[Short-Horizon Return Prediction for SMCI Using Machine Learning: A Iak-Form Market Efficiency Study]

[Braedon Potts][*]

Department of Computational Mathematics, Science and Engineering

Michigan State University, East Lansing, MI 48824

(Dated: December 8, 2025)

Abstract

This project investigates whether short-horizon price movements of Super Micro Computer Inc. (SMCI) can be predicted using standard machine learning methods. Using daily OHLCV data from Yahoo Finance, I construct purely backward-looking technical features, including 1-day log returns, rolling mean returns over multiple windows, rolling volatility, and log-volume changes. The primary task is to forecast next-day log returns via regression, using Ridge regression and Gradient Boosting Regressor, evaluated against naive baselines that predict zero or the in-sample mean return. I use a strictly time-ordered 60/20/20 train/validation/test split to avoid look-ahead bias.

Across all models, out-of-sample performance is very close to the naive baselines: test 2 R 2 values are slightly negative, and trading strategies based on model forecasts produce Sharpe ratios near or below zero and underperform a buy-and-hold benchmark. As an extension, I reframe the problem as predicting only the sign of next-day returns and train Logistic Regression and Random Forest classifiers. These models reach test accuracies around 48–49

Overall, my results suggest that, for SMCI over this period, simple price- and volume-based technical features do not provide a robust edge for one-day-ahead prediction, consistent with Iak-form market efficiency.

BACKGROUND AND MOTIVATION

The core question of this project is: can I use historical price and volume data to predict next-day returns for Super Micro Computer Inc. (SMCI), and if so, is the signal strong enough to be economically meaningful?

This problem matters because even a small, reliable edge in short-horizon return prediction has direct implications for trading profitability, risk management, and my understanding of how "efficient" financial markets really are. Quantitative traders and portfolio managers care because such signals can drive systematic trading strategies. Academics and students care because return predictability (or the lack of it) is a concrete way to test Iak-form market efficiency.

If this problem Ire "solved" in a strong sense—i.e., if I found a robust, stable model that forecasts returns—then one could, in principle, construct strategies that outperform a buy-

and-hold benchmark after costs. Conversely, failing to find such a model provides evidence that, at this horizon and with these features, the market is hard to beat.

Prior work includes technical analysis rules, linear factor models, and more recent use of machine learning (tree ensembles, neural networks) for stock return prediction, typically with mixed and often fragile results.

my desired outcome is not just to "make money," but to rigorously test whether simple, backward-looking technical features contain exploitable predictive information for SMCI's one-day-ahead returns. Machine learning helps by flexibly mapping nonlinear relationships betIen features and returns (Ridge, Gradient Boosting, Logistic Regression, Random Forests) and by providing a systematic framework to evaluate out-of-sample predictive poIr, rather than relying on ad-hoc trading rules.

DATA DESCRIPTION

Describe ymy data and any issues there might be. This section should have clear ansIrs to all these questions:

Data Origins

The dataset used in this project consists of daily price and volume data for Super Micro Computer Inc. (SMCI), a U.S.-listed technology company that trades on the NASDAQ. The data are obtained programmatically via the yfinance Python library, which provides an interface to Yahoo Finance's historical market database. Yahoo Finance aggregates trade and quote data from U.S. exchanges and over-the-counter markets and publishes standardized time series for each ticker.

Dataset Characteristics

- Number of samples (rows): 2,223 daily observations for SMCI
- Number of features (columns): 7 engineered predictors (8 columns total including the target)

- Data types: All predictors and the target are numerical time–series features (floats) indexed by trading date; there are no categorical or geographical variables.

- Target variable: Next-day log return, defined as $\log(C_{t+1}) - \log(C_t)$ where C_t is the closing price on day t. This represents the one-day-ahead percentage change in price on a log scale.

Data Quality Analysis

Missing Values

There are two kinds of missing values in this project. First, the raw OHLCV data from Yahoo Finance has almost no NaNs, but a few isolated glitches (including some zero-volume days). These do not cluster in any obvious pattern and are treated as missing completely at random; I drop rows with missing OHLC and treat zero volume as missing, then forward-fill volume and drop any remaining NaNs. Second, the engineered rolling features and the one-step-ahead target introduce structural missingness at the start (insufficient lookback for rolling windows) and at the very end (no next-day price), which I remove by dropping rows with undefined features or target before modeling.

Class Balance

For the classification task, the dataset is reasonably balanced. In the training set, "up" days are 51.4 and "down" days are 48.6; in the validation set, "up" is 56.9 vs. 43.1; in the test set, "up" is 46.1 vs. 53.9. These proportions are close to 50–50 and do not indicate severe class imbalance. Given this mild imbalance, I do not apply any explicit balancing technique (no oversampling, undersampling, or class-Iighting). Models are trained on the original labels and evaluated against a majority-class baseline to ensure that small improvements over 50 accuracy are interpreted correctly.

Statistical Summary

The engineered features and target all behave like typical daily equity-return data. The 1-day log return and the next-day log-return target are tightly centered around zero with relatively small day-to-day movements, but both have noticeably heavy tails: there are a few large positive and negative jumps rather than a strictly "Gaussian" shape. The 5-day mean return is even more concentrated near zero, reflecting that averaging over several days smooths out much of the noise. The 10-day volatility feature is strictly positive and right-skeId, with most days showing low volatility and a smaller number of high-volatility episodes. The log volume change is roughly symmetric around zero, with occasional large spikes that correspond to big shifts in trading activity. The correlation structure of the data is also fairly simple. The various rolling mean returns are strongly correlated with one another and moderately correlated with the 1-day return, which is expected because they are overlapping averages of the same series. In contrast, volatility and log-volume change have much Iaker correlations with the return-based features. Most importantly, every feature shows only a very small correlation (close to zero) with the next-day log-return target. This indicates that there is no strong linear relationship betIen my predictors and the quantity I want to forecast, which helps explain why even relatively flexible models struggle to extract a robust predictive signal.

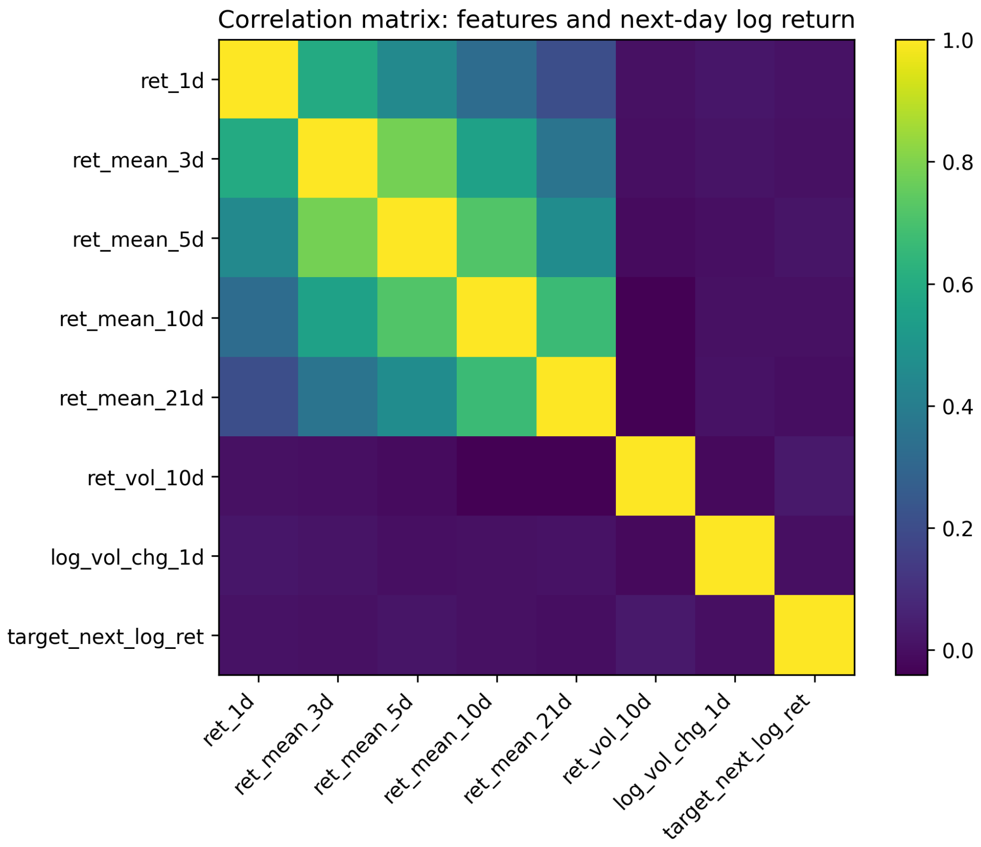

FIG. 1: Correlation matrix of engineered features and the next-day log-return target.

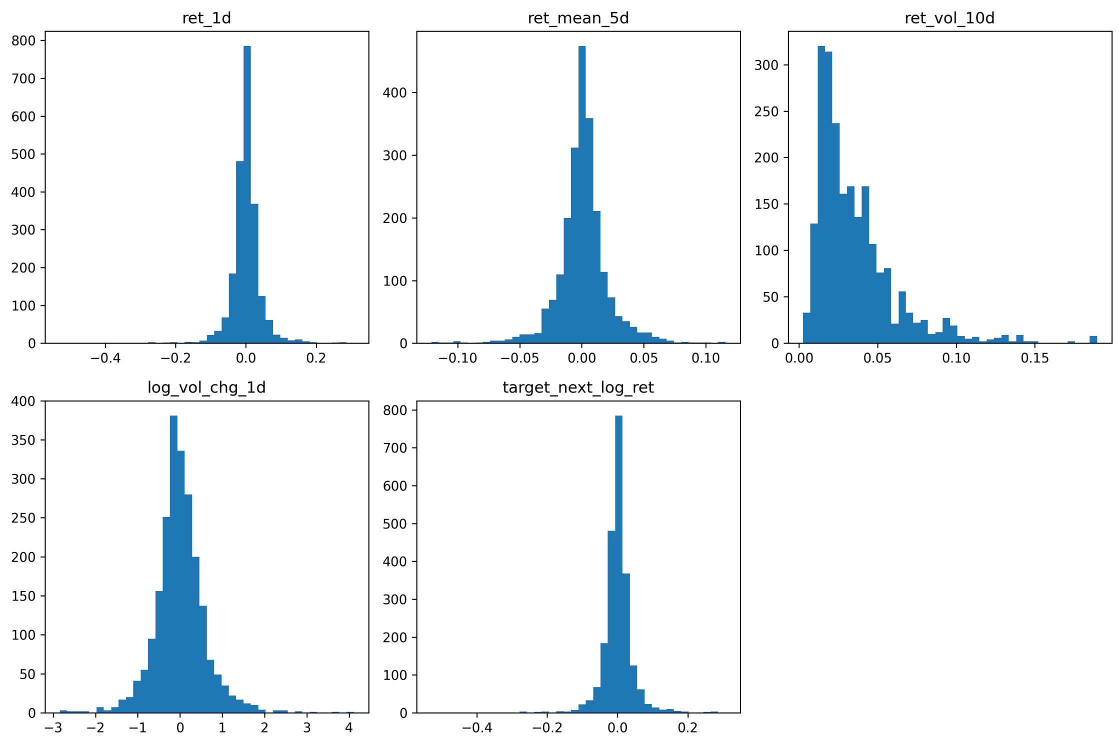

FIG. 2: Univariate distributions of key engineered features and the next-day log-return target for SMCI. All variables are centered near zero with heavy tails and a few extreme observations typical of daily equity returns.

PREPROCESSING

Data Splitting

I treat the SMCI data as a time series, so I use a chronological split rather than a random or stratified split. After constructing the features and the next-day log-return target, I sort all observations by trading date and divide them into three contiguous blocks: the earliest 60% of days for training, the next 20% for validation, and the most recent 20% for testing. This gives 1,333 training observations, 445 validation observations, and 445 test observations.

A time-based split matches the real prediction setting, where models are trained on past data and evaluated on future data. Randomly shuffling days would mix past and future, letting information from later periods leak into training and inflating performance. The validation set is used only for model and hyperparameter selection, and the test set is reserved for a final, unbiased performance estimate.

For the classification extension that predicts only the direction of the next-day return, I reuse the same time-ordered split. The proportions of up and down days in each split are already close to balanced, so no additional stratification or resampling is needed.

Feature Engineering

All features are constructed from past daily price and volume data so that they are available at the time of prediction and do not leak future information.

First, I transform closing prices to log space and compute the one-day log return,

$$r_t = \log(C_t) - \log(C_{t-1}),$$

which serves as a basic measure of recent movement. From this series I build several rolling statistics that capture short- and medium-term trends. Specifically, I compute rolling mean returns over 3, 5, 10, and 21 trading days. These features summarize recent momentum and help the models distinguish betIen quiet periods and sustained moves.

To represent risk, I compute a 10-day rolling standard deviation of the daily log returns, which acts as a simple volatility indicator. This captures whether the stock has been trading in a stable range or in a more turbulent regime.

Finally, I include trading activity through volume. I take the logarithm of daily volume to reduce skew, then compute the one-day change in log volume. This feature measures relative surges or drops in trading interest rather than absolute size.

The target variable is the next-day log return, defined as $\log(C_{t+1}) - \log(C_t)$. All predictors are based only on information up to day t, so the models attempt to map past returns, volatility, and volume changes to the next-day return.

Scaling, Transformation, and Encoding

I apply a few simple transformations to make the raw financial data more suitable for modeling. Closing prices and trading volumes are first converted to logarithms, which reduces skew and turns multiplicative changes into additive ones. From these log prices I compute daily log returns and rolling statistics, and from log volume I compute the one-day change in log volume. These transformations produce features that are roughly centered and comparable in scale.

For model-specific scaling, I standardize the input features when training Ridge regression and Logistic Regression. In both cases I use a StandardScaler inside a scikit-learn pipeline, which subtracts the mean and divides by the standard deviation of each feature using only the training data. This prevents features with larger numerical ranges from dominating the linear models. Tree-based models, namely Gradient Boosting and Random Forest, are trained on the original feature scales since they are largely insensitive to monotone feature scaling.

MACHINE LEARNING TASK AND OBJECTIVE

This section focuses on the machine learning aspect of the project.

Why Machine Learning?

Short-horizon stock returns are noisy and driven by many interacting factors, so it is very hard for a human or a simple rule to reliably spot patterns by eye. Traditional approaches often rely on a few hand-crafted technical indicators or on linear models with strong assumptions about how returns behave. These methods struggle when relationships are Iak, nonlinear, or unstable over time.

Machine learning gives us a way to learn directly from the data, using flexible models that can combine many features and capture subtle patterns if they exist. By training on historical returns and then testing strictly on future data, I can let the models search for predictive structure in an automated and systematic way, and I can quantify how much, if any, improvement they provide over naive baselines like "assume the return is zero."

Task Type

This is a supervised classification and regression task. In that setting, the target is a binary label indicating whether the next-day return is positive or not, so the models predict the direction of the move (up or down) rather than its exact size.

MODELS

Describe the machine learning models you will compare. You need at least three models in increasing order of complexity.

Model Selection

I focus on a small set of supervised models that cover both linear and nonlinear approaches. For the regression task, I use Ridge regression as my main linear model with L2 regularization, and a Gradient Boosting Regressor as a more flexible tree-based ensemble. For the classification extension, where the target is only the direction of the next-day return, I use Logistic Regression as a linear baseline and a Random Forest classifier as a nonlinear benchmark.

Hyperparameters for all models are chosen using the validation set, keeping the test set completely untouched until the final evaluation. In later subsections, I describe each model, its role in the comparison, and its performance in more detail.

Model 1: Ridge Regression

My first model is Ridge regression, a simple linear model with an L2 regularization term on the coefficients. Ridge learns a Iighted sum of the input features and shrinks the Iights toward zero, which helps control overfitting when features are noisy or correlated. In this project, the inputs are the engineered technical features based on past returns, rolling means, volatility, and volume changes, and the target is the next-day log return.

I use a scikit-learn pipeline that standardizes each feature and then applies Ridge regression. The only main hyperparameter is the regularization strength, which I select by evaluating several candidate values on the validation set and choosing the one with the loIst validation error. Ridge serves as a clear baseline that captures only linear relationships betIen the features and the next-day return, so any improvement from more complex models can be judged relative to this simple, interpretable starting point.

Model 2: Gradient Boosting Regressor

The second model is a Gradient Boosting Regressor, which builds an ensemble of many shallow decision trees. Trees are added one at a time, and each new tree is trained to reduce the errors of the current ensemble, so the model gradually improves its fit to the data. This allows gradient boosting to capture nonlinear relationships and interactions betIen features that a simple linear model like Ridge regression cannot represent.

I use the same engineered technical features as inputs and keep the next-day log return as the continuous target. The main hyperparameters are the number of trees, the maximum depth of each tree, and the learning rate. I explore a small grid of these settings and choose the combination that performs best on the validation set. Gradient boosting serves as my primary nonlinear regression model and lets us test whether more flexible function classes can extract any additional predictive signal from past returns, volatility, and volume changes.

Model 3: Random Forest Classifier

As a more complex model I extended the problem to a classification setting, I use a Random Forest classifier to predict only the direction of the next-day return. A random forest is an ensemble of decision trees, each trained on a bootstrap sample of the data and a random subset of features. The final prediction is obtained by majority vote across all trees. This design reduces the variance of individual trees and allows the model to capture nonlinear patterns and interactions in the features.

I reuse the same engineered technical features as inputs, but convert the target into a binary label that indicates whether the next-day log return is positive or not. The main hyperparameters are the number of trees and the maximum depth of each tree, which I tune on the validation set. The random forest serves as a flexible nonlinear benchmark for the direction prediction task and allows us to test whether more complex structure in the feature space leads to meaningfully better up or down forecasts than simpler linear models.

Regularization and Hyperparameter Tuning

Ridge regression uses L2 regularization on the coefficients. We select the regularization strength α by evaluating a small grid of values on the validation set and choosing the one with the lowest validation error.

For the Gradient Boosting Regressor, we tune a short list of settings for the number of trees, maximum tree depth, and learning rate, again picking the combination that performs best on the validation set.

In the classification setting, we keep Logistic Regression mostly at default settings and tune a small grid of hyperparameters for the Random Forest classifier, focusing on the number of trees and the maximum depth. In all cases, hyperparameters are chosen using only the training and validation data, and the test set is used once at the end for final evaluation.

TRAINING METHODOLOGY

Loss Functions

Model 1 (Ridge Regression). Ridge regression minimizes the mean squared error with an L2 penalty on the weights:

$$\mathcal{L}_{\text{Ridge}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\big(y_i - \mathbf{w}^T\mathbf{x}_i\big)^2 + \lambda\|\mathbf{w}\|_2^2. \tag{1}$$

Training is performed by scikit-learn using closed-form or iterative solvers. We track performance on the validation set while varying λ, and select the value that best balances low error and stability, then report final results on the test set to check for overfitting.

Model 2 (Gradient Boosting Regressor). Gradient boosting builds an additive model $F_M(\mathbf{x}) = \sum_{m=1}^{M} \nu f_m(\mathbf{x})$ and minimizes the mean squared error:

$$\mathcal{L}_{\text{GB}} = \frac{1}{n}\sum_{i=1}^{n}\big(y_i - F_M(\mathbf{x}_i)\big)^2. \tag{2}$$

Each new tree f_m is fitted to the residuals of the current model. We control overfitting by limiting tree depth, the number of trees M, and the learning rate ν, and choose among a small grid of settings based on validation error, then evaluate once on the test set.

Logistic Regression (direction prediction). For the binary label $y_i \in \{0,1\}$ indicating whether the next-day return is positive, Logistic Regression minimizes the regularized cross-entropy loss:

$$\mathcal{L}_{\text{LogReg}}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \log\sigma(\mathbf{w}^T\mathbf{x}_i) + (1-y_i)\log\big(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\big)\Big] + \lambda\|\mathbf{w}\|_2^2, \tag{3}$$

where $\sigma(z) = 1/(1+e^{-z})$. Training uses gradient-based optimization in scikit-learn, and we monitor accuracy on the validation set to avoid over- and under-fitting.

Random Forest Classifier (direction prediction). Random Forest does not optimize a single global loss, but each individual decision tree is grown by choosing splits that reduce node impurity, typically measured by the Gini index:

$$G(t) = \sum_k p_{k,t}\big(1-p_{k,t}\big), \tag{4}$$

where $p_{k,t}$ is the proportion of class k in node t. The forest prediction is the majority vote over all trees. To control overfitting, we limit tree depth and choose the number of trees based on validation accuracy, then evaluate the final model on the test set.

Training Process

I treat the problem as a time series task, so I do not use k-fold cross-validation. Instead, I split the data chronologically into training, validation, and test sets, and I always train on the earliest data and evaluate on later periods to avoid look-ahead bias.

For Ridge regression, I train a sequence of models with different values of the regularization parameter. I fit the model on the training set and record the validation mean squared error, then choose the value that gives the lowest validation error. I follow a similar procedure for the Gradient Boosting Regressor. I define a small grid over the number of trees, maximum depth, and learning rate, train one model for each combination, and keep the configuration that performs best on the validation set.

In the classification setting, I train Logistic Regression once with standardized features, and I train several Random Forest classifiers with different numbers of trees and depths. I select the Random Forest configuration with the highest validation accuracy.

To monitor learning and guard against over- or under-fitting, I focus on how training and validation metrics change across hyperparameter choices. I use only the training and validation sets for model selection, and I evaluate the final chosen models once on the held-out test set to obtain an unbiased estimate of out-of-sample performance.

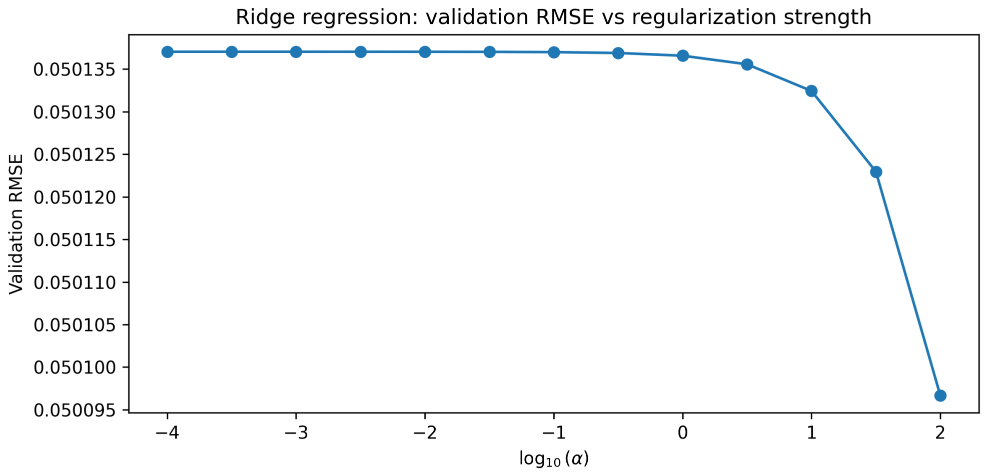

FIG. 3: Validation RMSE for Ridge regression as a function of the regularization parameter.

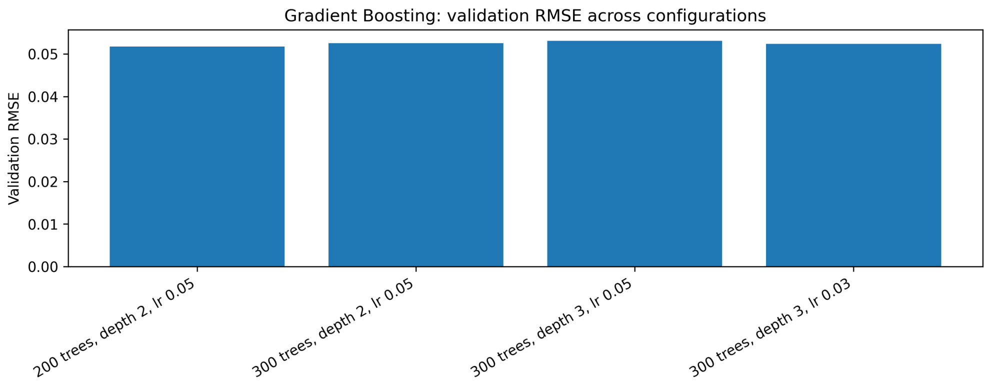

FIG. 4: Validation RMSE for different Gradient Boosting configurations, varying the number of trees, maximum depth, and learning rate.

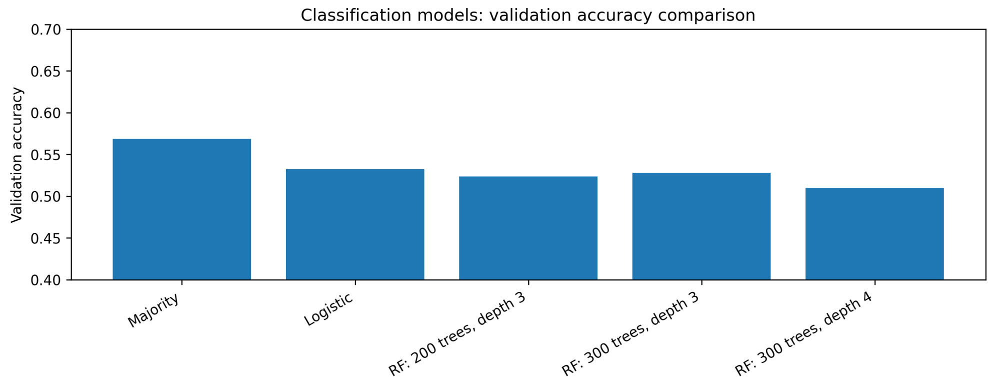

FIG. 5: Validation accuracy for the majority baseline, Logistic Regression, and several Random Forest configurations on the direction prediction task.

Model Summary Table

TABLE I: Summary of models, parameters, and training methodology.

Model	**Parameters**	**Hyperparameters**	**Loss Function**	**Regularization**
Model 1: Ridge	$\mathbf{w}, b$	α	MSE	L2
Model 2: GB Regressor	Tree weights	$n\_estimators$, depth, lr	MSE	Tree constraints
Model 3: RF Classifier	Tree splits	$n\_estimators$, depth	Gini impurity	Bagging

METRICS

Primary Metric

My primary evaluation metric for the regression task is Root Mean Squared Error (RMSE). RMSE measures the typical size of the prediction error in the same units as the target, so it is easy to interpret as an average error in next-day log return. I consider a model to be doing well only if its test RMSE is clearly lower than simple baselines that predict zero or the in-sample mean return.

Secondary Metrics

For regression I also report Mean Absolute Error (MAE) and the coefficient of determination R^2. MAE gives the average absolute error and is less sensitive to outliers. The R^2 score compares the model to a constant baseline and shows how much of the variance in the target is explained.

For the classification task, where the goal is to predict only the direction of the next-day return, I use accuracy as the main metric and also report precision, recall, and F1 score for the positive (up) class. In addition, I look at trading related summaries such as hit rate on the sign of the return and the Sharpe ratio, but these are mainly used for interpretation rather than as primary optimization targets.

Metric Definitions

For regression, with true values y_i and predictions $\hat{y}_i$:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{5}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{6}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{7}$$

where $\bar{y}$ is the mean of the true values.

For binary classification, with true labels y_i and predicted labels $\hat{y}_i$, let TP be true positives, FP be false positives, TN be true negatives, and FN be false negatives. Then

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}, \tag{8}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{9}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{10}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{11}$$

RESULTS AND MODEL COMPARISON

Performance Comparison

TABLE II: Model performance metrics on test set (regression task).

Model	**RMSE**	**MAE**	R^2	**Sharpe**
Constant-mean baseline	0.0632	0.0432	-0.0015	-0.4831
Ridge Regression	0.0634	0.0436	-0.0091	-0.4711
Gradient Boosting Regressor	0.0642	0.0440	-0.0342	-0.7145

All three regression models perform almost the same on the test set. RMSE and MAE differ only in the third decimal place, and every model has a slightly negative R^2, which means none of them beat a constant prediction by much. When I turn the predictions into a simple long-or-flat trading rule, all Sharpe ratios are negative and close to zero, with Gradient Boosting actually doing a bit worse than the simpler models.

Ridge regression is the easiest and fastest model to train, since it is just a linear model with one main hyperparameter. Gradient Boosting is more complex and takes longer to tune, but does not deliver better test performance. Given this, I view the constant-mean baseline and Ridge regression as the most reasonable choices for this task. Ridge is my preferred "best" model among the learned regressors: it is simple, cheap to train, and performs essentially as well as the more flexible Gradient Boosting model, which supports the conclusion that short-horizon SMCI returns are very hard to predict with these features.

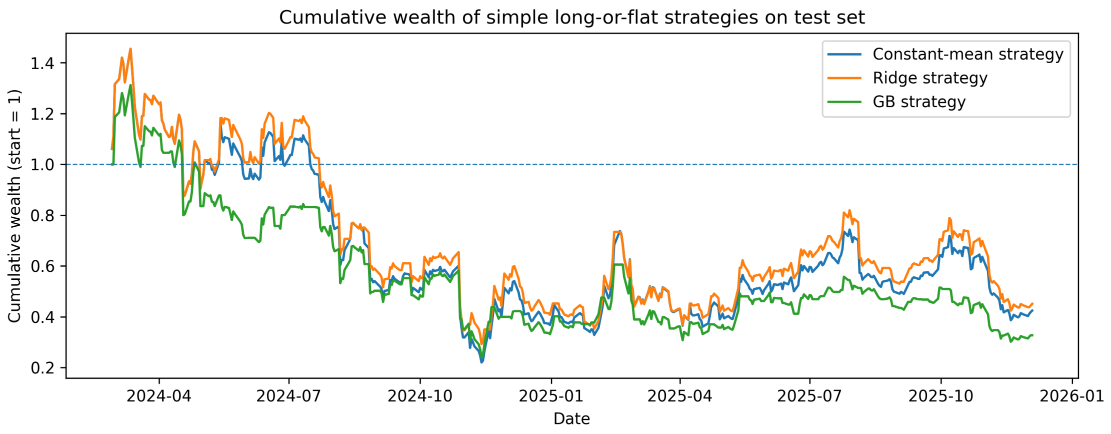

FIG. 6: Cumulative wealth of simple long-or-flat strategies on the test set, based on the constant-mean prediction, Ridge Regression, and Gradient Boosting. All curves remain near or below the starting value of one, which shows that none of the models yields a robust, profitable trading strategy.

Computational Efficiency

TABLE III: Training and inference time for each model.

Model	**Training Time (s)**	**Inference Time (s)**	**Hardware Used**
Ridge Regression	0.0041	0.0006	CPU (laptop)
Gradient Boosting Regressor	0.2362	0.0004	CPU (laptop)
Random Forest Classifier	0.1510	0.0263	CPU (laptop)

Ridge Regression trains almost instantly and makes predictions

Analysis and Discussion

The results show that no model extracts a strong predictive signal from these features. On the regression task, the constant-mean baseline, Ridge Regression, and the Gradient Boosting Regressor all have very similar RMSE and MAE on the test set, and all three have slightly negative R^2 values. This means that, on average, they do not beat the simple strategy of predicting the same constant return every day. When I convert the predictions into a long-or-flat trading rule, the annualized Sharpe ratios are negative and close to zero, which confirms that none of the models produce a profitable or stable strategy.

Given this, I treat Ridge Regression as the best learned model for the task. It is extremely fast to train and evaluate, uses only one main hyperparameter, and performs essentially as well as Gradient Boosting while being simpler and easier to interpret. Gradient Boosting is more flexible and more expensive to train, but it does not improve test performance and in fact produces a slightly worse Sharpe ratio.

The classification extension leads to a similar conclusion. Logistic Regression and the Random Forest classifier achieve test accuracies only a few points above the majority-class baseline, with comparable precision, recall, and F1 scores. This small gain is not convincing in a noisy financial setting. Overall, the evidence suggests that, with the engineered features and one-day horizon used here, short-term SMCI returns are very close to random from the model's point of view, and that increasing model complexity does not meaningfully improve performance.

MODEL INTERPRETATION

Feature Importance

After choosing Ridge Regression as my best model, I interpret its coefficients as a simple measure of feature importance. Since the inputs are standardized, features with larger absolute coefficients contribute more to the prediction of the next-day return. I fit the final Ridge model on the combined training and validation data and then inspect the absolute values of its weights.

Overall, no single feature dominates the model, which is consistent with the weak predictive performance. The short-horizon return features and their rolling means have slightly larger coefficient magnitudes than the volatility and volume-based features, which suggests that recent price movement carries the most signal, although that signal is still very small. To cross-check this, I also look at tree-based feature importances from the Gradient Boosting model. They show a similar pattern: recent returns and short rolling means receive somewhat higher importance scores, but the differences across features are modest. This supports the idea that all of the engineered features contain only weak, noisy information about the next-day return.

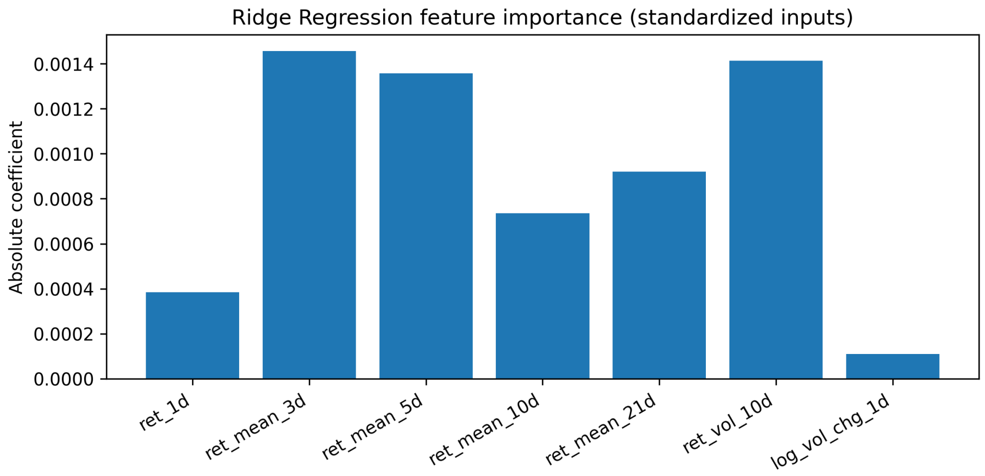

FIG. 7: Feature importance for the final Ridge Regression model, measured by the absolute value of the standardized coefficients.

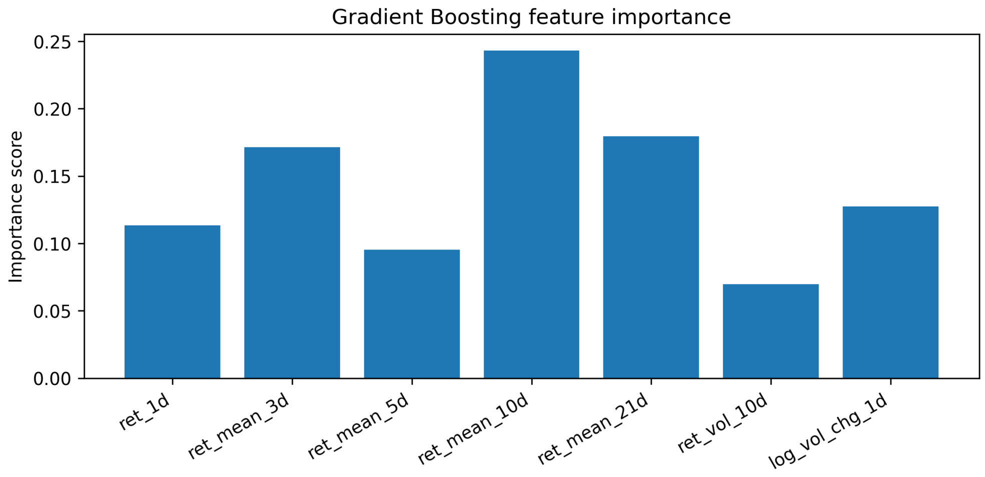

FIG. 8: Feature importance scores from the final Gradient Boosting model.

Model Behavior Analysis

The Ridge Regression model makes predictions as a linear combination of the standardized features, so each feature nudges the prediction up or down in proportion to its coefficient. The plot of true versus predicted next-day returns on the test set shows that almost all predictions lie in a very narrow band around zero, while the true returns vary much more. The points are spread horizontally with almost no visible alignment along the 45-degree line, which confirms that the model has very limited ability to track large positive or negative moves.

To understand which inputs drive the small variation that does exist in the predictions, I plot the predicted next-day return against two of the most important features: the 3-day mean return and the 10-day volatility. The model tends to predict slightly lower next-day returns after negative recent performance (ret_mean_3d) and slightly higher returns when recent volatility (ret_vol_10d) is higher. However, both relationships are weak and nearly linear, and the absolute size of the predictions remains small. These plots reinforce the conclusion that the model mostly shrinks predictions toward zero and that the engineered features contain only a weak signal about the next-day return.

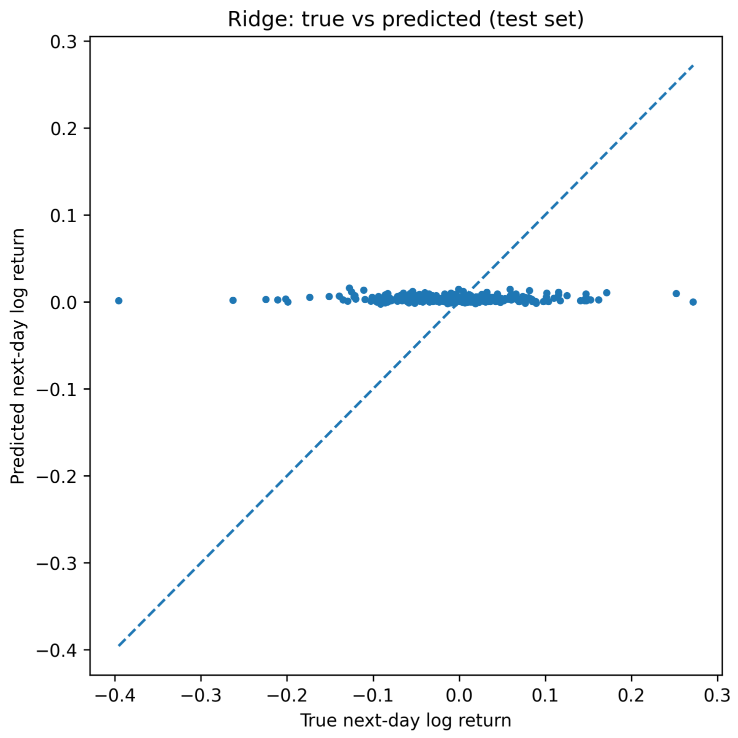

FIG. 9: True versus predicted next-day log returns on the test set for the Ridge Regression model. Predictions are tightly clustered near zero, while true returns vary more widely, indicating weak predictive power.

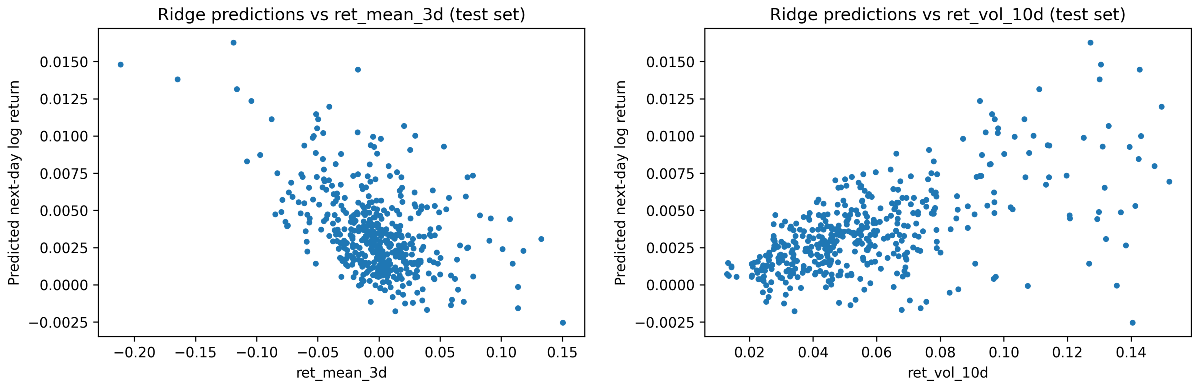

FIG. 10: Ridge predictions versus two important features on the test set. Left: a slight negative relationship with the 3-day mean return. Right: a slight positive relationship with 10-day volatility. In both cases, the effect is weak and predictions stay close to zero.

CONCLUSION

Summary of Findings

In this project I tried to predict the next-day log return of SMCI using engineered features based on recent returns, rolling means, volatility, and volume changes. I compared a constant-mean baseline, Ridge Regression, and a Gradient Boosting Regressor for the regression task, and I also looked at Logistic Regression and a Random Forest classifier for predicting only the direction of the next-day move. On the test set, all regression models achieved very similar RMSE and MAE, and all had slightly negative R^2, which means they did not improve over a constant prediction. The simple long or flat strategy based on the model outputs also produced Sharpe ratios that were negative and close to zero. Because Ridge Regression is fast, simple, and performs just as well as the more complex models, I consider it the best learned model in this setting, but it still does not reach a practically useful level of predictive accuracy.

Limitations and Future Work

The main limitation is that the features I used contain only a weak signal about next-day returns. They are built from a single stock, a short lookback window, and standard technical indicators, so there may simply not be much predictable structure at this horizon. Another limitation is that I used relatively simple models and a straightforward time split, without more advanced regularization or model averaging. In future work, I could enrich the feature set with information from the broader market, option-implied measures, or news and sentiment, and explore different horizons such as weekly or monthly returns where predictability might be stronger. I could also try more sophisticated models, such as gradient-boosted trees with richer tuning, or sequence models that operate directly on price paths, while still maintaining a careful time-series evaluation.

Final Remarks

Overall, this project shows how to build an end-to-end pipeline for a financial prediction task, from data collection and feature engineering through model training, tuning, and evaluation. The negative result is informative: even with several modern machine learning models, short-horizon SMCI returns remain very hard to forecast in a way that is both statistically and economically meaningful. The exercise reinforced the importance of strong baselines, proper time-aware data splitting, and realistic performance metrics. It also highlighted that in noisy domains like finance, improving the quality and diversity of features may matter more than adding model complexity, and that honest out-of-sample evaluation is essential before trusting any predictive model.

* [pottsbr1@msu.edu]

[1] Yahoo Finance, "Super Micro Computer, Inc. (SMCI) historical data," *finance.yahoo.com*, accessed 2025.

[2] Ran Aroussi, "yfinance: Download market data from Yahoo! Finance's API," GitHub repository, https://github.com/ranaroussi/yfinance, accessed 2025.

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer (2009).

[5] OpenAI, "ChatGPT (GPT-5.1 Thinking)," large language model, https://chat.openai.com, accessed 2025.