Advanced Programming 2026

Swiss Portfolio backtesting tool for retirement planning

Final Project Report

Andre Willhalm
andre.willhalm@unil.ch
Student ID: 19618149

10.01.2026

Abstract

Retirement portfolio planning is traditionally based on deterministic rules such as the "4% rule", which ignore sequence-of-returns risk, regime shifts, and foreign exchange exposure. These simplifications are especially problematic for Swiss investors, whose portfolios are typically invested in a mix of Swiss franc and U.S. dollar denominated assets. In this project, we develop a Monte Carlo portfolio simulation framework that incorporates historical bootstrapping, regime-aware return dynamics, currency conversion, inflation-adjusted withdrawals, and Bayesian portfolio optimization. Monthly asset returns are resampled using independent, block, and regime-switching bootstrap methods, allowing realistic modeling of volatility clustering and economic cycles. A flexible withdrawal engine supports both optimized reduction (alpha rule) and a strict spending floor in adverse months. Portfolio weights, withdrawal rates, and dynamic cut parameters are optimized using Optuna to satisfy survival probability constraints while maximizing sustainable withdrawals. I evaluate survival probabilities, terminal wealth distributions, and drawdowns across bootstrap regimes. The results show that optimized portfolios significantly improve survival and allow higher withdrawals compared to naive allocations, while regime-based bootstrapping produces more conservative and realistic risk estimates. The framework provides a robust decision-support tool for long-term retirement planning under uncertainty.

Keywords: data science, Python, machine learning, Bayesian optimization, Monte-Carlo simulation, iid/block/regime bootstrapping

Contents

1 Introduction

The classical "4% rule" [1][**Bengen1994**] and its variants rely heavily on historical sequence-of-returns assumptions that often underestimate serial dependence and regime shifts in financial markets. Real markets exhibit volatility clustering, long bear markets, periods of relative stagnation or slow growth, currency fluctuations, and regime changes that can severely impact long-term portfolio sustainability. These risks are especially pronounced for Swiss investors, whose portfolios are frequently exposed to assets in foreign currencies, especially US stocks and bonds, and therefore subject to foreign exchange risk relative to the Swiss franc. Moreover, many studies model withdrawals as a percentage of portfolio value, which fails to capture the rigidity of real retirement spending needs.

This leads us to the key research question of this project:

How does sustainability, for a multi-asset, multi-currency portfolio under fixed, inflation-adjusted withdrawals vary across different historical bootstrapping assumptions, and how can portfolio allocations be optimized to maximize survival and/or spending?

To answer this, I develop a Monte Carlo simulation framework that uses historical bootstrapping rather than parametric return models. We incorporate three resampling schemes — independent, block, and regime-based bootstrapping to capture serial correlation, volatility clustering, and macroeconomic regime shifts. On top of this engine, we implement a flexible withdrawal system with both proportional cuts and strict minimum spending floors.

The objective of this project is not only to simulate outcomes, but also to optimize portfolio composition and withdrawal parameters using Bayesian optimization to maximize sustainability under uncertainty.

2 Literature Review

Early retirement sustainability studies [1][2][**Bengen 1994**, **Trinity study; Cooley 1998**] established fixed-percentage withdrawal heuristics based on historical U.S. market returns. Subsequent work highlighted sequence-of-returns risk and its dominant role in portfolio failure, motivating the use of Monte Carlo simulation instead of purely historical backtests.

Block bootstrapping has been proposed as a method to preserve autocorrelation in return series [3][**Politis & Romano 1994**], while regime-switching models capture volatility clustering and macroeconomic states [4][**Hamilton 1989**]. More recent research emphasizes the importance of incorporating regime dynamics in portfolio simulations.[5][**Ang & Bekaert 2002**] Ang & Bekaert demonstrated that regime shifts strongly affect long-horizon investment outcomes.

In parallel, dynamic withdrawal strategies such as the [6][**Guyton-Klinger 2006**] Guyton-Klinger rules and variable spending frameworks attempt to reduce ruin risk by cutting withdrawals in bad markets. However, these methods are typically evaluated using simplified return models and without foreign exchange effects.

Bayesian optimization has seen increasing adoption in financial modeling due to its efficiency in high-dimensional, non-convex parameter spaces. However, its application to withdrawal sustainability under bootstrap-based Monte Carlo simulation remains limited.

This paper builds upon these strands by combining multi-asset multi-bootstrapped Monte Carlo simulation, FX and inflation-aware returns, regime modeling, and optimization-based asset allocation and withdrawal calibration in a single coherent framework.

3 Methodology

This section describes the data, modeling framework, and computational pipeline used to evaluate and optimize retirement portfolio withdrawal strategies under uncertainty.

3.1 Data Description

The empirical foundation of this study is a historical monthly returns dataset constructed from daily returns from [7]yfinance library, Yahoo! Finance's API, for assets and currency exchange rates, and Switzerland's [8]*Bureau Federal de la Statistique (BFS)* for monthly historical inflation rates. The daily returns were transformed into monthly returns by computing end-of-the-month to end-of-the-month returns. The dataset includes major asset classes relevant to a Swiss investor, namely U.S. equities (S&P 500), Swiss equities (SMI), gold, long-term (20+y) U.S. government bonds, and mid-term (7-15y) Swiss government bonds. Asset returns are provided either in U.S. dollars or Swiss francs depending on the original market denomination. In addition, the dataset contains a USD/CHF exchange rate series and Swiss inflation rates series, allowing both currency conversion and real withdrawal adjustments. Dividend yielding assets such as equities and were, when available, fetched as adjusted-close, accounting for splits and dividend distribution. Asset Returns were total, accounting for dividend and coupon distribution, rendering the simulation less conservative, as swiss investors have to pay dividend and wealth taxes, that are here not accounted for. The inflation rates were collected from BFS and initially pasted into an excel file where dates and percentage returns were formatted to match yfinance data, before being loaded into my directory. Asset and FX returns were fetched and cleaned separately from swiss inflation rates.

The time span of the dataset begins January 1st, 1975 and extends to the most recent available observations, which was the 22nd of December 2025. It yielded more than 500 monthly data points. However, the dataset, except for inflation rates dating back to January 1983, was constrained to January 2008 by the limited availability of the 7-15y Swiss government bonds asset (ticker CSBGC0.SW). The dataset currently consists of and displays 213 data points when run. Each row corresponds to one month and contains total returns for all investable assets, the foreign exchange return, and the inflation rate. This structure enables consistent simulation of multi-asset portfolios with currency effects and inflation-aware spending.

Before simulation, all U.S.-denominated asset returns are converted into Swiss franc returns using the relation

$$r_t^{CHF} = (1 + r_t^{USD})(1 + r_t^{FX}) - 1,$$

where r_t^{FX} is the USD/CHF return. This ensures that all portfolio returns are measured in the investor's domestic currency. Inflation data are not treated as an investable asset but are used only to adjust withdrawals when real spending is enabled.

The dataset is checked for missing values, alignment errors, and inconsistent column names during loading. Missing values in asset returns, inflation, and FX series are handled via forward-fill and backward-fill after computing monthly returns, ensuring a complete panel suitable for bootstrapping and regime detection.

3.2 Approach

The core of the methodology is a Monte Carlo simulation framework driven by historical bootstrapping. Rather than assuming parametric return distributions, future return paths are generated by resampling from the empirical return history. Three bootstrapping methods are implemented: independent and identically distributed (IID) sampling, block bootstrapping of contiguous months, and regime-based bootstrapping using K-means machine learning for volatility clustering, using a Markov transition matrix. For each simulated path, a multi-asset portfolio evolves according to the sampled monthly returns. Withdrawals are taken at the end of each month following one of two rules, depending on the optimization mode selected. The first rule is an *alpha-cut* rule, where withdrawals are reduced by a factor α in months with negative portfolio returns. The second is a *true withdrawal floor* rule, in which the investor withdraws a preferred amount in positive months and a fixed minimum floor in negative months. This rule enforces consumption stability during market downturns and reflects realistic spending constraints. Bayesian optimization is used to select optimal portfolio weights and withdrawal control parameters. Three optimization modes are supported. Mode A optimizes asset weights only while enforcing a strict withdrawal floor rule. Mode B maximizes the withdrawal rate subject to a target survival probability. Mode C maximizes survival by adjusting both weights and the alpha parameter. [9]Optuna's Tree-structured Parzen Estimator (TPE) algorithm is used to efficiently search the high-dimensional parameter space. The bootstrapping method used for the Monte Carlo simulation during the Optuna optimization is either iid or block based, depending on the mode selected in the code. Only then, after the selected number of Optuna trial runs ends, will wealth paths be resampled using all 3 bootstrapping methods. This is done to avoid overwhelming the simulator.

All preprocessing steps are designed to transform raw financial time series into a consistent, complete, and simulation-ready dataset while minimizing implicit modeling assumptions. The starting point consists of historical monthly return series for multiple asset classes, including equities, government bonds, gold, inflation, and foreign exchange rates. Raw data are loaded from a consolidated CSV file constructed using yfinance library and BFS for Swiss inflation rates. Prior to any modeling, all series are aligned from daily to a common monthly frequency to ensure temporal consistency across assets, except for inflation rates that are already monthly. For assets denominated in U.S. dollars, returns are converted into Swiss francs (CHF) to reflect the perspective of a Swiss-based investor. This conversion is performed using the standard multiplicative return identity:

$$r_t^{CHF} = (1 + r_t^{USD})(1 + r_t^{FX}) - 1,$$

where r_t^{USD} denotes the asset return in U.S. dollars and r_t^{FX} represents the monthly USD/CHF exchange rate return. The foreign exchange series itself is retained in the dataset for conversion purposes but is explicitly excluded from the investable asset universe. The inflation series is therefore removed from the asset return matrix prior to portfolio simulation but retained for withdrawal indexation. After all transformations, the investable asset universe is defined by excluding inflation and foreign exchange columns. Only assets with complete historical coverage after preprocessing are retained to ensure numerical stability during simulation and optimization. Missing observations arising from differences in asset inception dates or data availability are handled using forward-fill followed by backward-fill imputation. This approach ensures a fully populated return matrix, which is required for block bootstrapping, regime-based sampling, and matrix-based portfolio calculations. Given the relatively low frequency (monthly) and the long-term nature of the analysis, this imputation strategy preserves continuity while avoiding the introduction of artificial return dynamics. Finally, user-specified portfolio weights are normalized to sum to one before simulation. If no user-defined weights are provided, a predefined baseline allocation is applied. No additional scaling, standardization, or return smoothing is performed, ensuring that all simulated paths are driven by empirically observed return distributions.

Portfolio survival is defined as the fraction of Monte Carlo paths that maintain positive wealth throughout the full investment horizon. This survival rate is the primary risk metric. Additional metrics include the number of negative months, median terminal wealth, average annualized returns net of withdrawals, volatility, and maximum drawdown metrics for the median portfolio, computed from representative median paths.

3.3 Implementation

The entire system is implemented with Python and follows a modular architecture. The numerical core relies on NumPy for vectorized computation and Pandas for data manipulation. As mentioned several times before, data are fetched from yfinance and BFS for inflation rates. Monte Carlo simulations and bootstrapping logic are implemented in a dedicated `MonteCarloSimulator` class, which encapsulates portfolio dynamics, withdrawal rules, and rebalancing logic. Regime-based bootstrapping using K-means is computed using Scikit-learn library. Optimization is handled by the Optuna library, which performs Bayesian hyperparameter search across asset weights, withdrawal rates, and alpha values. Plots are computed using Matplotlib.

The project is structured into separate modules for data loading, simulation, optimization, and visualization. A pipeline module orchestrates the full workflow, including running baseline simulations with user-defined weights and optimized simulations across all bootstrap modes. This ensures that results are comparable across consistent experimental settings.

The user can interact with the system either through a command-line interface or a Streamlit-based graphical interface. Both frontends call the same underlying pipeline, guaranteeing consistency between batch experiments and interactive analysis.

The following code snippet in my models.py file encompasses weights, month-to-month returns application, withdrawal and rebalancing logic used in the simulation. (Some code lines are voluntarily skipped to avoid the code block taking up too much space in the report)

```
for t in range(n_periods):

    # START-OF-MONTH weights (before returns)
    start_total = float(holdings.sum())
    start_w = holdings / start_total if start_total > 0 else self.weights_vec
    # 1) Apply asset returns to holdings
    holdings *= (1.0 + R[t, :])

    total_before_withdraw = float(holdings.sum())

    # If ruined
    if floor_at_zero and total_before_withdraw <= 0:
        holdings[:] = 0.0
        paths[p, t + 1] = 0.0
        continue

    # 2) Compute portfolio return for alpha rule
    # portfolio return = weighted return based on holdings weights *at start of
     month*
    if total_before_withdraw > 0:
        port_r = float(np.dot(start_w, R[t, :]))
    else:
        port_r = -1.0


    # 3) Compute withdrawal (either alpha_cut or strict neg_to_floor)
    if withdrawal_rule == "neg_to_floor":
        if pref_amt is None or floor_amt is None:
            raise ValueError("withdrawal_rule='neg_to_floor' requires
   preferred_withdrawal and withdrawal_floor.")

```

```
        if floor_amt > pref_amt:
            raise ValueError("withdrawal_floor must be <= preferred_withdrawal for withdrawal_rule='neg_to_floor'.")
        # Strict rule:
        #   negative month -> floor
        #   non-negative month -> preferred
        withdrawal = float(floor_amt if port_r < 0.0 else pref_amt)



    else:
    # Default legacy behavior: start from withdraw_amt (preferred), cut by alpha in negative months,
        withdrawal = float(withdraw_amt)

        if port_r < 0.0 and alpha > 0.0:
            withdrawal *= (1.0 - alpha)

        if withdrawal_floor is not None:
            withdrawal = max(withdrawal, float(withdrawal_floor))


    # Cannot withdraw more than total wealth
    withdrawal = min(withdrawal, total_before_withdraw)

    # Withdraw proportionally from all holdings
    if withdrawal > 0 and total_before_withdraw > 0:
        withdraw_frac = min(1.0, withdrawal / total_before_withdraw)
        holdings *= (1.0 - withdraw_frac)
        withdrawal_paid = withdrawal

    # 3b) Inflation-adjust next period's base withdrawal (keeps real spending constant)
    if infl_values is not None:
        infl = 1.0 + float(infl_values[idx[t]])
        withdraw_amt *= infl
        if pref_amt is not None:
            pref_amt *= infl
        if floor_amt is not None:
            floor_amt *= infl

    # 4) Rebalance (yearly)
    if self.config.rebalance_frequency == "yearly":
    # rebalance at end of each year (after withdrawals)
    if (t + 1) % self.periods_per_year == 0 and total_after_withdraw > 0:
        holdings = total_after_withdraw * self.weights_vec.copy()
```

Listing 1: Core monthly portfolio returns, withdrawal and rebalancing logic

4 Results

This section presents the empirical performance of the proposed Monte Carlo and bootstrap-based portfolio simulation framework. I evaluate both the user-defined baseline portfolio (BASIC) and the optimized portfolio obtained via Bayesian optimization (OPTIMIZED), under three distinct resampling schemes: independent and identically distributed (IID) bootstrapping, block bootstrapping, and regime-based bootstrapping.

4.1 Experimental Setup

All experiments were executed on a consumer-grade workstation equipped with an Intel i5-class CPU, 8 GB of RAM, and running Windows 11. The simulation and optimization pipeline was implemented in Python 3.11. The main libraries used were NumPy and Pandas for numerical computation and data manipulation, Optuna for Bayesian optimization, and Streamlit and Matplotlib/Altair for visualization.

The Monte Carlo engine simulated 1,000 portfolio paths per configuration, over a 30-year horizon with monthly frequency (360 periods). For the block bootstrap, a block size of 12 months was used, corresponding to one calendar year, to preserve serial correlation. The regime-based bootstrap used $K = 3$ regimes, a stable, a bearish and a bullish regime, a rolling volatility window of 12 months, and a minimum of 24 observations per regime to ensure stability of the clustering and transition matrix estimation.

Bayesian optimization was run for 50 trials using the Tree-structured Parzen Estimator (TPE) sampler. Depending on the selected optimization mode, the optimizer searched over portfolio weights, withdrawal reduction parameter α, and/or withdrawal rate multipliers. In all cases, the target survival constraint was set to 95%, meaning that at least 95% of simulated paths had to avoid ruin. Random state was set to 42 the ensure reproducibility. Inflation-aware withdrawals is activated by default. α is capped at a 50% reduction and the withdrawal multiplier to 2.

4.2 Performance Evaluation

The following results were generated from a baseline scenario of a 1'200'000 CHF portfolio value, with a basic portfolio allocation of 50% invested in the S&P500 and 50% in the Swiss Market Index. The baseline withdrawals were a preferred amount of 4000 CHF per month with a minimum (floor) of 3000 CHF per month. This scenario replicates the classical 4% rule for a common portfolio for a swiss investor, with a more conservative 3% rule when monthly portfolio returns are negative. Only optimization modes A and C are displayed in the results section for space concern and in order to compare the impact of optimized asset allocation vs optimized asset allocation and α for the same goal of maximizing survival rate. Mode B will be commented in the discussion section and plots and tables will be available in the appendix. The first set of tables display survival rates, number of failed portfolios, number of negative months for the median portfolio and P10, Median and P90 terminal wealths, for all 3 bootstrapping modes for both the baseline and the optimized portfolios. The second set of tables display the average annualized portfolio returns net of withdrawals as well as the average annualized volatility and max drawdown, for each scenario and bootstrapping method.

Optimization mode A:

Scenario	Bootstrap	Survival	Failed (count)	Failed (%)	Neg months / path (median)	Neg months / path (mean)	Total months	Median terminal (CHF)	P10 terminal (CHF)	P90 terminal (CHF)
BASIC (user weights, floor rule)	IID	91.20%	88.00	8.80%	134.00	133.53	360.00	CHF 2,132,024.93	CHF 80,441.43	CHF 8,110,978.30
BASIC (user weights, floor rule)	BLOCK	94.60%	54.00	5.40%	134.00	133.58	360.00	CHF 3,428,975.87	CHF 439,474.53	CHF 10,105,165.46
BASIC (user weights, floor rule)	REGIME	90.80%	92.00	9.20%	133.00	132.69	360.00	CHF 2,217,762.03	CHF 61,519.55	CHF 8,858,543.07
OPTIMIZED (Bayesian)	IID	99.00%	10.00	1.00%	132.00	132.09	360.00	CHF 2,853,021.22	CHF 886,968.29	CHF 6,759,643.42
OPTIMIZED (Bayesian)	BLOCK	100.00%	0.00	0.00%	134.00	133.84	360.00	CHF 3,308,219.84	CHF 1,263,278.35	CHF 6,488,537.53
OPTIMIZED (Bayesian)	REGIME	99.60%	4.00	0.40%	131.00	130.51	360.00	CHF 3,177,203.93	CHF 1,027,742.89	CHF 7,819,336.84

Figure 1: Baseline vs optimized comparison

Scenario	Bootstrap	Median-path CAGR	Median-path Vol (ann.)	Median-path Max Drawdown
BASIC	IID	1.93%	12.75%	39.41%
BASIC	BLOCK	3.56%	12.77%	51.34%
BASIC	REGIME	2.08%	12.67%	56.80%
OPTIMIZED	IID	2.92%	8.90%	23.88%
OPTIMIZED	BLOCK	3.44%	8.31%	17.37%
OPTIMIZED	REGIME	3.29%	8.25%	23.56%

Figure 2: Baseline vs optimized performance table

Optimization mode C:

Scenario	Bootstrap	Survival	Failed (count)	Failed (%)	Neg months / path (median)	Neg months / path (mean)	Total months	Median terminal (CHF)	P10 terminal (CHF)	P90 terminal (CHF)
BASIC (user weights, floor rule)	IID	91.20%	88.00	8.80%	134.00	133.53	360.00	CHF 2,132,024.93	CHF 80,441.43	CHF 8,110,978.30
BASIC (user weights, floor rule)	BLOCK	94.60%	54.00	5.40%	134.00	133.58	360.00	CHF 3,428,975.87	CHF 439,474.53	CHF 10,105,165.46
BASIC (user weights, floor rule)	REGIME	90.80%	92.00	9.20%	133.00	132.69	360.00	CHF 2,217,762.03	CHF 61,519.55	CHF 8,858,543.07
OPTIMIZED (Bayesian)	IID	99.20%	8.00	0.80%	140.00	139.44	360.00	CHF 4,819,448.03	CHF 1,492,464.62	CHF 12,153,708.19
OPTIMIZED (Bayesian)	BLOCK	100.00%	0.00	0.00%	139.00	138.65	360.00	CHF 5,455,696.79	CHF 2,105,934.16	CHF 10,850,754.84
OPTIMIZED (Bayesian)	REGIME	99.60%	4.00	0.40%	138.00	138.38	360.00	CHF 5,266,261.07	CHF 1,717,271.42	CHF 13,010,374.12

Figure 3: Baseline vs optimized comparison

Scenario	Bootstrap	Median-path CAGR	Median-path Vol (ann.)	Median-path Max Drawdown
BASIC	IID	1.93%	12.75%	39.41%
BASIC	BLOCK	3.56%	12.77%	51.34%
BASIC	REGIME	2.08%	12.67%	56.80%
OPTIMIZED	IID	4.74%	11.13%	34.48%
OPTIMIZED	BLOCK	5.18%	9.68%	20.38%
OPTIMIZED	REGIME	5.05%	10.15%	22.48%

Figure 4: Baseline vs optimized performance table

4.3 Visualizations

The following plots display the optimized asset allocation as a pie chart and the evolution of the median wealth paths for the baseline and optimized portfolios for each 3 bootstrapping methods.

Optimization mode A:

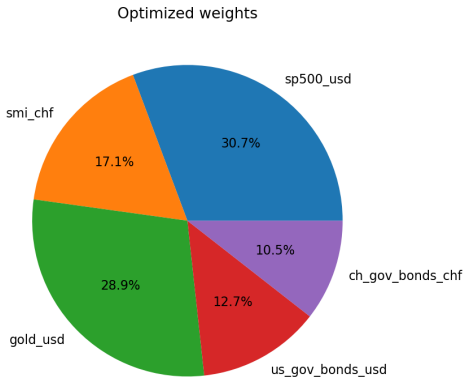

Figure 5: optimal asset allocation

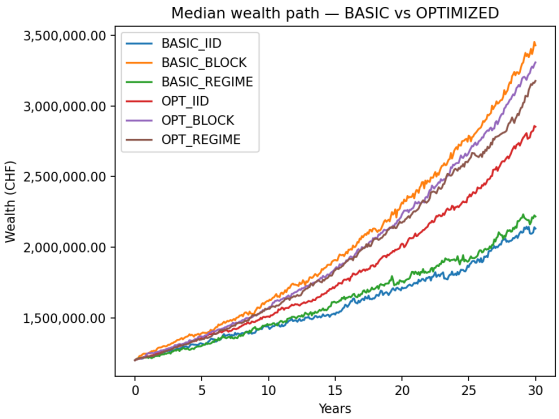

Figure 6: Median wealth paths

Optimization mode C:

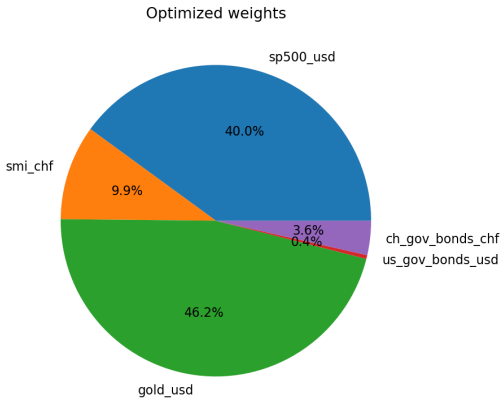

Figure 7: optimal asset allocation

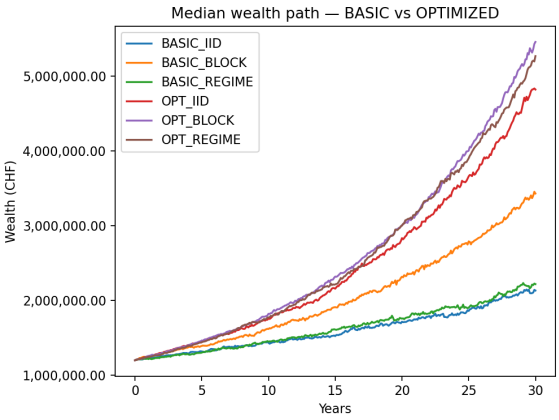

Figure 8: Median wealth paths

5 Discussion

A major strength of the proposed framework is its ability to produce economically meaningful portfolio allocations and withdrawal strategies under multiple sources of uncertainty. The introduction of the strict withdrawal floor rule (Mode A) provided a realistic behavioral constraint. Investors rarely reduce spending arbitrarily during market crashes, and the preferred-versus-floor withdrawal rule captured this asymmetry well. The Bayesian optimization procedure was able to identify portfolios that substantially improved survival probability compared to the user's baseline allocation while maintaining or increasing the withdrawal rate. In particular, Mode A (optimize asset allocation under floor rule) found significantly more sustainable (5-9% points) portfolios without changing the withdrawal structure desired. Mode C enabled the investor to preserve the improved survivability while slightly reducing α to 23% compared to the strict floor rule of mode A of a 25% reduction, thanks to a better asset allocation under α rule. Mode B (maximizing withdrawal amount under target survival rate) allows the investor to significantly increase his withdrawal amount while still slightly improving survival odds compared to baseline (see Appendix for mode B figures). The bootstrap-based return generation also proved effective. The iid sampling method generates a flatter distribution with more dispersed wealth paths, providing worse survival odds for more conservative backtesting, although one could argue a less credible simulation. The Regime bootstrapping method preserved important time-series dependencies that are absent under iid sampling, such as volatility clustering and persistent drawdowns. As a result, the simulated wealth paths exhibited more realistic crisis-like periods, making the stress-testing of withdrawal strategies much more credible. In particular, it captured transitions between low-volatility and high-volatility environments, producing a more realistic distribution of wealth paths than under iid sampling, illustrated by similar survival odds but slightly higher P10, median and P90 terminal wealth values. On the other hand, block bootstrapping displays significantly better survival odds and terminal wealth than the other 2 bootstrapping methods. It can be explained by "thinner" tails than iid and regime sampling might "overcluster" high volatility-low returns sequences, thus facing higher sequence-of-returns risks.

One of the main challenges was the interaction between withdrawals, portfolio returns, and bootstrapped regimes. When withdrawals are fixed in nominal terms, even moderate drawdowns early in retirement can lead to rapid portfolio depletion. This made the optimization landscape highly non-linear and sometimes unstable, especially when alpha and withdrawal rates were both optimized. Another challenge was computational. Each Optuna trial required a thousand of Monte Carlo paths, and each path involved bootstrapped multi-asset simulations with rebalancing and inflation adjustments. This made optimization expensive and required careful control of the number of paths and trials to keep runtime manageable. The regime bootstrapping method was also sensitive to hyperparameters such as the volatility window and minimum samples per regime. If these were chosen poorly, regime clustering became unstable or collapsed into iid behavior.

The results confirmed expectations of more conservative simulation results under iid sampling. However, the stark difference between block bootstrapping and the other 2 methods' results was surprising. I expected regime-based bootstrapping to deliver results closer to the block method. It was also expected that optimized portfolios would tilt toward lower-volatility and defensive assets, as this was observed in most optimized allocations, by higher proportions of gold. As expected, the S&P500 consistently received high weights as it possesses the highest average returns, allowing for higher sustainable withdrawals as you need high returns to offset high withdrawals. What was more surprising in the asset allocation was the generally "low" allocation to swiss equities (SMI) as I assumed that an asset non-subjected to currency risk with higher returns than bonds and gold would take a significant portion of the portfolio.

Several limitations should be acknowledged. First, the historical bootstrap assumes that

the future will resemble the past in distribution and regime structure. Structural breaks, prolonged low-return environments, or unprecedented macroeconomic conditions are not captured. Second, transaction costs, although generally very low with current online brokers, taxes, and liquidity constraints are not modeled. These would reduce effective returns and increase the true risk of ruin. The model also assumes perfect rebalancing at no cost. Taxes in Switzerland on investments are twofold: A dividend withholding tax depending on the financial assets' country of domicile, 35% for swiss assets, 15% for american assets. Although recoverable through tax credits, part of the dividends are directly automatically taxed and tax credits are only applied the following year. Dividends are then taxed at an investor's marginal tax rate, thus varying significantly from one investor to an other depending on all of their other sources of income and their amounts. Switzerland also has a wealth tax on an individual's net worth. Although relatively low, this tax might force certain investors to cede small shares of their portfolio each year. The returns computed in this model are total returns assuming dividends and coupons reinvested. This evidently does not take into account taxes and artificially raises survival odds. Third, withdrawals are modeled deterministically aside from the alpha and floor rules and inflation is monthly-adjusted, which is not really realistic from a behavioral point of view. In reality, retirees may adapt spending in more complex, state-dependent ways. Finally, regime identification is based on k-means clustering, which imposes sharp regime boundaries and does not fully capture continuous changes in market dynamics.

6 Conclusion and Future Work

6.1 Summary

By combining multi-asset historical bootstrapping, regime-aware return sampling, inflation-adjusted withdrawals, and Bayesian optimization, the system provides a flexible and economically grounded tool for long-horizon portfolio analysis.

The results demonstrate that portfolio survival and sustainable withdrawal rates depend critically on how return dynamics are modeled. IID bootstrapping produces slightly unrealistic outcomes, whereas block bootstrapping preserves serial correlation better. The Bayesian optimization engine successfully exploited these dynamics to identify portfolio allocations and withdrawal strategies that improved survival probability while maintaining or increasing spending relative to user-defined baselines.

The introduction of a strict withdrawal floor further enhanced realism by modeling consumption rigidity during market downturns. Overall, the framework delivers a robust decision-support system for retirement planning that is both data-driven and behaviorally plausible.

6.2 Future Directions

Several extensions could further improve the framework.

Methodological improvements include incorporating stochastic inflation models, dynamic spending rules that depend on portfolio health, and more advanced regime models such as Gaussian mixture models or macroeconomic factor-based regimes. Adding transaction costs, taxes, and rebalancing frictions would also improve realism.

Additional experiments could analyze sensitivity to different historical periods, test extreme stress scenarios, and compare performance against traditional rules such as the 4% rule or constant-mix strategies. Cross-validation across sub-periods could also assess robustness.

Real-world applications include integrating the model into financial planning tools, enabling advisors and individuals to evaluate personalized retirement strategies under realistic market uncertainty. With further development, the system could support scenario-based planning, regulatory stress testing, and automated portfolio design for long-term investors.

References

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

[2] Cooley, P. L., Hubbard, C. M., & Walz, D. T. (1998). *Retirement savings: Choosing a withdrawal rate that is sustainable*. AAII Journal, 20(3), 16–21.

[3] Politis, D. N., & Romano, J. P. (1994). *The stationary bootstrap*. Journal of the American Statistical Association, 89(428), 1303–1313.

[4] Hamilton, J. D. (1989). *A new approach to the economic analysis of nonstationary time series and the business cycle*. Econometrica, 57(2), 357–384.

[5] Ang, A., & Bekaert, G. (2002). *International asset allocation with regime shifts*. Review of Financial Studies, 15(4), 1137–1187.

[6] Guyton, J. T., & Klinger, W. J. (2006). *Decision rules and portfolio management for retirees*. Journal of Financial Planning, 19(10), 48–57.

[7] Yahoo Finance (2025). *Historical market data*. Available at: `https://finance.yahoo.com`

[8] Swiss Federal Statistical Office (2025). *Swiss Consumer Price Index*. Available at: `https://www.bfs.admin.ch/bfs/fr/home/statistiques/prix/indice-prix-consommation/resultats-detailles.assetdetail.36328230.html`

[9] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A next-generation hyperparameter optimization framework*. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2623–2631.

A Additional Figures

Optimization mode B:

The model found an α of 46%.

Scenario	Bootstrap	Survival	Failed (count)	Failed (%)	Neg months / path (median)	Neg months / path (mean)	Total months	Median terminal (CHF)	P10 terminal (CHF)	P90 terminal (CHF)
BASIC (user weights, floor rule)	IID	91.20%	88.00	8.80%	134.00	133.53	360.00	CHF 2,132,024.93	CHF 80,441.43	CHF 8,110,978.30
BASIC (user weights, floor rule)	BLOCK	94.60%	54.00	5.40%	134.00	133.58	360.00	CHF 3,428,975.87	CHF 439,474.53	CHF 10,105,165.46
BASIC (user weights, floor rule)	REGIME	90.80%	92.00	9.20%	133.00	132.69	360.00	CHF 2,217,762.03	CHF 61,519.55	CHF 8,858,543.07
OPTIMIZED (Bayesian)	IID	90.30%	97.00	9.70%	131.50	130.41	360.00	CHF 2,961,624.99	CHF 36,540.30	CHF 11,092,581.48
OPTIMIZED (Bayesian)	BLOCK	95.40%	46.00	4.60%	130.00	129.79	360.00	CHF 4,259,581.54	CHF 641,938.61	CHF 11,362,692.26
OPTIMIZED (Bayesian)	REGIME	90.60%	94.00	9.40%	131.00	129.96	360.00	CHF 3,133,476.74	CHF 35,041.36	CHF 10,981,781.05

Figure 9: Baseline vs optimized comparison

Scenario	Bootstrap	Median-path CAGR	Median-path Vol (ann.)	Median-path Max Drawdown
BASIC	IID	1.93%	12.75%	39.41%
BASIC	BLOCK	3.56%	12.77%	51.34%
BASIC	REGIME	2.08%	12.67%	56.80%
OPTIMIZED	IID	3.05%	12.90%	58.74%
OPTIMIZED	BLOCK	4.32%	12.59%	31.55%
OPTIMIZED	REGIME	3.25%	12.14%	45.60%

Figure 10: Basic vs optimized performance table

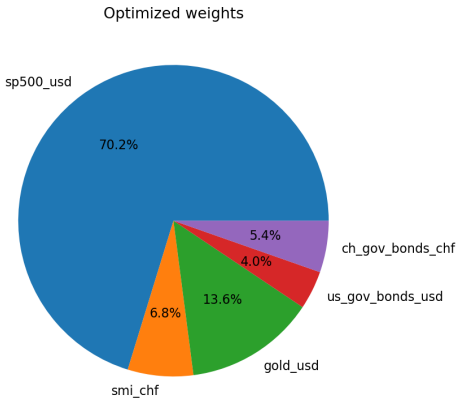

Figure 11: optimal asset allocation

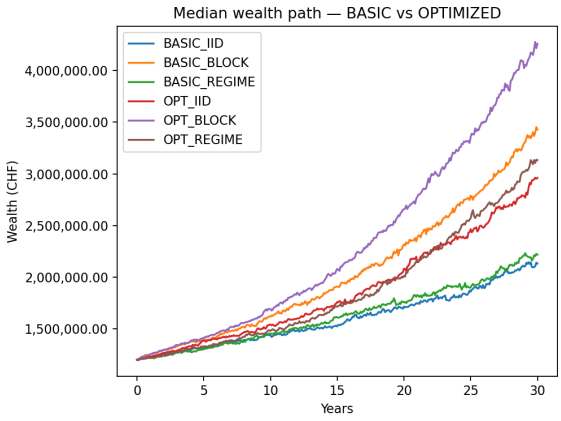

Figure 12: Median wealth paths

B Code Repository

GitHub Repository: `https://github.com/DeDe1026/portfolio-backtesting-tool`

Repository structure:

```
portfolio-backtesting-tool/
|
+-- app/
|   +-- streamlit_app.py          # Interactive UI
|
+-- data/
|   +-- raw_cache/                # Daily asset data
|   +-- raw/
|       +-- monthly_returns_native.csv
|       +-- switzerland_inflation_monthly_clean.csv
|       +-- switzerland_inflation_monthly.csv
|
+-- results/
|   +-- comp_basic_vs_optimized.csv   # Clean + raw
|   +-- perf_basic_vs_optimized.csv   # Clean + raw
|   +-- optuna_trials.csv
|   +-- best_params.json
|   +-- *.png                         # Pie chart + wealth paths
|
+-- scripts/
|   +-- clean_fso_inflation_csv.py    # Inflation CSV cleaning
|
+-- src/
|   +-- models.py                 # Monte Carlo engine
|   +-- pipeline.py               # BASIC + OPTIMIZED orchestration
|   +-- optimization.py           # Optuna optimization logic
|   +-- compare_plots.py          # Plotting logic
|   +-- inflation.py              # Inflation computation logic
|   +-- data_fetcher.py           # yfinance data fetching
|   +-- data_loader.py            # CSV loading and validation
|   +-- regime.py                 # Regime bootstrapping
|   +-- build_dataset.py          # Monthly returns Dataset builder
|
+-- .gitignore                    # Github tracking limit framework
+-- AI_Usage.md
+-- environment.yml               # Dependencies
+-- main.py                       # CLI entry point
+-- Proposal.md
+-- README.md
```

Installation instructions

conda env create -f environment.yml
conda activate portfolio-project
- Python 3.11
- Numpy, Pandas, Matplotlib, Scikit-learn, Scipy, Jupyter, Optuna, Pip, Streamlit, Yfinance, Pandas-datareader

Reproducibility

Ensuring that random state is set to 42. To reproduce the exact results shown in this report you need an initial capital value of 1'200'000, an asset allocation of 50%S&P500 and 50%SMI and withdrawal amounts of 4000 for preferred and 3000 for floor. α is capped at 50%, withdrawal multiplier at 2. Inflation-aware withdrawals is enabled. For the various bootstrapping methods, you must set block to 12 and regime to 3 for the number of clusters, 12 for rolling volatility window and 24 for the minimum number of samples per regime. For the optimization you must ensure that block bootstrapping is selected as a bootstrapping mode *during* the Optuna run. The number of Optuna trials is 50 and the number of simulations is 1000. All these values except for the random state, are meant to be changed to try different scenarios.