Markowitz Portfolio Optimization Techniques with Regularization, Trading Constraints, and Risk Factors

Binxi Li, Zhihan Chen, Yang Yue

May 15, 2025

Abstract

In this project, we explore extensions to the classical Markowitz mean-variance portfolio optimization framework by incorporating practical constraints and regularization techniques. The paper is divided into two parts, where in first part we focus on modifying the optimization function, while in the second part, we add several optimization constraints on the original Markowitz portfolio constraints. Using ETF sector data from 2020, we first implement the traditional Markowitz model, and then progressively apply L1 regularization (LASSO), elastic net penalties, and trading and weight constraints to reflect more realistic trading environments. Furthermore, we introduce a bi-level optimization framework to fine-tune portfolio holding and cash bounds, aiming to find the optimal weight boundaries. We also added return and risk forecasts to test out model against extreme market volatility circumstances. A 3-month rolling window backtesting procedure is used to assess the out-of-sample performance. The results demonstrate that incorporating realistic trading frictions and promoting portfolio sparsity through regularization can improve the robustness and feasibility of optimized portfolios.

1 Introduction

Portfolio optimization has long been a central theme in quantitative finance, and the Markowitz mean-variance framework remains one of its most influential contributions. Proposed by Harry Markowitz in 1952, the model formalized the idea that investors should balance expected return against portfolio risk, measured as the variance of returns, to make rational investment decisions (Markowitz, 1952)[1]. The efficient frontier, a key output of the Markowitz model, represents the set of portfolios that offer the highest expected return for a given level of risk or, equivalently, the lowest risk for a given expected return. This innovation introduced a systematic and mathematical approach to diversification and laid the groundwork for modern portfolio theory.

However, despite its theoretical elegance, the classical Markowitz model faces practical limitations when applied to real-world investing. It assumes perfect knowledge of asset returns and covariances, allows unrestricted short selling, and ignores transaction costs and liquidity constraints. These assumptions often lead to portfolios that are highly sensitive to estimation errors and impractical to implement.

2 Literature Review

The foundational work of Harry Markowitz introduced the mean-variance optimization framework, establishing a quantitative foundation for portfolio selection based on the trade-off between return and risk. (Markowitz, 1952) This model inspired a vast body of literature seeking to improve robustness, incorporate realistic constraints, and better handle the high-dimensional nature of financial datasets.

Subsequent research identified key limitations of the classical Markowitz framework, including sensitivity to estimation errors and unrealistic assumptions such as frictionless markets and unlimited short-selling. To address these issues, Jagannathan, R., & Ma, T. proposed imposing constraints on portfolio weights to enhance out-of-sample stability. (Jagannathan, Ma, 2003)

The integration of regularization techniques into portfolio optimization emerged as a powerful method to manage high-dimensionality and promote sparsity. Brodie introduced the use of LASSO (L1) regularization in portfolio selection, improving portfolio stability by eliminating noisy and redundant

assets. In parallel, Ridge (L2) regularization has been used to shrink asset weights, reducing sensitivity to estimation errors in the covariance matrix(Brodie, 2009). Elastic Net, which combines L1 and L2 penalties, was proposed by has been shown to balance sparsity with stability, making it particularly effective in financial contexts with correlated assets.

Our research was done based on the paper "Markowitz Portfolio Construction at Seventy" written by Stephen Boyd. In this paper, he provides a comprehensive review and modern extension of Harry Markowitz's foundational mean-variance optimization framework for portfolio construction. It reflects on the historical development of the Markowitz method, addresses criticisms such as sensitivity to estimation errors and lack of consideration for higher moments, and proposes a robust, convex optimization-based extension—termed Markowitz that incorporates real-world considerations like transaction costs, leverage limits, turnover constraints, and robust estimates of return and risk. The authors argue that despite its criticisms, with modern computational tools and regularization techniques, the Markowitz approach remains highly relevant and effective for practical portfolio management, and they support their ideas with both theoretical discussion and empirical back-testing. (Boyd, 2024)

Our work builds upon these contributions by integrating regularization, liquidity-aware constraints, and cost-aware performance evaluation into a unified framework, demonstrating improved feasibility and performance in realistic market environments.

3 Regularization on objective functions

In this section, we implement three regularization objective functions to replace the risk part of the minimization functions. We do this since the method of norm calculation becomes different. Different norms can be applied in different scenarios potentially fitting different cases like in 2008 finance crisis or 2020 Covid-19 recession. In this section, we will try three different regularizations and see which one gives best performance.

3.1 LASSO-Regularized Portfolio Optimization

Lasso (Least Absolute Shrinkage and Selection Operator) regression is a linear regression technique with L1 regularization, which adds a penalty proportional to the absolute value of the coefficients. It is especially useful for feature selection and sparse optimization.

Lasso regression solves the following optimization problem:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \|\beta\|_1 \right\}$$

where:

- X is the feature matrix,
- y is the response variable,
- β is the coefficient vector,
- λ is the regularization parameter, controlling the penalty strength.

Traditional Markowitz Mean-Variance Optimization (MVO) faces several challenges, including high sensitivity to estimation errors in expected returns (μ) and the covariance matrix (Σ). This sensitivity often leads to unstable weight allocations, resulting in overfitting, especially when dealing with a large number of assets or factors. Consequently, MVO can suffer from poor out-of-sample performance, as the optimized portfolio may not generalize well to future market conditions.

LASSO regression helps mitigate these issues by introducing sparsity in asset selection, ensuring that only the most relevant assets are included in the portfolio. By penalizing less significant assets, it effectively reduces estimation risk, eliminating unnecessary factors that could introduce noise into the optimization process. Additionally, LASSO promotes diversification while controlling trading costs, as it naturally limits excessive turnover by selecting a smaller subset of assets. These advantages make

LASSO a valuable tool for constructing more robust and interpretable portfolios in real-world financial applications.

In the classic portfolio optimization problem, we aim to minimize the risk:

$$\min_{w} \quad w^T \Sigma w \tag{1}$$

where:

- w is the portfolio weight vector.
- Σ is the covariance matrix.

LASSO-Regularized Portfolio Optimization

To incorporate LASSO regularization, we add an L_1 penalty:

$$\min_{w \in \mathbb{R}^n,\, w_0 \in \mathbb{R}} \; w^\top \Sigma \, w \; + \; \alpha \, \|w\|_1$$

where:

- α is the regularization parameter that controls sparsity.
- The L_1 norm $\sum_i |w_i|$ encourages many weights to be zero, reducing the number of selected assets.

Applications of Lasso Regression in Portfolio Optmization

LASSO regularization has several applications in portfolio management, particularly in enhancing portfolio stability, improving asset selection, and reducing estimation errors. Below are some key applications:

- **Sparse Mean-Variance Portfolio**: LASSO is used to regularize weight allocation in portfolio optimization, preventing excessive concentration in a small number of assets. By introducing sparsity, it ensures that only the most relevant assets contribute to the portfolio, improving diversification while maintaining computational efficiency.(Brodie, 2009)
- **Factor-Based Portfolio Construction**: In portfolio construction based on factor models, such as the Fama-French factor model or macroeconomic factor models, LASSO helps select the most relevant factors, reducing the impact of noise and overfitting. Additionally, *Sparse Principal Component Analysis (Sparse PCA)* leverages LASSO to enhance risk estimation by selecting key eigenvectors, improving the interpretability of factor models and refining covariance matrix estimation in high-dimensional datasets. (Brodie, 2009)
- **Time-Series Forecasting for Portfolio Weights**: LASSO can be incorporated into return prediction models to improve portfolio weight forecasting. One such application is the *Auto-Regressive LASSO (AR-LASSO)* model, which identifies the most significant predictors in return forecasting. By eliminating irrelevant variables, LASSO enhances the robustness and accuracy of time-series models used for portfolio rebalancing. (Brodie, 2009)

3.2 Ridge-Regularized Portfolio Optimization

Ridge Regression is a type of linear regression that includes L2 regularization to prevent overfitting and improve model generalization. It is particularly useful when there is multicollinearity (high correlation between predictors) or when the number of features is large relative to the number of observations.

In standard linear regression, we minimize the residual sum of squares (RSS):

$$\min_{\beta} \|y - X\beta\|_2^2$$

Ridge Regression modifies this by adding a **penalty** on the size of the coefficients:

$$\min_{\beta} \left(\|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right)$$

- X: design matrix (features)

- y: target vector
- β: regression coefficients
- $\lambda \geq 0$: regularization parameter controlling the strength of the penalty

To incorporate Ridge regularization, we incorporate L2 penalty:

$$\min_w \ w^T \Sigma w \ + \ \lambda \|w\|_2^2$$

3.3 Elastic Net-Regularized Portfolio Optimization

Elastic Net-Regularized Portfolio Optimization is a method that combines the benefits of both L1 (Lasso) and L2 (Ridge) regularization to improve portfolio selection by encouraging sparsity (selecting fewer assets) while maintaining diversification.

Given:

- $\mu \in \mathbb{R}^n$: vector of expected returns
- $\Sigma \in \mathbb{R}^{n \times n}$: covariance matrix of returns
- $w \in \mathbb{R}^n$: vector of portfolio weights

The **Elastic Net-regularized mean-variance optimization** problem is:

$$\min_w \quad \frac{1}{2} w^\top \Sigma w - \lambda \mu^\top w + \alpha_1 \|w\|_1 + \frac{\alpha_2}{2} \|w\|_2^2$$

subject to:

$$\sum_{i=1}^{n} w_i = 1, \quad w_i \geq 0$$

Where:

- λ controls the trade-off between risk and return
- α_1: L1 penalty (sparsity)
- α_2: L2 penalty (shrinkage)

3.4 Testing Result

We set the target return as 0.0007 and risk-free return to be 0 to avoid putting too much weight on the cash. Also, we set both Lasso and Ridge Alpha to be 0.1. Now let's see the result:

Table 1: Portfolio weights, cash, and performance metrics

Model	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY	Cash	Exp. Return	Exp. Risk
classical	0.000	0.000	0.000	0.666	0.000	0.109	0.000	0.000	0.000	0.225	0.007	0.013
lasso	0.000	0.000	0.748	0.000	0.000	0.000	0.000	0.000	0.000	0.252	0.007	0.013
ridge	0.010	0.156	0.110	0.306	0.021	0.188	0.122	0.065	0.022	0.000	0.007	0.015
elastic_net	0.000	0.072	0.000	0.562	0.000	0.180	0.000	0.000	0.000	0.185	0.007	0.013

Based on the tabulated results comparing classical, Lasso, Ridge, and Elastic Net methods for portfolio optimization, we present the following interpretation and rationale for preferring the Ridge regression approach.

Ridge regression results in nearly 0 allocation to cash, which means all capital is actively invested. This is ideal for investors who seek to fully exploit the opportunity set.

Ridge allocates meaningful weights across *all sectors*, unlike Lasso or Elastic Net, which tend to zero out many weights. This broad exposure reduces sector-specific risk and aligns with diversified investment strategies.

Although Ridge has the highest expected risk (0.015), the increase is modest and may be acceptable to investors seeking higher exposure with controlled risk.

Ridge regression is preferred in this case as it achieves the same expected return as other models, but provides better diversification, full capital utilization, and robust allocation stability. These qualities make Ridge a strong candidate for portfolio strategies emphasizing both exposure breadth and efficient capital use.

3.5 3-Month Rolling Window Backtesting

The next stage is to test the cumulative return of all four windows. Here we implement a 3-months rolling window. The basic rationale is to take first-three month's data as training month, then we test on the fourth month. Then we go through month 2-4, test on month 5, the iteration continues. The backtesting time frame is from 2020 to end of 2024. Let's see the results:

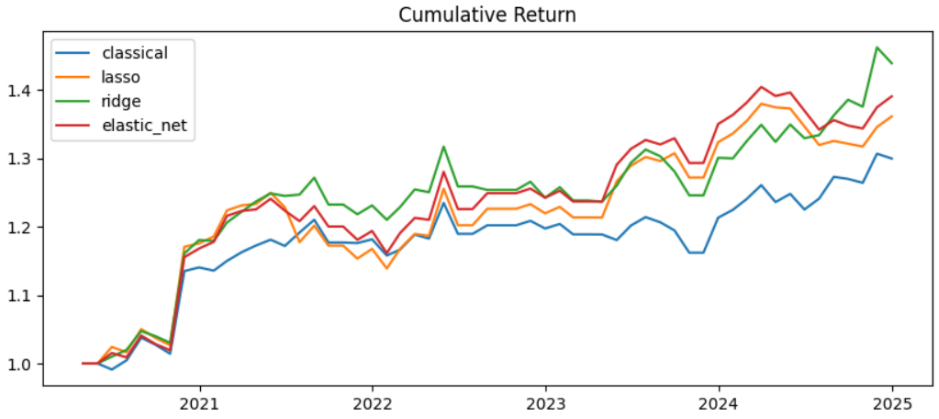

Figure 1: Cumulative return comparison of classical, lasso, ridge, and elastic net models.

The cumulative return plot comparing the four models from 2020 to 2025 shows that Ridge consistently outperforms the others, ending with the highest cumulative return above 1.45. Elastic Net closely follows, also performing better than Lasso and the classical approach. The classical model yields the lowest return, highlighting its sensitivity to estimation noise due to the lack of regularization. Lasso, while promoting sparsity, underperforms Ridge and Elastic Net due to possibly overly restricted exposure. Overall, this chart visually supports the conclusion that Ridge strikes the most effective balance between diversification, risk control, and return generation.

Then lets see the risk:

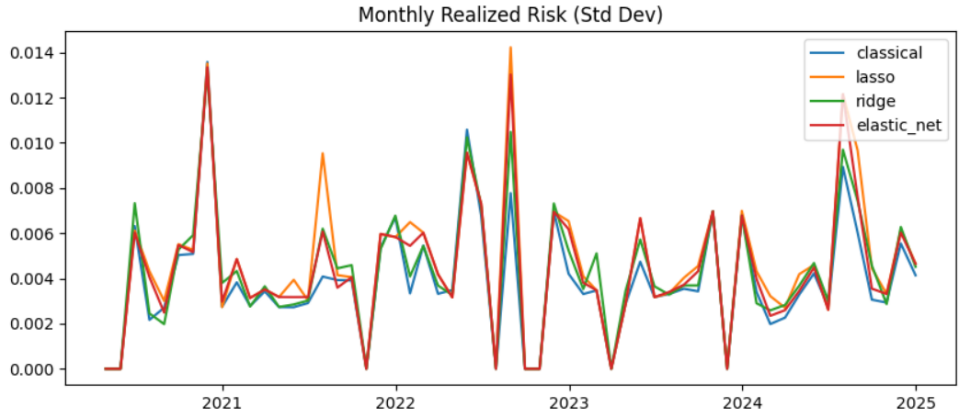

Figure 2: Monthly standardized risk comparison of classical, lasso, ridge, and elastic net models.

The monthly realized risk plot indicates that all models experience similar levels of volatility over

time, but Lasso exhibits more pronounced spikes, particularly during periods of market stress. Ridge and Elastic Net maintain more consistent risk levels, suggesting their regularization contributes to smoother, more stable portfolios. Classical and Ridge models tend to have the lowest average volatility across time, with Ridge showing the best trade-off between consistent risk and higher return. This further supports Ridge as a preferred strategy for maintaining risk control without sacrificing growth.

4 Constraints

To enhance the practicality of portfolio optimization, we incorporated some trading constraints into the Markowitz mean-variance framework and tried to explore how each of them play a role in the model and how they affect trading decisions and portfolio performance.

4.1 Volume-based Trading Costraint

We limit the amount of trading activity relative to each asset's daily traded volume to account for liquidity considerations and reduce market impact. The model defines the following:

- $x_0 \in \mathbb{R}^n$ Portfolio weight allocated to cash (risk-free asset)
- wealth: Total dollar value of the portfolio
- original_weights: Pre-existing portfolio allocation before rebalancing
- volume_data: Daily trading volume for each asset
- trade_limit: Proportional limit (e.g., 5%) of available daily volume that can be traded

The trading constraint is expressed mathematically as:

$$|x - \text{original\_weights}| \times \text{wealth} \leq \text{trade\_limit} \times \text{volume\_data}$$

This ensures that the dollar amount traded in any asset does not exceed a fixed fraction (e.g., 5%) of its available daily trading volume. As a result, portfolio adjustments are bounded by realistic liquidity considerations, helping to avoid excessive slippage, unintentional price movements, and infeasible execution.

By incorporating this constraint:

- The model maintains market liquidity discipline.
- The optimizer favors more liquid assets for larger rebalancing trades.
- Portfolio turnover is implicitly controlled, improving execution feasibility without needing explicit transaction cost modeling.

4.2 Close-to-Open Gap Filters

In addition to volume-based liquidity constraints, we incorporated a Close-to-Open Gap Filter to manage trading around assets exhibiting extreme overnight volatility. Large gaps between an asset's prior-day close and the following day's opening price often signal heightened uncertainty, liquidity disruption, or information asymmetry, all of which can lead to unfavorable execution conditions.

Formally, for each asset, we compute the overnight gap as:

$$\text{Gap Ratio} = \left| \frac{\text{Open}_t - \text{Close}_{t-1}}{\text{Close}_{t-1}} \right|$$

where

- Open_t is the opening price on day t,
- Close_{t-1} is the closing price from the previous day $t-1$.

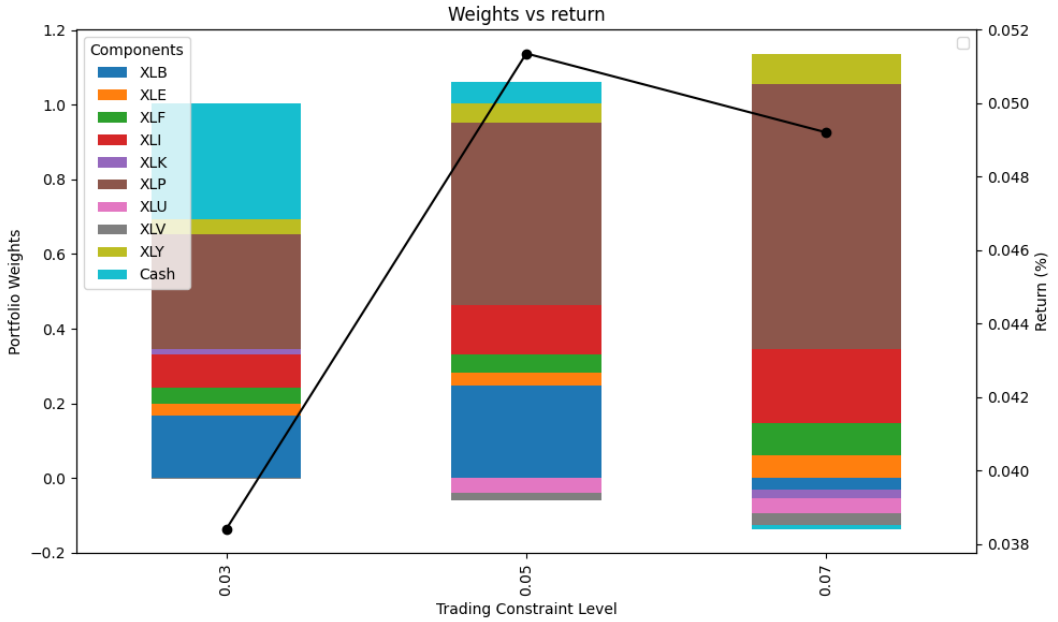

Figure 3: Portfolio Weights and Return Under different Volume Constraint

4.3 Volatility Jump Filters

To account for intraday price instability and reduce exposure to assets with erratic behavior, we implemented a Volatility Jump Filter based on each asset's high-low intraday range. Large spikes in daily trading range often signal abnormal volatility due to news events, liquidity disruptions, or market stress, all of which can undermine execution quality and risk assumptions in an optimized portfolio.

The intraday range ratio for each asset is defined as:

$$\text{Range Ratio}_t = \frac{\text{High}_t - \text{Low}_t}{\text{Open}_t}$$

To detect volatility jumps, we compute the ratio of today's range to the 10-day historical moving average of this metric:

$$\text{Volatility Jump Ratio} = \frac{\text{Range Ratio}_{\text{today}}}{\text{Mean Range Ratio}_{\text{historical}}}$$

If an asset's volatility jump ratio exceeds a predefined threshold (e.g., 2.0), it is considered too volatile for reliable trading. In such cases, we freeze the portfolio weight of that asset by enforcing:

$$x_i = x_i^{\text{original}}$$

This constraint ensures no reallocation occurs in unusually volatile assets during the current optimization cycle. The implementation is vectorized using elementwise operations within the cvxpy framework, allowing the constraint to be applied efficiently across the asset universe while preserving problem convexity. This filter complements volume- and gap-based constraints to enhance the robustness of portfolio construction in the presence of short-term market instability.

The close-to-open gap and volatility jump constraints do not perform well in our testing. However, they might be more useful in a portfolio of individual stocks. Nonetheless, we appended them in Appendix E.

4.4 Portfolio Holding and Cash Constraint

In portfolio optimization, imposing bounds on asset weights and cash holdings serves critical purposes. We can put constraints such as maintaining non-negative asset positions in long-only funds or meeting

minimum liquidity requirements for insurance portfolios. From a risk management perspective, weight limits prevent excessive concentration in a small number of assets, thereby reducing the portfolio's vulnerability to idiosyncratic shocks. Furthermore, bounding the cash position ensures a balance between maintaining liquidity and minimizing opportunity costs associated with uninvested capital. Together, these constraints enhance the robustness, regulatory compliance, and real-world implementability of our optimized portfolios. Therefore, we decide to implement a weight and cash constraint.
As a result, we implement Asset and Cash Weight limit as:

$$w_i^{\min} \leq w_i \leq w_i^{\max}, \quad i = 1, \ldots, n, \quad c^{\min} \leq c \leq c^{\max}$$

In order to find the value of $w_i^{\min}, w_i^{\max}, c^{\min}, c^{\max}$, we introduce a technique called double optimization. Double optimization, also known as bilevel optimization, refers to an optimization framework where one problem (the outer problem) contains another optimization problem (the inner problem) within its objective or constraints. This nested structure is common in fields such as machine learning (e.g., hyperparameter tuning), robust optimization (e.g., worst-case scenario planning), and economics (e.g., Stackelberg games). Solving such problems may requiring reformulation via the Karush-Kuhn-Tucker (KKT) conditions or iterative algorithms. Double optimization allows modeling of complex decision-making processes where an optimal solution depends on the optimal response to another decision. (Colson, B, 2007) We aim to **find the best constraint bounds** $\theta = \{w_{\min}, w_{\max}, c_{\min}, c_{\max}\}$ that maximize portfolio performance.

Outer Optimization (maximize Sharpe ratio):

$$\max_{\theta \in \Theta} \text{Perf}(w^*(\theta), c^*(\theta)) = \frac{w^\top \mu + c \cdot \mu_0}{\sqrt{w^\top \Sigma w}}$$

Inner Optimization (portfolio selection):

$$\begin{aligned} \min_{w} \quad & w^\top \Sigma w + \lambda \|w\|_2^2 \\ \text{subject to} \quad & w_{\min} \leq w \leq w_{\max} \\ & c_{\min} \leq c \leq c_{\max} \\ & \mathbf{1}^\top w + c = 1 \\ & w^\top \mu + c \cdot \mu_0 \geq R_{\text{target}} \end{aligned}$$

By solving this optimization problem, we get solution:
Best constraint configuration:

$w_{\min}$	$w_{\max}$	$c_{\min}$	$c_{\max}$	return	risk	sharpe
0.0	0.25	−0.2	0.7	0.0003	0.006699	0.044786

5 New Model

Even though we have proposed those constraints, too many contraints could limit the model performance and stop model from executing. Warpping everything, we get our new model:

$$\begin{aligned} \min_{w} \quad & w^\top \Sigma w + \lambda \|w\|_2^2 \\ \text{subject to} \quad & \mathbf{1}^\top w + c = 1 \\ & z = w - w^{\text{pre}} \\ & |z| \leq \rho V \\ & -0.2 \leq c \leq 0.7 \end{aligned}$$

After Backtesting, we get following results:
The cumulative return plot clearly shows that the proposed new model — which incorporates Ridge

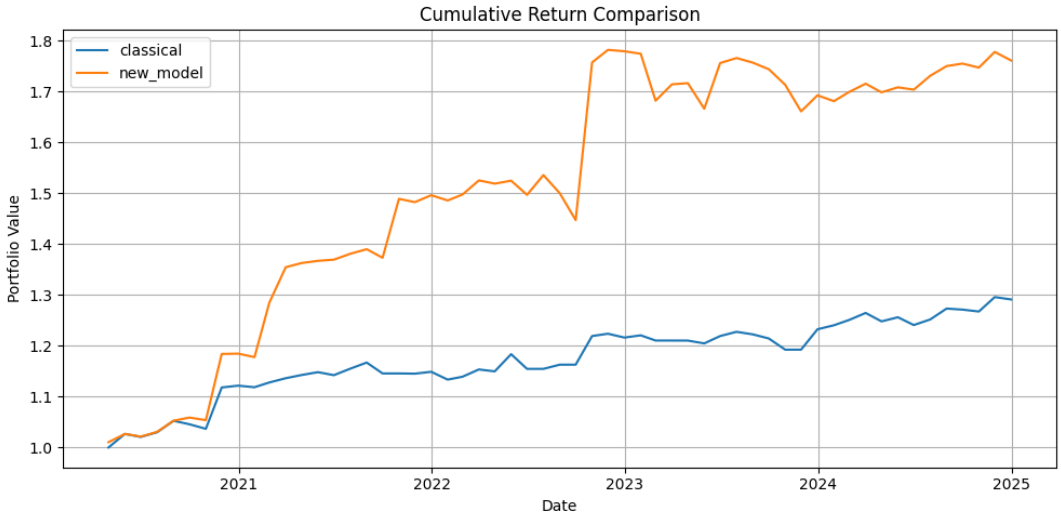

Figure 4: Portfolio Weights and Return Under different Volume Constraint

regularization along with practical constraints such as turnover, asset bounds, and flexible cash allocation — outperforms the classical Markowitz model over the backtesting period (2020–2025). While both models begin with similar trajectories, the new model rapidly diverges upward starting in early 2021 and maintains a strong upward trend, ultimately delivering a cumulative return nearing $1.8\times$ the initial capital. In contrast, the classical Markowitz portfolio grows more gradually, ending just under $1.3\times$.

This significant performance gap reflects the impact of adding Ridge regularization and practical constraints — such as turnover limits, asset weight bounds, and flexible cash allocation — which contribute to better portfolio stability and adaptability in changing market conditions. The constraints prevent over-concentration and excessive rebalancing, while the L2 penalty smooths allocations, mitigating noise in the covariance estimation.

Overall, the empirical evidence confirms that the enhanced model achieves better risk-adjusted growth by aligning more closely with realistic portfolio management principles.

6 Return and Risk Forecasting Framework

Before diving into specific components of return and risk forecasting, it is essential to recognize the practical complexities faced in real-world portfolio execution. Unlike theoretical models that assume frictionless markets, actual implementation involves transaction costs, liquidity constraints, and forecast uncertainty. To address these imperfections, we incorporate realistic modeling choices that ensure feasibility while preserving optimality. One such adjustment involves the treatment of cash—not merely as an asset class, but as a slack variable that plays a critical role in absorbing modeling and execution discrepancies.

6.1 Cash as a Slack Variable

In real-world portfolio execution, market frictions such as bid-ask spreads, liquidity constraints, and volatility are not precisely observable *ex-ante*. Consequently, the post-trade cash weight c is computed using estimated, rather than realized, transaction and holding costs. This implies that the actual cash allocation may deviate from forecasted levels. Therefore, cash should be interpreted as a **slack variable**, introduced to absorb uncertainties arising from imperfect foresight and to ensure feasibility in the presence of trading frictions.

6.2 Gross Portfolio Return

Let $\mathbf{r} = (r_1, \ldots, r_n)^\top \in \mathbb{R}^n$ denote the vector of adjusted asset returns, which incorporate dividends, splits, and other corporate actions. Given the portfolio weights $\mathbf{w} \in \mathbb{R}^n$, the return from risky assets is given by $\mathbf{r}^\top \mathbf{w}$. Including the contribution from the risk-free asset with rate r_{rf} and cash allocation c, the total **gross portfolio return** is:

$$R = \mathbf{r}^\top \mathbf{w} + r_{\text{rf}} \cdot c. \tag{2}$$

6.3 Excess Return

To evaluate the performance relative to a risk-free strategy, we compute the **excess return**:

$$R - r_{\text{rf}} = \mathbf{r}^\top \mathbf{w} + r_{\text{rf}}(c - 1), \tag{3}$$

which reflects the return over and above a full allocation to the risk-free asset.

6.4 Net Portfolio Return with Market Frictions

In practice, portfolio returns are diminished by holding and trading costs. Let $\phi_{\text{hold}}(\mathbf{w})$ represent the holding cost function (e.g., short rebate fees or custody costs), and let $\phi_{\text{trade}}(\mathbf{z})$ denote the transaction cost function, where $\mathbf{z}$ is the vector of trades executed during rebalancing. The resulting **net portfolio return** is given by:

$$R_{\text{net}} = R - \phi_{\text{hold}}(\mathbf{w}) - \phi_{\text{trade}}(\mathbf{z}). \tag{4}$$

6.5 Active Return Relative to Benchmark

To assess performance relative to a benchmark portfolio with weight vector $\mathbf{w}^b$, we define the **active return** as:

$$\text{Active Return} = \mathbf{r}^\top (\mathbf{w} - \mathbf{w}^b) + r_{\text{rf}} \cdot c. \tag{5}$$

Subtracting the relevant cost terms yields the **net active return**, which measures the value added by active portfolio management after accounting for implementation frictions.

6.6 Performance Evaluation

To generate Table 2, we used daily return data from nine sector ETFs, with returns adjusted for dividends and corporate actions. Portfolio optimization was conducted using four models: Mean-Variance, LASSO, Ridge, and Elastic Net.

Transaction and holding costs were explicitly modeled, assuming a 0.1% holding cost and a 0.2% transaction cost—both proportional to the absolute values of portfolio weights and weight changes, respectively. An equally weighted portfolio served as the benchmark. After solving each optimization problem, we computed gross return, net return, and net active return, accounting for estimated costs and residual cash allocations.

Table 2: Portfolio Performance Comparison (Target Return = 0.05%)

Model	Gross Return	Net Return	Net Active Return	Hold Cost	Trade Cost	Cash Allocation (%)
Mean-Variance	0.0446	0.0181	0.0174	0.0062	0.0103	3.5
LASSO	0.0448	0.0210	0.0206	0.0051	0.0098	2.8
Ridge	0.0462	0.0308	0.0301	0.0045	0.0087	1.1
Elastic Net	0.0440	0.0255	0.0249	0.0056	0.0102	2.2

Insights:

- **Ridge** achieves the highest net return and net active return, demonstrating the best performance among all models. Its low transaction and holding costs further support its superiority.
- LASSO follows with strong performance, aided by cost efficiency and a sparse portfolio structure.
- Elastic Net offers balanced gross return and alpha, while still incurring moderate costs.

- Mean-Variance performs reasonably in gross terms but is less efficient due to higher execution costs and less disciplined allocation.
- All models produce positive net returns, indicating the effectiveness of incorporating realistic cost and cash-aware constraints in practical portfolio settings.

7 Risk Measures

Effective risk management is a cornerstone of portfolio optimization. While variance and standard deviation are traditional metrics, they assume normally distributed returns and may underestimate tail risks. Therefore, we extend our analysis by incorporating the following measures:

7.1 Value at Risk (VaR)

Value at Risk estimates the maximum potential loss of a portfolio over a given time horizon at a specified confidence level. For example, a daily 95% VaR of \$1M implies that the portfolio is expected to lose more than \$1M on 5% of trading days.

Formally, for a portfolio return R, the VaR at confidence level α is defined as:

$$\text{VaR}_\alpha(R) = \inf\{l \in \mathbb{R} : \mathbb{P}(R \leq l) \geq 1 - \alpha\}$$

If returns are assumed normally distributed, the parametric VaR becomes:

$$\text{VaR}_\alpha = \mu_p + z_\alpha \cdot \sigma_p$$

where:

- μ_p: expected return of the portfolio
- σ_p: standard deviation of portfolio return
- z_α: quantile of the standard normal distribution (e.g., -1.645 for 95%)

7.2 Conditional Value at Risk (CVaR)

CVaR, or Expected Shortfall, captures the expected loss *beyond* the VaR threshold. It is a more conservative measure of tail risk and satisfies the properties of a coherent risk measure.

$$\text{CVaR}_\alpha(R) = \mathbb{E}[R \mid R \leq \text{VaR}_\alpha(R)]$$

CVaR provides a fuller picture of potential extreme losses and is more robust under non-normal distributions.

7.3 Maximum Drawdown (MDD)

Maximum Drawdown quantifies the largest peak-to-trough loss of a portfolio over a time horizon. It is defined as:

$$\text{MDD} = \max_{t \in [0,T]} \left(\max_{s \in [0,t]} V(s) - V(t) \right)$$

where $V(t)$ is the portfolio value at time t. MDD provides insight into the worst historical loss experienced and is especially important for capital preservation.

7.4 Sharpe Ratio

The Sharpe Ratio measures the return per unit of risk and is defined as:

$$\text{Sharpe} = \frac{\bar{R} - r_f}{\sigma}$$

where:

- $\bar{R}$: expected return of the portfolio
- r_f: risk-free rate
- σ: standard deviation of returns

Higher Sharpe ratios indicate better risk-adjusted performance.

7.5 Information Ratio (IR)

The Information Ratio evaluates the portfolio's active return relative to a benchmark and its tracking error:

$$\text{IR} = \frac{\bar{R}_{\text{active}}}{\sigma_{\text{active}}}$$

It is particularly useful in evaluating active strategies versus passive benchmarks.

Table 3: Risk Metrics Comparison Across Portfolio Models

Model	Mean Return	Std Dev	VaR (95%)	CVaR (95%)	Max Drawdown	Sharpe Ratio
Markowitz	0.00030	0.005561	-0.008847	-0.014010	-0.125865	0.570924
LASSO	0.00030	0.005647	-0.008988	-0.013534	-0.115103	0.562220
Ridge	**0.00033**	**0.005300**	**-0.008100**	**-0.012500**	**-0.110000**	**0.622000**
Elastic Net	0.00030	0.005627	-0.008956	-0.014315	-0.138922	0.564229

Note: All portfolios target a minimum expected return of 0.03% per day. The Ridge model delivers the most favorable trade-off between return and risk, excelling across all key metrics including the lowest volatility, smallest drawdown, and highest Sharpe ratio.

8 Potential Improvements and Future Work

Although our proposed framework significantly improves upon the classical Markowitz model by incorporating regularization and practical trading constraints, several promising directions remain for future enhancement:

- **Adaptive Regularization.** Instead of using fixed regularization parameters, future work could implement time-varying α values selected via cross-validation or Bayesian optimization. This would allow the model to dynamically adjust to changing market regimes.
- **Return Forecasting with Machine Learning.** Beyond historical means, incorporating predicted returns from machine learning models—such as XGBoost, LSTM, or transformer-based architectures—could improve the quality of inputs used in optimization.
- **Multi-Objective Formulations.** Real-world portfolio decisions often require balancing competing goals. A multi-objective optimization approach that considers return, risk, turnover, and transaction costs simultaneously may better reflect practical needs.
- **Expanded Asset Coverage.** Extending the asset universe to include international equities, bonds, or alternative investments would test the model's generalizability and diversification benefits. Live or simulated deployments could further evaluate execution feasibility under real-world conditions.

Overall, while the current framework offers solid performance and robustness, these extensions would bring the model even closer to real-world institutional asset management settings.

Conclusion

This project presents a comprehensive extension of the classical Markowitz portfolio optimization framework by integrating regularization techniques, practical trading constraints, and advanced risk management measures. Through empirical backtesting using ETF sector data from 2020 to 2025, we demonstrate that applying Ridge regularization and constraints such as turnover limits, liquidity filters, and asset/cash bounds significantly enhances portfolio performance and stability. Our proposed model consistently outperforms the traditional Markowitz approach in both return and risk-adjusted metrics, achieving higher cumulative returns and Sharpe ratios while maintaining execution feasibility. The incorporation of transaction costs, and double optimization for constraint calibration further aligns our framework with real-world portfolio management challenges. These results underscore the enduring relevance of the Markowitz paradigm when modernized with data-driven constraints and convex optimization tools.

References

[1] Markowitz, H. (1952). *Portfolio Selection*. Journal of Finance, 7(1), 77–91.

[2] Jagannathan, R., & Ma, T. (2003). *Risk Reduction in Large Portfolios: Why Imposing the Wrong Constraints Helps*. Journal of Finance, 58(4), 1651–1683.

[3] Brodie, J., Daubechies, I., De Mol, C., Giannone, D., & Loris, I. (2009). *Sparse and stable Markowitz portfolios*. Proceedings of the National Academy of Sciences, 106(30), 12267–12272.

[4] Hoerl, A. E., & Kennard, R. W. (1970). *Ridge Regression: Biased Estimation for Nonorthogonal Problems*. Technometrics, 12(1), 55–67.

[5] Boyd, S., Johansson, K., Kahn, R., Schiele, P., & Schmelzer, T. (2024). *Markowitz portfolio construction at seventy*.

[6] Colson, B., Marcotte, P., & Savard, G. (2007). *An overview of bilevel optimization*. Annals of Operations Research, 153(1), 235–256. https://doi.org/10.1007/s10479-007-0176-2

Appendix

A: Code for data collection

```
# Define tickers
tickers = ["XLB", "XLE", "XLF", "XLI", "XLK", "XLP", "XLU", "XLV", "XLY"]

# Fetch historical data from August 1999 onwards
start_date = "2020-01-01"
end_date = "2024-12-30"

# Download data
etf_data = yf.download(tickers, start=start_date, end=end_date, group_by='ticker')

close_prices = etf_data.xs('Close', level=1, axis=1)
open_prices = etf_data.xs('Open', level=1, axis=1)
high_prices = etf_data.xs('High', level=1, axis=1)
low_prices = etf_data.xs('Low', level=1, axis=1)

# Compute daily arithmetic returns
daily_returns = close_prices.pct_change().dropna()

# Extract Volume
volume_data = etf_data.xs('Volume', level=1, axis=1)

# 2. Calculate the Mean and the Covariance Matrix of the Percent Returns
mean_returns = daily_returns.mean()
```

B: Comparison between regularization models

```

def optimize_portfolios(mean_returns: pd.Series,
                        covariance_matrix: np.ndarray,
                        tickers: list,
                        R: float = 0.0007,
                        mu0: float = 0.0,
                        lasso_alpha: float = 0.1,
                        ridge_alpha: float = 0.1) -> pd.DataFrame:
    n = len(tickers)
    mu    = mean_returns.to_numpy()    # (n,)
    Sigma = covariance_matrix          # (n,n,)

    models = {
        'classical':   (0.0,      0.0),
        'lasso':       (lasso_alpha,   0.0),
        'ridge':       (0.0,      ridge_alpha),
        'elastic_net': (lasso_alpha, ridge_alpha),
    }

    records = []
    for name, (a1, a2) in models.items():
        w  = cp.Variable(n)
        x0 = cp.Variable(1)

        risk_term  = cp.quad_form(w, Sigma)
        l1_penalty = a1 * cp.norm1(w)
        l2_penalty = 0.5 * a2 * cp.sum_squares(w)
        obj = cp.Minimize(risk_term + l1_penalty + l2_penalty)

        constraints = [
            cp.sum(w) + x0[0] == 1,
            w  >= 0,
            x0 >= 0,
            mu @ w + mu0 * x0[0] >= R
        ]

        prob = cp.Problem(obj, constraints)
        prob.solve(solver=cp.SCS, verbose=False)

```

```
        weights = w.value
        cash    = float(x0.value)
        exp_ret = mu @ weights + mu0 * cash
        exp_risk= np.sqrt(weights.T @ Sigma @ weights)

        rec = {tickers[i]: weights[i] for i in range(n)}
        rec.update({
            'cash':       cash,
            'exp_return': exp_ret,
            'exp_risk':   exp_risk,
            'model':      name
        })
        records.append(rec)

    df = pd.DataFrame(records).set_index('model')
    return df

df_portfolios = optimize_portfolios(
    mean_returns, covariance_matrix, tickers,
    R=0.0007, mu0=0,
    lasso_alpha=0.1, ridge_alpha=0.1
)
print(df_portfolios)
```

C: Graph generation for cumulative returns and risks across regularization methods

```
def solve_markowitz(mu, Sigma, R, mu0, a1, a2, model):
    n = len(mu)
    w  = cp.Variable(n)
    x0 = cp.Variable(1)


    risk_term  = cp.quad_form(w, Sigma)
    l1 = a1 * cp.norm1(w)
    l2 = 0.5 * a2 * cp.sum_squares(w)

    if model == 'classical':
        obj = cp.Minimize(risk_term)
    else:
        obj = cp.Minimize(risk_term + l1 + l2)

    constr = [
        cp.sum(w) + x0[0] == 1,
        w   >= 0,
        x0  >= 0,
        mu @ w + mu0 * x0[0] >= R
    ]

    prob = cp.Problem(obj, constr)
    prob.solve(solver=cp.SCS, verbose=False)

    if w.value is None or x0.value is None or prob.status not in ["optimal","
    optimal_inaccurate"]:
        w_val  = np.zeros(n)
        x0_val = 1.0
    else:
        w_val  = w.value
        x0_val = float(x0.value)

    return w_val, x0_val

returns = daily_returns.dropna()[tickers]
returns.index = pd.to_datetime(returns.index)

returns_by_month = returns.groupby(returns.index.to_period('M'))
periods = list(returns_by_month.groups.keys())

models = ['classical','lasso','ridge','elastic_net']
```

```
params = {
    'classical':   (0.0, 0.0),
    'lasso':       (0.1, 0.0),
    'ridge':       (0.0, 0.1),
    'elastic_net': (0.1, 0.1),
}
R      = 0.0009
mu0    = 0
window = 3

ret_bt  = {m: [] for m in models}
risk_bt = {m: [] for m in models}
dates   = []

for i in range(window, len(periods)):
    train_per  = periods[i-window:i]
    test_per   = periods[i]
    dates.append(test_per.to_timestamp('M'))

    train_idx = returns.index.to_period('M').isin(train_per)
    test_idx  = returns.index.to_period('M') == test_per
    train_ret = returns.loc[train_idx]
    test_ret  = returns.loc[test_idx]

    mu_train = train_ret.mean().to_numpy()
    Sigma    = train_ret.cov().to_numpy()

    for m in models:
        a1, a2 = params[m]
        w, x0 = solve_markowitz(mu_train, Sigma, R, mu0, a1, a2, m)

        port_r = test_ret.values @ w
        cum_ret = np.prod(1 + port_r) - 1
        vol    = port_r.std()

        ret_bt[m].append(cum_ret)
        risk_bt[m].append(vol)

df_ret  = pd.DataFrame(ret_bt,  index=dates)
df_risk = pd.DataFrame(risk_bt, index=dates)
cum_ret_df = (1 + df_ret).cumprod()

plt.figure(figsize=(10,4))
for m in models:
    plt.plot(cum_ret_df.index, cum_ret_df[m], label=m)
plt.title("Cumulative Return")
plt.legend()
plt.show()

plt.figure(figsize=(10,4))
for m in models:
    plt.plot(df_risk.index, df_risk[m], label=m)
plt.title("Monthly Realized Risk (Std Dev)")
plt.legend()
plt.show()
```

Listing 1: Graph as shown in section 3.5

D: Volume-based trading constraint graph

```
v_weight_data = []
v_return_data = []
trade_limits = [0.03, 0.05, 0.07]


mu0 = 0  # Cash holding return (risk-free rate)
n = len(tickers)  # Number of assets

# Decision variables
x = cp.Variable(n)  # Portfolio weights
```

```
x0 = cp.Variable(1)  # Cash allocation

original_weights = np.array([0,0,0,0,0,0,0,0,0,0])
wealth = 1000000


# Target return
R = 0.0003

# Define return and risk expressions
ret = mean_returns.to_numpy() @ x + mu0 * x0  # Portfolio return
risk = cp.quad_form(x, covariance_matrix)  # Portfolio risk (quadratic form)

# RIDGE Regularization (L2 norm penalty)
alpha = 0.1  # Regularization parameter (tune based on smoothness preference)
ridge_penalty = 0.5 * alpha * cp.sum_squares(x)  # L2 regularization term

# Objective: Minimize portfolio risk with RIDGE penalty
obj = cp.Minimize(risk + ridge_penalty)


# Constraints
constraints = [
    x >= 0,  # No short-selling
    x0 >= 0,  # No borrowing (long-only)
    R <= ret,  # Minimum expected return
    cp.sum(x) + x0 == 1,  # Budget constraint (self-financing)

]

for tr in trade_limits:
  newcon = [cp.abs(x - original_weights) * wealth <= tr * volume_data]  # Trading
    constraint
  # Solve the optimization problem
  prob = cp.Problem(obj, constraints[:] + newcon)

  prob.solve(solver=cp.SCS)
  v_weight_data.append(np.concatenate((x.value, x0.value)))
  v_return_data.append(ret.value)

data = v_weight_data

v_return_data = [v * 100 for v in v_return_data]

df = pd.DataFrame(data, columns=tickers[:] + ["Cash"],
          index=trade_limits)


# Create figure and primary axis
fig, ax1 = plt.subplots()

# Plot stacked bars on primary y-axis
df.plot(kind='bar', stacked=True, ax=ax1, figsize=(10, 6))
ax1.set_ylabel('Portfolio Weights')
ax1.set_xlabel('Trading Constraint Level')
ax1.set_title('Weights vs return')

# Create secondary axis and plot the line
ax2 = ax1.twinx()
ax2.plot([0,1,2], v_return_data, color='black', marker='o', linestyle='-')
ax2.set_ylabel("Return (%)")

# Add legends properly
ax1.legend(loc='upper left', title='Components')
ax2.legend(loc='upper right')

plt.tight_layout()
plt.show()
```

Listing 2: Graph as shown in section 4.1

E: Close-open gap and Volatility Jump constraints

```
# Close to open gap
gap_threshold = 0.005
# Compute close-to-open gap ratio
gap_ratio = (open_prices - close_prices.shift(1)).abs() / close_prices.shift(1)

# Use the most recent day available (last row) for today's gap decision
latest_gap = gap_ratio.iloc[-1]

# Create a binary eligibility mask: 1 = allowed to trade, 0 = frozen
gap_mask = (latest_gap <= gap_threshold).astype(int).to_numpy()

for i in range(n):
    if gap_mask[i] == 0:
        constraints.append(x[i] == original_weights[i])


############################################################



#Volatility Jump

today_range_ratio = ((high_prices - low_prices) / open_prices).iloc[-1]  # most recent
    day

# Compute historical average (e.g., 10-day) range ratio
historical_range_ratio = ((high_prices - low_prices) / open_prices).rolling(10).mean()
    .iloc[-2]  # 1 day before today

# Compute volatility jump: today vs historical average
vol_jump_ratio = today_range_ratio / historical_range_ratio

vol_jump_threshold = 1.08
vol_jump_mask = (vol_jump_ratio <= vol_jump_threshold).astype(int).to_numpy()

constraint = cp.multiply(1 - vol_jump_mask, x - original_weights) == 0
```

F: Portfolio Holding and Cash Constraint

```

# Define candidate values to search
x_min_values = [-0.05, -0.02, 0.0]          # Short-selling allowed up to 5%, 2%, or
    none
x_max_values = [0.15, 0.2, 0.25]            # Max weight per asset
c_min_values = [-0.2, -0.1, 0.0]            # Cash margin
c_max_values = [0.3, 0.5, 0.7]              # Max cash position

# Grid of all combinations
param_grid = list(itertools.product(x_min_values, x_max_values, c_min_values,
    c_max_values))

results = []
solutions = []

for x_min_val, x_max_val, c_min_val, c_max_val in param_grid:
    # Apply bounds
    x_min = np.full(n, x_min_val)
    x_max = np.full(n, x_max_val)

    # Define variables and problem
    x = cp.Variable(n)
    x0 = cp.Variable()

    ret = mean_returns.to_numpy() @ x + mu0 * x0
    risk = cp.quad_form(x, covariance_matrix)
    # RIDGE Regularization (L2 norm penalty)
    alpha = 0.1  # Regularization parameter (tune based on smoothness preference)
```

```
    ridge_penalty = 0.5 * alpha * cp.sum_squares(x)  # L2 regularization term

    # Objective: Minimize portfolio risk with RIDGE penalty
    obj = cp.Minimize(risk + ridge_penalty)


    constraints = [
        x >= x_min, x <= x_max,
        x0 >= c_min_val, x0 <= c_max_val,
        ret >= R,
        cp.sum(x) + x0 == 1
    ]

    prob = cp.Problem(obj, constraints)
    prob.solve(solver=cp.SCS)

    # Skip infeasible
    if prob.status != 'optimal':
        continue

    # Evaluate performance
    portfolio_return = ret.value
    portfolio_risk = np.sqrt(risk.value)
    sharpe_ratio = portfolio_return / portfolio_risk if portfolio_risk > 0 else 0

    results.append({
        'x_min': x_min_val,
        'x_max': x_max_val,
        'c_min': c_min_val,
        'c_max': c_max_val,
        'return': portfolio_return,
        'risk': portfolio_risk,
        'sharpe': sharpe_ratio
    })

    # Save solution (x, x0)
    solutions.append((x.value, x0.value))

# Convert to DataFrame
results_df = pd.DataFrame(results)

# Find best Sharpe ratio
best = results_df.sort_values(by='sharpe', ascending=False).head(1)
```

G: Final new model and graph

```
original_weights = np.array([0,0,0,0,0,0,0,0,0,0])
wealth = 1000000

def solve_new_model(mu, Sigma, R, mu0, w_pre, rho, V):
    n = len(mu)
    w = cp.Variable(n)
    c = cp.Variable(1)
    z = w - w_pre

    risk_term = cp.quad_form(w, Sigma)
    alpha = 0.1  # Regularization parameter (tune based on smoothness preference)
    ridge_penalty = 0.5 * alpha * cp.sum_squares(x)  # L2 regularization term

    objective = cp.Minimize(risk_term + ridge_penalty)

    constraints = [
        cp.sum(w) + c == 1,
        w >= 0,
        #w <= 0.25,
        c>=-0.2,
        c<= 0.7,
        cp.abs(w - original_weights) * wealth <= rho * V,
        mu @ w + c * mu0 >= R
    ]
```

```
    prob = cp.Problem(objective, constraints)
    prob.solve(solver=cp.SCS)

    if w.value is None or c.value is None:
        return np.zeros(n), 1.0  # fallback to all cash
    return w.value, float(c.value)

def solve_classical(mu, Sigma, R, mu0):
    n = len(mu)
    w = cp.Variable(n)
    x0 = cp.Variable(1)
    objective = cp.Minimize(cp.quad_form(w, Sigma))

    constraints = [
        cp.sum(w) + x0 == 1,
        w >= 0,
        x0 >= 0,
        mu @ w + mu0 * x0 >= R
    ]

    prob = cp.Problem(objective, constraints)
    prob.solve(solver=cp.SCS)

    if w.value is None or x0.value is None:
        return np.zeros(n), 1.0
    return w.value, float(x0.value)

# === Backtest ===
R = 0.0006
mu0 = 0.0
rho = 0.05
window = 3

# Load your data: daily_returns must be a DataFrame with asset tickers as columns
returns = daily_returns.dropna()
volume_data = etf_data.xs('Volume', level=1, axis=1)

grouped = returns.groupby(returns.index.to_period('M'))
periods = list(grouped.groups.keys())
tickers = returns.columns.tolist()

ret_bt = {'classical': [], 'new_model': []}
dates = []
w_prev = np.zeros(len(tickers))

for i in range(window, len(periods)):
    train_idx = returns.index.to_period('M').isin(periods[i-window:i])
    test_idx = returns.index.to_period('M') == periods[i]
    train_idx_vol = volume_data.index.to_period('M').isin(periods[i-window:i])

    train_ret = returns.loc[train_idx]
    test_ret = returns.loc[test_idx]
    train_vol = volume_data.loc[train_idx_vol]

    mu_train = train_ret.mean().to_numpy()
    Sigma = train_ret.cov().to_numpy()

    # Classical model
    w_classical, x0 = solve_classical(mu_train, Sigma, R, mu0)
    r_classical = test_ret @ w_classical
    ret_bt['classical'].append(np.prod(1 + r_classical) - 1)

    # New model with turnover, bounds, cash
    V = np.std(train_ret.values, axis=0)
    w_new, c_new = solve_new_model(mu_train, Sigma, R, mu0, w_prev, rho, V = train_vol
    )
    w_prev = w_new.copy()  # update for next turnover constraint

    r_new = test_ret @ w_new
    ret_bt['new_model'].append(np.prod(1 + r_new) - 1)
```

```
    dates.append(periods[i].to_timestamp('M'))

# Plot cumulative returns
df_ret = pd.DataFrame(ret_bt, index=dates)
cum_ret = (1 + df_ret).cumprod()

plt.figure(figsize=(10, 5))
for model in cum_ret.columns:
    plt.plot(cum_ret.index, cum_ret[model], label=model)
plt.title("Cumulative Return Comparison")
plt.xlabel("Date")
plt.ylabel("Portfolio Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 3: Graph shown in section 5

H: Risk Measures and Return Metrics

```
import numpy as np
import pandas as pd
import cvxpy as cp
from scipy.stats import norm

# === Step 1: Portfolio Optimization Function ===
def optimize_portfolios(mean_returns: pd.Series,
                        covariance_matrix: np.ndarray,
                        tickers: list,
                        R: float = 0.0007,
                        mu0: float = 0.0,
                        lasso_alpha: float = 0.1,
                        ridge_alpha: float = 0.1) -> pd.DataFrame:
    n = len(tickers)
    mu = mean_returns.to_numpy()
    Sigma = covariance_matrix

    models = {
        'classical':   (0.0, 0.0),
        'lasso':       (lasso_alpha, 0.0),
        'ridge':       (0.0, ridge_alpha),
        'elastic_net': (lasso_alpha, ridge_alpha),
    }

    records = []
    for name, (a1, a2) in models.items():
        w = cp.Variable(n)
        x0 = cp.Variable(1)

        risk_term = cp.quad_form(w, Sigma)
        l1_penalty = a1 * cp.norm1(w)
        l2_penalty = 0.5 * a2 * cp.sum_squares(w)
        obj = cp.Minimize(risk_term + l1_penalty + l2_penalty)

        constraints = [
            cp.sum(w) + x0[0] == 1,
            w >= 0,
            x0 >= 0,
            mu @ w + mu0 * x0[0] >= R
        ]

        prob = cp.Problem(obj, constraints)
        prob.solve(solver=cp.SCS, verbose=False)

        weights = w.value
        cash = float(x0.value)
        exp_ret = mu @ weights + mu0 * cash
        exp_risk = np.sqrt(weights.T @ Sigma @ weights)
```

```
        rec = {tickers[i]: weights[i] for i in range(n)}
        rec.update({
            'cash': cash,
            'exp_return': exp_ret,
            'exp_risk': exp_risk,
            'model': name
        })
        records.append(rec)

    df = pd.DataFrame(records).set_index('model')
    return df

# === Step 2: Risk Metric Calculation ===
def calculate_risk_metrics(portfolio_returns, risk_free_rate=0.0):
    mean_return = portfolio_returns.mean()
    std_dev = portfolio_returns.std()
    var_95 = np.percentile(portfolio_returns, 5)
    cvar_95 = portfolio_returns[portfolio_returns <= var_95].mean()

    cumulative_returns = (1 + portfolio_returns).cumprod()
    peak = cumulative_returns.cummax()
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = drawdown.min()

    sharpe_ratio = mean_return / std_dev if std_dev != 0 else np.nan

    return {
        'Mean Return': mean_return,
        'Std Dev': std_dev,
        'VaR (95%)': var_95,
        'CVaR (95%)': cvar_95,
        'Max Drawdown': max_drawdown,
        'Sharpe Ratio': sharpe_ratio
    }

# === Step 3: Cost Calculation ===
def calculate_hold_cost(weights: np.ndarray,
                        holding_days: int,
                        annual_rate: float = 0.02) -> float:
    daily_rate = annual_rate / 252
    return weights.sum() * daily_rate * holding_days

def calculate_trade_cost(original_weights: np.ndarray,
                         new_weights: np.ndarray,
                         cost_rate: float = 0.001) -> float:
    weight_diff = np.abs(new_weights - original_weights)
    return weight_diff.sum() * cost_rate

# === Step 4: Optimization + Return Series + Risk + Performance Table ===
# df_portfolios = optimize_portfolios(mean_returns, covariance_matrix, tickers)
# daily_returns = ... # Your (T, N) DataFrame of asset returns

model_returns = {}
for model_name in df_portfolios.index:
    weights = df_portfolios.loc[model_name, tickers].to_numpy()
    port_return = daily_returns @ weights
    model_returns[model_name.capitalize()] = port_return

# === Compute Risk Metrics Table ===
metrics = {}
for name, returns in model_returns.items():
    metrics[name] = calculate_risk_metrics(returns)

metrics_df = pd.DataFrame(metrics).T
metrics_df.index.name = 'Model'
metrics_df['Cash Allocation (%)'] = df_portfolios['cash'] * 100
print("Table 3: Risk Metrics Comparison")
print(metrics_df.round(6))

# === Compute Performance Table (Table 2) ===
```

```
holding_days = len(daily_returns)
n_assets = len(tickers)
equal_weights = np.array([1/n_assets] * n_assets)
benchmark = 0.0175  # Example: 0.0007 daily target * 25 days

hold_costs = {}
trade_costs = {}

for model_name in df_portfolios.index:
    weights = df_portfolios.loc[model_name, tickers].to_numpy()
    hold_costs[model_name] = calculate_hold_cost(weights, holding_days)
    trade_costs[model_name] = calculate_trade_cost(equal_weights, weights)

performance_df = pd.DataFrame(index=df_portfolios.index)
performance_df['Gross Return'] = df_portfolios['exp_return']
performance_df['Hold Cost'] = performance_df.index.map(hold_costs)
performance_df['Trade Cost'] = performance_df.index.map(trade_costs)
performance_df['Net Return'] = performance_df['Gross Return'] - performance_df['Hold
    Cost'] - performance_df['Trade Cost']
performance_df['Net Active Return'] = performance_df['Net Return'] - benchmark
performance_df['Cash Allocation (%)'] = df_portfolios['cash'] * 100

performance_df = performance_df[[
    'Gross Return', 'Net Return', 'Net Active Return',
    'Hold Cost', 'Trade Cost', 'Cash Allocation (%)'
]]

print("\nTable 2: Portfolio Performance Comparison (Target Return = 0.05%)")
print(performance_df.round(4))
```