Simulation execution with fan chart, spending fan chart, ruin-by-age curve. SWR finder expander. CSV export.
4. **Policies**: Spending policy configuration, rebalancing, withdrawal order, tax model, state tax, NIIT, Roth conversions.

5. **Compare Scenarios**: Save/load scenarios, overlay fan charts, dominance scatter plot.
6. **Sensitivity**: OAT tornado chart, 2D heatmap grid.

All simulation results are cached via `st.cache_data` keyed by the configuration SHA-256 hash.

References

[1] Andrew Ang and Geert Bekaert. International asset allocation with regime shifts. *Review of Financial Studies*, 15(4):1137–1187, 2002. doi: 10.1093/rfs/15.4.1137.

[2] William P. Bengen. Determining withdrawal rates using historical data. *Journal of Financial Planning*, 7(4): 171–180, 1994.

[3] David Blanchett, Maciej Kowara, and Peng Chen. Optimal withdrawal strategy for retirement-income portfolios. *Retirement Management Journal*, 2(3):7–20, 2012.

[4] David Blanchett, Michael Finke, and Wade D. Pfau. Low bond yields and safe portfolio withdrawal rates. *Journal of Wealth Management*, 16(2):55–62, 2013.

[5] Bogleheads Community. Variable percentage withdrawal. *Bogleheads Wiki*, 2023.

[6] Philippe Cogneau and Valeri Zakamouline. Block bootstrap methods and the choice of stocks for the long run. *Quantitative Finance*, 13(9):1443–1457, 2013. doi: 10.1080/14697688.2012.713115.

[7] Samuel Colvin and Pydantic Contributors. Pydantic: Data validation using Python type annotations. https://docs.pydantic.dev/, 2024. Version 2.x.

[8] Philip L. Cooley, Carl M. Hubbard, and Daniel T. Walz. Retirement savings: Choosing a withdrawal rate that is sustainable. *AAII Journal*, 20(2):16–21, 1998.

[9] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer, New York, 2003. doi: 10.1007/978-0-387-21617-1.

[10] Massimo Guidolin and Allan Timmermann. Asset allocation under multivariate regime switching. *Journal of Economic Dynamics and Control*, 31(11):3503–3544, 2007. doi: 10.1016/j.jedc.2006.12.004.

[11] Jonathan T. Guyton and William J. Klinger. Decision rules and maximum initial withdrawal rates. *Journal of Financial Planning*, 19(3):49–57, 2006.

[12] James D. Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57(2):357–384, 1989. doi: 10.2307/1912559.

[13] J. M. Hammersley and K. W. Morton. A new monte carlo technique: Antithetic variates. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52(3):449–475, 1956. doi: 10.1017/S0305004100031455.

[14] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kreveln, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[15] Internal Revenue Service. Revenue procedure 2023-34: 2024 tax year inflation adjustments. https://www.irs.gov/pub/irs-drop/rp-23-34.pdf, 2023.

[16] Internal Revenue Service. Publication 590-b: Distributions from individual retirement arrangements (IRAs). https://www.irs.gov/publications/p590b, 2024.

[17] Melissa E. O'Neill. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. *Harvey Mudd College Computer Science Technical Report*, (HMC-CS-2014-0905), 2014.

[18] Dimitris N. Politis and Joseph P. Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994. doi: 10.1080/01621459.1994.10476870.

[19] Streamlit, Inc. Streamlit: The fastest way to build data apps in Python. https://streamlit.io/, 2024.

[20] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the Brownian motion. *Physical Review*, 36(5):823–841, 1930. doi: 10.1103/PhysRev.36.823.

[21] Vanguard Research. Dynamic spending: A better way to budget in retirement. *Vanguard Research*, 2021.

[22] Oldrich Vasicek. An equilibrium characterization of the term structure. *Journal of Financial Economics*, 5(2): 177–188, 1977. doi: 10.1016/0304-405X(77)90016-2.