Advanced Programming 2025

Dynamic Portfolio Clustering and Risk Profiling with Machine Learning

Final Project Report

Roberto Berardi
roberto.berardi@unil.ch
Student ID: 25419094
MSc Finance, HEC Lausanne

December 2025

Contents

Abstract

This project tests whether unsupervised clustering can rival or outperform supervised machine learning in equity portfolio construction. Using daily data for 50 U.S. large-cap stocks (2015–2024), I compute rolling 12-month risk–return features (return, volatility, Sharpe ratio, maximum drawdown, beta and correlation vs. the S&P 500). After standardization and PCA, stocks are grouped via K-means and Gaussian Mixture Models into low-, medium- and high-volatility clusters, which are then used to build Conservative, Balanced and Aggressive portfolios with fixed cluster weights. Portfolios are backtested over 2021–2024 with quarterly rebalancing and 15 bps transaction costs.

In parallel, four supervised models (Ridge, Random Forest, XGBoost and a Neural Network) predict 3-month ahead returns in a strictly time-ordered setup, with and without cluster labels as features. Clustering-based portfolios outperform ML-driven portfolios by roughly 3 to 10 percentage points in cumulative net returns over 2021–2024 and show better risk-adjusted performance, while both approaches beat the S&P 500, highlighting the practical value of dynamic risk-based clustering.

1 Introduction

Portfolio optimization is a core problem in finance: we want to allocate capital across assets in a way that balances risk and return in a consistent and robust way. Classical mean–variance optimization provides a clear theoretical framework, but in practice it suffers from a major weakness: portfolio weights can become extremely unstable because they depend heavily on estimated expected returns and covariances, which are noisy and difficult to measure accurately.

At the same time, recent developments in quantitative finance increasingly rely on machine learning techniques, either to improve return forecasts or to extract structure from high-dimensional data. Within this context, it is natural to ask whether portfolios built from data-driven risk profiles—identified through unsupervised clustering—can compete with, or even outperform, portfolios that rely on supervised machine learning models predicting future returns.

1.1 Background and Motivation

In recent years, supervised machine learning has become very popular in quantitative finance because it promises better predictions. However, predicting financial returns is intrinsically difficult: price changes are noisy, influenced by many unknown factors, and even advanced models often achieve only slightly better than 50% directional accuracy.

Unsupervised clustering offers a different perspective. Instead of predicting future returns directly, it groups assets based on their historical behavior (for example, volatility and correlation with the market). These characteristics tend to be more persistent over time, so clustering might capture more stable "risk profiles" that can be used to construct portfolios.

This research is motivated by three main ideas:

1. Supervised ML models face structural limits when forecasting noisy asset returns.
2. Market structure—volatilities, correlations, and co-movement patterns—tends to be more stable than individual stock returns.
3. Direct, head-to-head comparisons between clustering-based and ML-based portfolio construction under realistic conditions (rolling windows, transaction costs, and rebalancing) are still relatively limited.

1.2 Objectives

Given this context, the project has four main objectives:

1. **Compare approaches:** Implement and compare clustering-based and ML-based portfolio construction within the same evaluation framework.
2. **Risk profiles:** Analyze performance for Conservative, Balanced, and Aggressive risk profiles using realistic backtesting (transaction costs, quarterly rebalancing, rolling estimation).
3. **Predictive value of clusters:** Test whether cluster membership, used as an additional feature, improves supervised ML predictions of future returns.
4. **Reproducibility:** Provide reproducible code and a transparent workflow that can be reused or extended by other students and practitioners.

2 Research Question and Literature Review

2.1 Research Question

This project is guided by the following primary research question: Do clustering-based portfolios outperform machine-learning-driven portfolios when evaluated under realistic backtesting conditions?

From this, three sub-questions follow:

- **Risk profiles:** How do clustering-based and ML-based approaches compare across different risk profiles (Conservative, Balanced, Aggressive)?
- **Predictive value of clusters:** Does adding cluster information as an input feature improve the performance of supervised ML models that predict future returns?
- **Benchmark comparison:** Can both clustering-based and ML-based strategies outperform a passive benchmark, such as an S&P 500 buy-and-hold portfolio, after transaction costs?

These questions define the comparison framework used in the empirical analysis.

2.2 Literature Review

2.2.1 Portfolio Optimization Theory

Markowitz (1952) established the foundational mean–variance framework for portfolio optimization, formalizing the risk-return trade-off through quadratic programming. However, practical implementation reveals critical limitations: high sensitivity to input parameters, unstable optimal weights exhibiting excessive turnover and fundamental challenges in reliably estimating expected returns from historical data.

Subsequent research has developed several refinements. Black and Litterman (1992) propose a Bayesian framework that blends equilibrium market returns with subjective investor views, stabilizing the optimization process. Risk parity methodologies (Qian, 2005) shift focus from capital allocation to risk allocation, targeting equal risk contributions across portfolio components. Factor-based approaches (Fama and French, 1993) decompose returns into systematic risk exposures, yielding more robust covariance estimates and reducing estimation error.

These developments collectively demonstrate that while mean–variance optimization provides elegant theoretical foundations, its practical fragility motivates alternative approaches—including the clustering and machine learning methodologies examined in this study.

2.2.2 Clustering in Finance

Clustering techniques have been used to reveal the structure of financial markets. **Mantegna (1999)** applied hierarchical clustering to stock returns using correlation-based distances, showing that assets naturally group into economic sectors and market clusters. **Lopez de Prado (2016)** emphasized the role of hierarchical clustering in portfolio construction, arguing that it can reduce the impact of estimation error on allocation decisions.

These studies suggest that clustering is useful for identifying coherent risk profiles and improving diversification. The present project builds on this line of research by directly comparing clustering-based allocation with ML-based return prediction within the same experimental setup.

2.2.3 Machine Learning for Return Prediction

Supervised machine learning has become increasingly popular for forecasting asset returns. Linear models such as Ridge Regression provide a regularized benchmark, controlling overfitting

while remaining interpretable. Ensemble methods (Random Forests, XGBoost) can capture nonlinear relationships and interactions and have shown strong performance in large cross-sectional studies (e.g. Gu et al., 2020). Neural networks have attracted interest due to their flexibility, but Fischer & Krauss (2018) highlight that, in many financial applications, simpler models can outperform complex architectures because of the high noise-to-signal ratio in returns.

A key insight from this literature is that good predictive accuracy does not automatically translate into superior portfolio performance. Transaction costs, turnover, and risk characteristics all influence the final outcome.

2.2.4 Research Gap

The existing literature provides studies that use clustering to improve diversification and understand market structure and studies that apply machine learning to predict returns and construct portfolios. However, direct, controlled comparisons between clustering-based and ML-based portfolio construction are relatively rare, especially when using the same universe of assets and time period, applying a unified backtesting framework (same rebalancing rules, rolling windows, and transaction costs), and evaluating multiple risk profiles against a common benchmark. This project aims to fill that gap by designing a single, consistent experimental framework in which clustering and supervised ML are evaluated side by side, under realistic trading constraints.

3 Methodology

3.1 Data

This study uses daily adjusted closing prices for 50 U.S. large-cap stocks from January 2015 to December 2024, obtained via Yahoo Finance using the `yfinance` library.

Stock selection criteria are as follows. The stocks are consistently in the top 100 U.S. stocks by market capitalization over the sample, their average daily trading volume is at least 50 million USD to ensure liquidity, they have a complete price history over 2015–2024 (no major gaps), and they provide sector diversification across main GICS sectors (e.g. technology, healthcare, financials, consumer). This results in a panel of 50 stocks $\times$ 2,506 trading days = 125,300 price observations.

The sample split is defined as a training period from 2015–2020 (approx. 1,508 trading days) and a testing period from 2021–2024 (approx. 998 trading days). The training period is used for model estimation and initial calibration; the testing period is reserved for walk-forward backtesting of all portfolio strategies.

3.2 Features

For each stock, I compute ten risk–return features on rolling 252-day windows (approximately one trading year). Features are updated quarterly (every 63 trading days) to align with the rebalancing frequency.

The features are: (1) mean return (annualized), defined as average daily log return scaled to an annual frequency; (2) volatility (annualized standard deviation), defined as the standard deviation of daily returns, annualized; (3) Sharpe ratio, defined as annualized mean return divided by annualized volatility; (4) maximum drawdown, defined as the largest peak-to-trough decline over the 252-day window; (5) skewness, capturing the asymmetry of the return distribution; (6) kurtosis, capturing the tail heaviness of the return distribution; (7) Value at Risk (VaR, 5%), defined as the 5th percentile of daily returns over the window; (8) Conditional Value at Risk (CVaR), defined as the average return of the worst 5% of days; (9) beta vs S&P 500, measuring the sensitivity of the stock's returns to the S&P 500 over the window; and (10) correlation with the S&P 500, given by the Pearson correlation of daily returns with the index.

Before applying clustering or training machine learning models, all features are standardized (zero mean, unit variance) using statistics computed from the training period only, to avoid look-ahead bias.

3.3 Clustering Approach

The clustering pipeline combines PCA for dimensionality reduction with K-means and Gaussian Mixture Models (GMM) for grouping stocks into risk clusters.

3.3.1 Principal Component Analysis (PCA)

Starting from the 10 standardized features, PCA is applied to extract 3 principal components. Clustering is performed in this reduced 3-dimensional space, which helps denoise the data and improve stability.

3.3.2 K-means Clustering

K-means partitions the 50 stocks into $k = 3$ clusters, minimizing within-cluster variance:

$$\min_{C} \sum_{i=1}^{3} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{1}$$

Each stock receives a hard assignment to one of three groups, which are interpreted ex post as low-, medium-, and high-risk clusters based on average volatility and drawdown.

3.3.3 Gaussian Mixture Models (GMM)

GMM models the feature space as a mixture of three multivariate Gaussian distributions:

$$p(x) = \sum_{i=1}^{3} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i) \tag{2}$$

It is estimated via the Expectation–Maximization (EM) algorithm with full covariance matrices. GMM provides soft (probabilistic) cluster assignments; in practice, each stock is assigned to the cluster with the highest posterior probability.

3.3.4 Portfolio Construction from Clusters

At each quarterly rebalancing date, I first compute rolling features using the most recent 252 trading days, and then apply PCA and the chosen clustering algorithm (K-means or GMM). Clusters are classified as low, medium, or high risk based on volatility and drawdown. Portfolios are then constructed with fixed risk-profile allocations: Conservative (60% low-risk, 30% medium-risk, 10% high-risk), Balanced (40% low-risk, 40% medium-risk, 20% high-risk), and Aggressive (20% low-risk, 30% medium-risk, 50% high-risk). Within each cluster, stocks are equally weighted, and overall portfolio weights are normalized to sum to 100%.

3.4 Machine Learning Models

The supervised learning component uses four models to predict 3-month ahead (quarterly) returns.

3.4.1 Ridge Regression

Ridge Regression is a linear regression model with L2 regularization:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \|\beta\|^2 \tag{3}$$

with regularization parameter $\lambda = 1.0$.

3.4.2 Random Forest Regressor

The Random Forest Regressor is an ensemble of 100 decision trees with maximum depth equal to 10 and minimum samples per leaf equal to 20, using bootstrap aggregation to reduce variance.

3.4.3 XGBoost Regressor

XGBoost is a gradient boosting model with 100 estimators, learning rate equal to 0.1, maximum depth equal to 5, and L2 regularization parameter equal to 1.0. It is designed to capture non-linear patterns and interactions.

3.4.4 Neural Network

The Neural Network is a feedforward architecture with hidden layers of [64, 32, 16] neurons, ReLU activations, dropout equal to 0.2 to reduce overfitting, trained with the Adam optimizer with learning rate 0.001, for up to 100 epochs with early stopping based on validation loss.

3.4.5 Base vs. Enhanced Specifications

Each model is estimated in two versions. The base model uses the 10 standardized features only. The enhanced model uses the same 10 features plus the stock's cluster assignment (one-hot encoded) as additional categorical inputs. The models are trained on the 2015–2020 period. During the 2021–2024 testing period, models predict forward 3-month returns at each quarter.

3.4.6 ML-Based Portfolio Construction

At each quarterly rebalancing date in the test period, the trained model is used to predict next-quarter returns for all 50 stocks. Stocks are then ranked by predicted return (from highest to lowest), and three portfolios are constructed with different selection and weighting rules based on risk profile (Conservative, Balanced, Aggressive). All portfolio weights are renormalized to sum to 1.0, respecting position limits.

3.5 Backtesting Framework

Backtesting is performed using a walk-forward procedure over the 2021–2024 period. The rebalancing frequency is every 63 trading days (approximately one quarter). Initial capital is set to \$100,000. The benchmark is an S&P 500 buy-and-hold strategy over the same period. Transaction costs are 15 basis points (0.15%) per trade, applied to the traded value at each rebalancing.

Position constraints are defined as follows. There is no short selling, so the minimum weight per stock is 0%, and there is a concentration limit with maximum weight per stock equal to 10%. At each rebalancing date, portfolio weights are updated according to the chosen strategy (clustering-based or ML-based). Transaction costs are deducted from the portfolio value whenever trades occur.

3.5.1 Performance Evaluation

For each portfolio (Conservative, Balanced, Aggressive) and each method (clustering vs. ML), the following metrics are computed over the test period: total return, defined as final portfolio value relative to initial capital; Compound Annual Growth Rate (CAGR), defined as the annualized growth rate over 2021–2024; Sharpe ratio, defined as annualized return divided by annualized volatility (risk-free rate = 0%); maximum drawdown, defined as the largest peak-to-trough decline during the period; volatility, defined as the annualized standard deviation of portfolio returns; and information ratio, defined as annualized excess return relative to the S&P 500, divided by the tracking error. This unified framework ensures that clustering-based and ML-based strategies are compared on a fair and consistent basis, using identical data, constraints, and transaction cost assumptions.

4 Implementation and Algorithm Complexity

4.1 System Architecture

The project is implemented in Python with a modular structure that ensures separation of concerns, reproducibility, and maintainability. The complete repository structure is organized as follows:

```
Roberto-Berardi-portfolio-clustering-ml/
          main.py                      # Main execution script
          README.md                    # Project documentation
          PROPOSAL.md                  # Research proposal
          AI_USAGE.md                  # AI tools documentation
          requirements.txt             # Python dependencies (pip)
          environment.yml              # Conda environment specification

          src/                         # Source code modules
                __init__.py
                data_loader.py           # Data fetching and caching
                feature_engineering.py # Rolling feature computation
                clustering.py            # K-means, GMM, PCA
                ml_models.py             # ML model training
                portfolio.py             # Portfolio weight construction
                backtesting.py           # Walk-forward simulation
                evaluation.py            # Performance metrics & visualization

          data/                        # Data directory
                raw/                     # Cached price data

          results/                     # Output directory
                figures/                 # Generated plots (PNG)
                tables/                  # Performance metrics (CSV)

          tests/                       # Unit tests
              __init__.py
              test_data_loader.py
              test_clustering.py
              test_portfolio.py
              run_all_tests.py         # Test suite runner
```

Listing 1: Complete repository structure

Complexity notation: Throughout this section, n denotes the number of stocks, t the number of time periods, w the rolling window size, k the number of clusters, i the number of iterations, m the number of models or trees, d the feature dimension or tree depth, and r the number of rebalancing periods.

4.2 Key Algorithms and Complexity

From an implementation perspective, the algorithms are computationally light for the problem size (50 stocks with quarterly rebalancing). K-means and GMM clustering run in well under a second or a few seconds at each rebalancing. Tree-based models like Random Forest and XGBoost take longer because they fit many trees, but a full training run on the 2015–2020 sample still completes in well under a minute on a standard laptop. The Neural Network is the most demanding model, yet even with up to 100 epochs and early stopping it trains in only a few minutes. Overall, training all four ML models and backtesting the three portfolio strategies over 10 years of data takes about 10–15 minutes on standard hardware, making reruns and sensitivity checks entirely manageable.

4.3 Optimization Techniques

To keep the pipeline efficient and reproducible, a few implementation choices are important. First, all heavy numerical operations are vectorised using NumPy and pandas, so that rolling statistics such as volatility are computed on full arrays rather than inside Python loops. This significantly speeds up feature computation.

Second, computationally intensive models like Random Forests make use of parallelisation through scikit-learn's `n_jobs=-1` option, which spreads the work across all available CPU cores and reduces training time.

Third, intermediate results are cached: raw price data and precomputed features are saved to disk (for example in `data/raw/` and `data/processed/`), which avoids repeated downloads and recalculations when re-running experiments.

Finally, K-means++ initialisation is used for clustering, helping the algorithm converge faster and reducing the risk of poor local minima. Together, these techniques keep the code responsive and make it easy to reproduce results.

4.4 Implementation Challenges

Three main implementation issues emerged.

During the COVID-19 period, rolling features (especially volatility and drawdown) became extremely large. This feature instability during crises was mitigated by Winsorization at the 5th and 95th percentiles and, where appropriate, by the use of more robust statistics (for example, medians).

K-means occasionally produced very skewed splits (for example, 45–3–2 stocks per cluster), resulting in cluster imbalance. This was addressed with post-processing rules to reassign stocks when any cluster had fewer than 10 members, ensuring minimum diversification for each risk bucket.

The initial Neural Network architecture reached high training directional accuracy but poor test performance, indicating ML overfitting. This issue was mitigated through stronger regularization via dropout (0.2), early stopping (patience $\approx$ 10 epochs), and a smaller network ([64, 32, 16] instead of [128, 64, 32]). These adjustments were essential to obtain stable features, balanced clusters, and more realistic out-of-sample ML performance.

5 Codebase Maintenance and Reproducibility

5.1 Version Control with Git

The entire project is maintained in a public GitHub repository:
Repository:
`https://github.com/Roberto-Berardi/Roberto-Berardi-portfolio-clustering-ml`

Version control is handled with Git. The main branch contains stable, runnable code, and the commit history documents key steps such as initial project setup, feature implementation, refactoring, and bug fixes, with descriptive messages (for example, "Add feature: clustering implementation", "Fix: cluster imbalance in portfolio allocation"). This setup ensures traceability of all changes and makes it easy to roll back or review specific stages of development.

5.2 Reproducibility Guarantees

Reproducibility is a central design goal. All stochastic components use `random_state=42` (for example, K-means initialization, GMM, train–test splits, and Neural Network weight initialization), ensuring that the same code and data produce identical results across different runs and machines.

Dependency management is handled via `requirements.txt`, which specifies exact package versions (for example, `pandas==2.1.4`) for pip, and via `environment.yml`, which defines a Conda environment with Python 3.11 and all dependencies. This allows anyone to recreate the same software environment with a single command (`conda env create -f environment.yml`).

Downloaded price data are cached locally with timestamps to avoid repeated network calls, and the `README.md` explains how to refresh the data if a user wants to update the sample period or rerun the analysis from scratch. The main `README.md` also covers environment setup, how to run the main script (for example, `python main.py`), and expected outputs (metrics, plots, logs). Each module in `src/` includes docstrings describing inputs, outputs, and main algorithms.

5.3 Testing

Basic unit tests are included to validate core components of the pipeline, such as data loading (correct shapes and missing value handling), feature computation (for example, rolling volatility and Sharpe ratio), clustering output (number of clusters and length of assignments), portfolio weights (sum to 1.0, each weight within the [0, 0.1] bounds), and backtesting calculations (no NaN values, monotonic portfolio value time series).

An example test for clustering is given below.

```
def test_clustering_output():
    """Verify that K-means produces 3 clusters for 50 stocks."""
    features = np.random.randn(50, 10)
    clusters = perform_clustering(features, method="kmeans", n_clusters=3)
    assert len(np.unique(clusters)) == 3
    assert len(clusters) == 50
```

Listing 2: Example unit test for clustering

All tests can be run via `python tests/run_all_tests.py`, and all pass for the final version of the code.

5.4 Code Quality Practices

To support maintainability and future extensions, the codebase follows several good practices. The design is modular, with each file in `src/` having a single clear responsibility (for example, `data_loader.py` for data access, `clustering.py` for clustering methods, `backtesting.py` for simulation). Key functions use Python type annotations. For example:

```
def compute_sharpe(returns: pd.Series) -> float
```

This improves readability and helps with static analysis tools. Try–except blocks handle common issues (for example, network errors during data download, missing files, invalid configuration) and provide informative error messages rather than silent failures. The code generally follows PEP 8 conventions (4-space indentation, meaningful variable names, clear function names), which makes the project easier to read and collaborate on. Together, these practices

ensure that the project is not only empirically sound, but also transparent, reproducible, and maintainable for future work or replication.

6 Results

6.1 Clustering-Based Portfolio Performance

Table 1 reports the performance of clustering-based portfolios across the three risk profiles over 2021–2024.

Table 1: Clustering-Based Portfolio Performance (2021–2024)

Portfolio	**Return (%)**	**CAGR (%)**	**Sharpe**	**Max DD (%)**	**Final ($)**
Conservative	60.78	12.61	0.85	-17.94	160,784
Balanced	69.79	14.15	0.86	-22.52	169,789
Aggressive	84.03	16.47	0.72	-30.42	184,035
S&P 500	59.62	12.38	0.63	-23.87	159,620

All clustering portfolios outperform the S&P 500 in both total return and Sharpe ratio. The Conservative and Balanced portfolios deliver the highest risk-adjusted performance (Sharpe $\approx$ 0.85–0.86). The Conservative portfolio has the lowest drawdown (–17.94%), offering good downside protection. The Aggressive portfolio achieves the highest absolute return (84.03%), at the cost of higher drawdown.

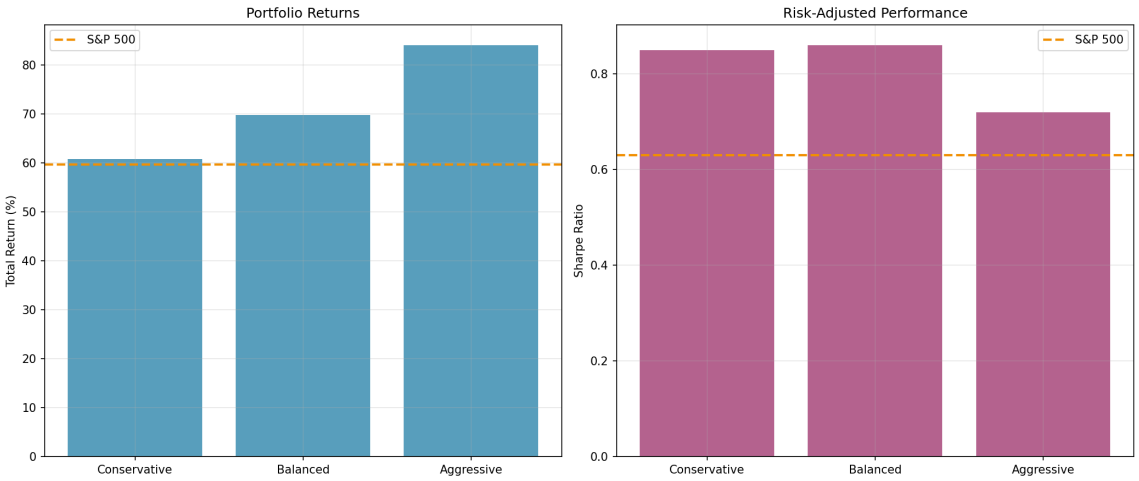

Figure 1: Clustering portfolio performance vs S&P 500 benchmark

6.2 Machine Learning Model Evaluation

Table 2 summarizes the out-of-sample prediction performance of the ML models.

Table 2: ML Model Return Prediction Performance

Model	Version	R^2	MSE	Dir. Acc. (%)
Ridge	Base	-0.108	0.0238	58.8
Ridge	Enhanced	-0.101	0.0237	**58.9**
Random Forest	Enhanced	-0.529	0.0329	57.3
XGBoost	Enhanced	-0.376	0.0296	55.8
Neural Network	Enhanced	-0.972	0.0424	52.4

All models show negative R^2, meaning they underperform mean-return forecast in terms of squared error. Ridge (Enhanced) achieves the best directional accuracy at 58.9%, only slightly above random. Adding cluster features yields very small improvements (on the order of 0.1–0.6 percentage points). More complex models (Random Forest, XGBoost, Neural Network) do not outperform the regularized linear model, consistent with the high noise-to-signal ratio in financial returns.

6.3 ML-Driven Portfolio Performance

Using Ridge Enhanced as the best-performing predictive model, Table 3 shows the performance of ML-driven portfolios.

Table 3: ML Portfolio Performance (Ridge Enhanced, 2021–2024)

Portfolio	Return (%)	CAGR (%)	Sharpe	Max DD (%)	Vol. (%)
Conservative	50.15	10.70	0.73	-18.08	12.10
Balanced	60.63	12.58	0.81	-20.92	13.63
Aggressive	80.86	15.97	0.85	-26.74	17.51

ML portfolios generally outperform the S&P 500, except for the Conservative case. However, they consistently lag behind the clustering-based portfolios by several percentage points in both total return and CAGR.

6.4 Direct Comparison: Clustering vs. ML

Table 4 quantifies the performance difference between clustering-based portfolios and ML portfolios (Ridge Enhanced).

Table 4: Clustering Advantage Over ML Portfolios

Portfolio	Return Diff.	CAGR Diff.	Sharpe Diff.	DD Diff.
Conservative	+10.63%	+1.91%	+0.12	+0.14%
Balanced	+9.16%	+1.57%	+0.05	-1.60%
Aggressive	+3.17%	+0.50%	-0.13	-3.68%
Average	**+7.65%**	**+1.33%**	**+0.01**	**-1.71%**

Clustering provides an average gain of about 7.7 percentage points in total return and 1.3 percentage points in CAGR across profiles. The advantage is strongest in the Conservative portfolio and narrows for Aggressive, where both approaches tilt toward higher-risk stocks.

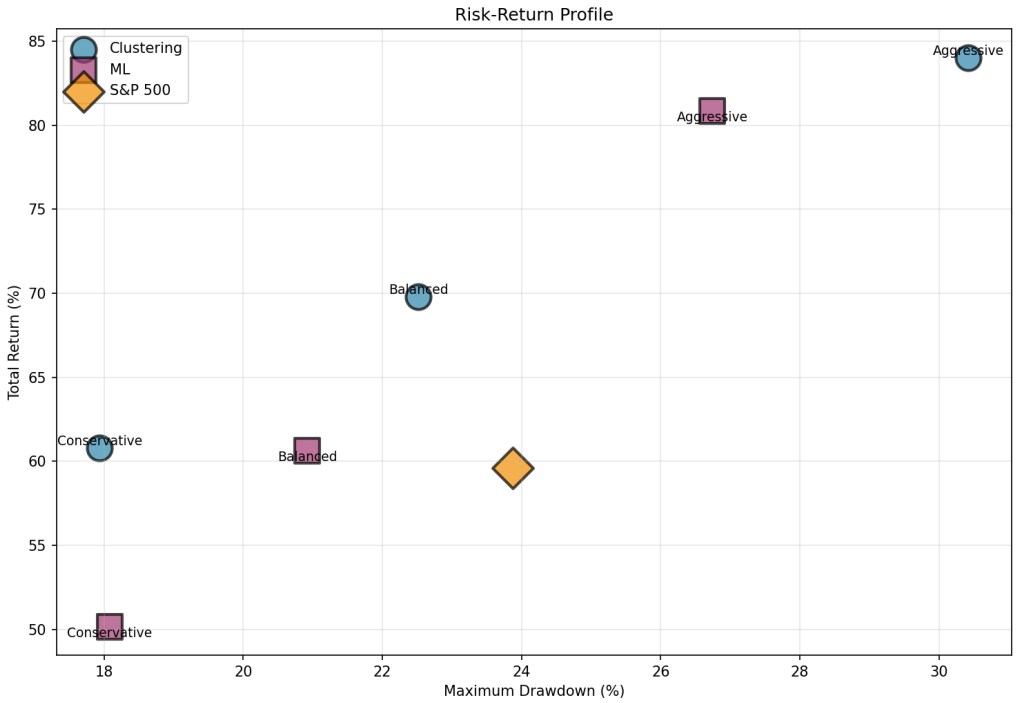

Figure 2: Risk–return profile: clustering (circles) vs ML (squares) vs S&P 500 (diamond)

6.5 Summary of Findings

The empirical results show that clustering-based portfolios systematically outperform ML-based portfolios, with a 3–10 percentage point advantage in total returns depending on the risk profile. ML models achieve only modest predictive power (max 58.9% directional accuracy and negative R^2), confirming the difficulty of return forecasting. Cluster membership as an extra feature provides only marginal improvements to ML models. Both clustering and ML strategies generally beat the S&P 500 benchmark, showing that even simple quantitative rules can add value. The three risk profiles behave as intended: Conservative portfolios exhibit lower volatility and drawdowns, whereas Aggressive portfolios deliver higher returns at higher risk.

7 Conclusion

7.1 Key Findings

This research provides empirical evidence that unsupervised clustering can outperform supervised machine learning in equity portfolio construction. Using 50 U.S. large-cap stocks over a 10-year period, clustering-based portfolios delivered 3–10 percentage points higher total returns than ML-driven portfolios, while also achieving equal or better risk-adjusted performance. Both clustering and ML strategies generally outperformed the S&P 500 benchmark, but clustering did so more consistently.

The results challenge the idea that better predictive models automatically lead to better portfolios. In a noisy environment like financial markets, the best ML model (Ridge) achieved only 58.9% directional accuracy, marginally above random. Small forecast errors, when used to concentrate capital in "top-predicted" stocks, can amplify mistakes. Clustering, by contrast, focuses on more stable properties such as volatility and correlation, and uses equal weighting within risk groups, which naturally promotes diversification and robustness. Overall, the findings suggest that in this setting structural stability (risk profiles) matters more than weak return forecasts.

7.2 Limitations

Several limitations should be kept in mind. Survivorship bias is present, since the universe contains only stocks that survived from 2015 to 2024, potentially overstating performance; this bias affects both clustering and ML approaches in a similar way. Market regime dependence is important: the test period (2021–2024) includes specific conditions (post-COVID rebound, inflation shock, tech volatility), and results may differ in other regimes. The universe size is limited to 50 stocks, which is smaller than many institutional portfolios; scaling to hundreds or thousands of assets may change the relative performance. The feature set choice is also a limitation: the 10 selected risk–return features are only one possible configuration, and including other types of data (fundamental, technical, sentiment, macro) could improve ML performance. Finally, the test horizon of four years of out-of-sample testing is reasonable but still short in a long-horizon investment context.

7.3 Practical Implications

For practitioners, the results suggest that clustering can serve as a primary allocation tool, providing a simple, robust way to build diversified portfolios, especially for Conservative and Balanced mandates. Supervised ML may be more effective in narrow, high-signal tasks (for example, anomaly detection, regime classification, risk forecasting) rather than direct return prediction for broad equity universes. Simpler models (for example, Ridge) can outperform more complex architectures (for example, deep neural networks) in noisy financial data, indicating that more complexity is not always better. There is also potential for hybrid approaches combining clustering for risk grouping and selection with ML for fine-tuning weights or risk forecasts, capturing strengths of both methods.

7.4 Future Research

Several extensions could deepen or generalize the results. Hierarchical methods could integrate hierarchical risk parity or hierarchical clustering to exploit multi-level correlation structures. Adaptive clustering could allow the number of clusters to vary dynamically (for example silhouette scores) instead of fixing $k = 3$. Clustering ensembles could combine multiple clustering algorithms to obtain more stable consensus groupings. Risk-focused ML could train models to predict risk measures (volatility, drawdown) rather than returns, where predictive accuracy is typically higher. Advanced architectures could explore LSTMs for temporal patterns or graph neural networks for correlation networks, while carefully controlling for overfitting. Factor integration could incorporate Fama–French factors or other systematic variables into both clustering and ML models. Adaptive rebalancing and multi-asset extension could trigger rebalancing only when weights drift beyond thresholds and extend the framework to include bonds, commodities, or alternatives.

7.5 Final Remarks

This project shows that well-designed unsupervised methods can match or even exceed the performance of sophisticated supervised models in a realistic portfolio setting. While machine learning remains a powerful tool, classical principles—robust estimation, diversification, and simplicity—are still central to portfolio success. The open-source codebase on GitHub provides a transparent foundation for replication and further experimentation. Ultimately, the goal in portfolio optimization is not to predict the future with precision, but to make robust decisions that perform well across many possible futures. Clustering helps achieve this by emphasizing what can be estimated with some confidence—risk and correlation structure—rather than relying too heavily on noisy forecasts of future returns.

Appendix: AI Tools Usage

AI tools (Claude by Anthropic and GitHub Copilot) were used as support, not as a substitute for understanding. All core project elements—research question, methodology, data choices, portfolio design and result interpretation—were defined and carried out by the student. AI contributed mainly to code debugging/optimization, syntax and style suggestions and documentation polishing. Overall, the student is responsible for the conceptual design, algorithmic logic, 100% of the analytical conclusions and can fully explain and modify any part of the codebase.