Project 2

Author

Jinyan Yong

Course Title

FIN704

Date

March 02, 2025

Instructor:

Michael Milewski

Contents

1 Introduction and Summary

1.1 Problem Statements

Realized volatility (RV) is a fundamental object in financial prediction, portfolio allocation, and risk control.In financial applications, realized volatility — a high-frequency measure of market variability — is of particular interest for forecasting risk and uncertainty. Traditional models such as the Heterogeneous Autoregressive (HAR) model proposed by Corsi (2009) have demonstrated that volatility exhibits long memory. However, these linear models may fail to capture complex nonlinear patterns in financial time series data. Therefore, a more complex model is applied. In this project, we try to use deep learning techniques to forecast the next-day realized volatility (i.e., $rv5_{t+1}$) of the S&P 500 index, and explore if external data like VIX and interest rates help in prediction. In our final report, we have conducted both univariate and multivariate analysis based on four models, Deep Neural Network (DNN), Deep & Wide Neural Network, Recurrent Neural Network(RNN) and Long-Short-Term-Memory(LSTM).

2 Data Engineering

We collected realized volatility (rv5) from the Oxford-Man Institute and external macroeconomic variables from FRED, including the VIX index (`vixcls`) and a set of US Treasury rates of different maturities (`DGS1`, `DGS2`, `DGS10`, `DGS30`, `DTB3`).

In total, we built one univariate model and three multivariate models, where we only used a univariate input (rv5) first, and then we extended our analysis by introducing macroeconomic indicators to test whether external information improves our volatility forecasting. Specifically, we designed 4 models: one with rv5 only, and three multivariate models including:

- all macro variables (VIX and interest rates),
- interest rates only,
- VIX only.

This variable selection is supported by the following financial theory.

- **Interest rates** reflect monetary policy and investor expectations about the macroeconomic environment, which are known to impact volatility through risk-free discounting and capital flow mechanisms.
- The **VIX index**, often referred to as the "fear gauge," directly measures market-implied volatility and thus is highly correlated with short-term movements in realized volatility.

Therefore, by exploring different combinations, we aimed to isolate the *marginal contribution* of each group of variables to model performance and understand which type of information offers the most predictive power in practice.

3 Exploratory Data Analysis

We begin our analysis by examining the raw volatility data before preprocessing. The original dataset contains 4641 daily observations of realized volatility (`rv5`) for the S&P 500 index, ranging from 2000 to mid-2018. However, several macroeconomic variables such as interest rates (`DGS1`, `DGS2`, etc.) and the `VIXCLS` index were stored as object types due to the presence of non-numeric characters. We also observed missing entries in these columns.

To clean the data, we first converted the `rv5` values to percentage squared units by multiplying by 100^2. We then replaced missing values and string placeholders with `NaN`, applied numeric coercion, and used linear interpolation to fill the gaps. Finally, rows with any remaining missing values were dropped.

- **Final Dataset Size:** 4641 entries, with 9 numerical columns after cleaning.
- **rv5 Statistics:** The realized volatility (`rv5`) has a mean of 1.096, a standard deviation of 2.48, and a maximum value of 77.48.

Figure 1 shows the time series of daily realized volatility, with sharp spikes around crisis periods (e.g., 2008). The distribution of `rv5` (Figure 2) is right-skewed, with the majority of values clustered below 5.

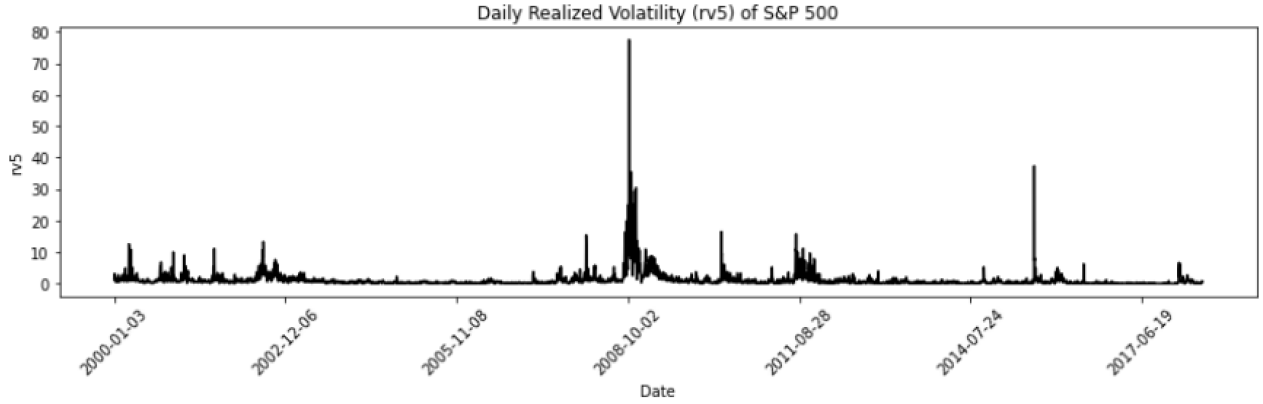

Figure 1: Daily Realized Volatility (`rv5`) of S&P 500

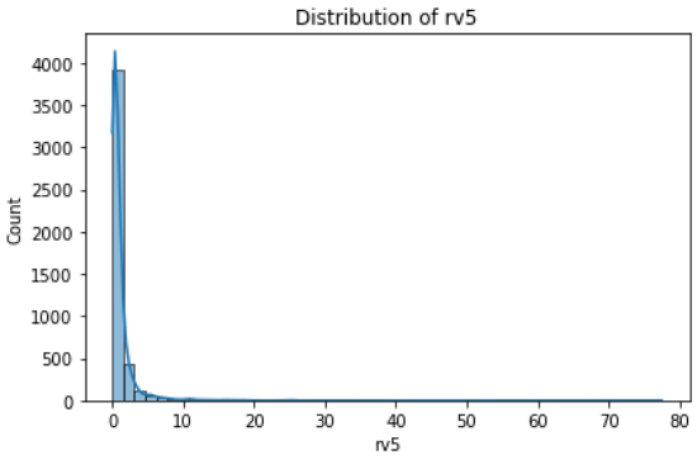

Figure 2: Distribution of `rv5`

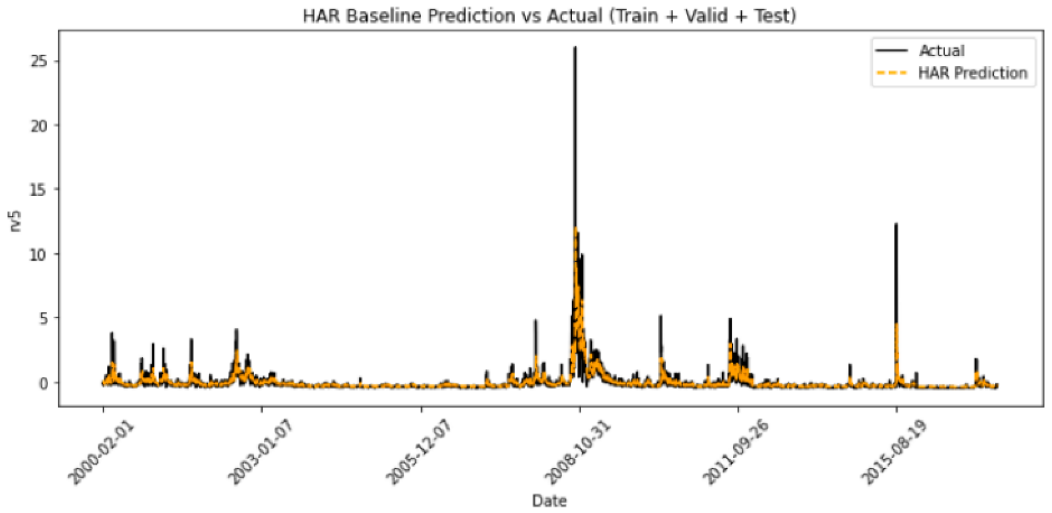

Figure 3: HAR Baseline Prediction vs Actual (Train + Valid + Test)

4 Baseline Model

For the baseline linear model (HAR) named Model 1 in LSTM code, we used lagged volatility values including the daily realized volatility (rv5), its 5-day moving average (rv5_5), and 21-day moving average (rv5_21). One thing to note about the baseline model is that we didn't apply inverse transformation in the preliminary report, making the error value too small and losing the meaning of comparison.

Model	Train MSE	Validation MSE	Test MSE
HAR (Adjusted)	3.78	2.05	0.23

Table 1: Adjusted MSE values for HAR model after inverse transformation

Additionally, it is also worth noting that the test set may corresponds to a relatively calm market period (2017 and later) with low and stable realized volatility. A linear structure of the HAR model makes it easier to fit and extend such a stable trend, making a relatively simple model can achieve a relatively low test error.

5 Single/Multi LSTM

5.1 Introduction and Model Motivation

Our project aims to forecast the daily realized volatility (RV) of the S&P 500 index using neural network-based models. Traditional models such as the HAR model have demonstrated that volatility exhibits long memory. However, there is some evidence showing that these linear models may fail to capture complex nonlinear patterns in financial time series data.

Recent studies (e.g., Bucci 2020; Bhandari et al. 2022) have shown that Neural Networks , particularly Long Short-Term Memory (LSTM) networks, are capable of modeling such dependencies and may outperform linear benchmarks under certain conditions.

In this study, we focus on building and comparing single-layer and multi-layer LSTM models for RV forecasting. Our goal

is to examine whether deep LSTM structures improve prediction accuracy, and to assess the effectiveness of incorporating external financial indicators such as the VIX index and treasury yields.

5.2 Model Description

We implement a multi-layer/single-layer(since we can adjust the number of layers, when the number comes to 1, which should be a single-layer model) LSTM model using the `Keras` API with TensorFlow backend. The architecture follows a sequential design tailored for one-step-ahead forecasting of the S&P 500 index's realized volatility (`rv5`).

5.2.1 Brief Description

LSTM (Long Short-Term Memory) is a type of neural network that's really good at handling time series data. Unlike regular neural networks(can only remember recent patterns), LSTMs can remember patterns over time, which is especially crucial for financial data, where past behavior can influence future trends.

What makes LSTMs special is adding more structure. LSTM uses gates to selectively keep or forget information. That's why LSTM can "remember" useful patterns from longer ago, while RNN tends to forget, additionally, RNN updates its memory at every step without much control.

5.2.2 Data Engineering

After the stage of preliminary report, instead of using univariable we added the VIX index, and interest rates, and combined them differently to explore if external data can contribute to predictive performance. In the LSTM code, they are named as follows:

- **Model 2**: `rv5` only (univariate)
- **Model 3**: `rv5` + all macro variables (VIX + interest rates)
- **Model 4**: `rv5` + interest rates only
- **Model 5**: `rv5` + VIX only

5.2.3 Model Structure

This section outlines the full LSTM modeling pipeline, excluding hyperparameter tuning details, which will be covered separately.

Data Preprocessing All input features are standardized using `StandardScaler` fitted on the training set to prevent data leakage(we can not apply a StandardScaler on the entire dataset. The same transformation is applied to validation and test sets. Additionally, the target variable `rv5_f1` is also standardized.

Dataset Splitting Chronological split is applied:

- Training: up to 2012
- Validation: 2013 to 2016
- Test: from 2017 onward

Sliding window datasets are generated using `timeseries_dataset_from_array` from TensorFlow. Following a chronological split instead of mixing data from different year, we can use past data to forecast the future. And one of important reasons for this time-based reason is that market conditions can change over the years, like interest rate policies, volatility, or macroeconomic trends.

Model Architecture

- **Input shape:** A sliding window of `sequence_length` time steps, each with multiple features (e.g., `rv5`, interest rates, and VIX) so input shape should be `(sequence_length, number_of_features)`.
- **LSTM layers:**
  - 1 to 4 stacked LSTM layers, each with ReLU activation.
  - The number of nodes per layer is proportional to `sequence_length`, scaled by 1.0 or 1.5.
  - For multiple layers, `return_sequences=True` is used except for the last.
- **Output:** A single neuron `Dense(1)` to predict next-period volatility (`rv5_f1`).
- **Training Details:**
  - Epochs: 50
  - Batch size: 128
  - Shuffle: Disabled (to maintain time order)
  - Optimizer: Adam (SGD prepared but not used in this round)
  - Loss: MSE
  - Early stopping: patience of 10 on validation MSE

Helper Functions Two helper functions are defined to convert predictions back to the original scale and calculate MSE accordingly:

- `tx_helper_predict()`: This function reverses the standardization process applied during our preprocessing. Using the mean and standard deviation from the training set `StandardScaler`, it transforms the model's output from standardized values back to their original scale. Additionally, it is also applicable to predictions on the validation and test sets, since the same scaler was used for their standardization.

- `tx_helper_mse()`: This function compares the model's de-standardized predictions to the true values and then computes MSE, which skips the first few steps because the model needs that initial window (the sequence length) to generate the first prediction, avoiding mismatch.

Model Selection and Inverse Transformation During random search, all loss values and predictions are kept in standardized scale. However, after selecting the best hyperparameters, a final model is retrained and evaluated with inverse-transformed predictions using the helper function `tx_helper_predict()`. Final train/valid/test MSEs are computed on the original scale using `tx_helper_mse()`.

Model Evaluation Process After selecting the best hyperparameters via random search, the model will be trained using the same training-validation split and and we will evaluate using validate error.

- **During training:** Loss values (`loss`, `val_loss`) are recorded on standardized data. These are visualized in the loss curve to monitor convergence and potential overfitting. (If there is abnormal convergence, we can adjust the learning rate. If the loss curve shows an upward trend in the validation set, may indicating overfitting.)
- **After training:** Final predictions on the training, validation, and test sets are made using the best model.
- **De-standardization:** Predictions are transformed back to the original scale using `tx_helper_predict()`, based on the training-set `StandardScaler`.
- **Final Evaluation:** De-standardized predictions are compared to true values using `tx_helper_mse()` to compute final MSEs on the original scale.
- **Visualization:**
  - `plot_lstm_predictions()` shows how well the model tracks the actual volatility in different data segments.
  - `plot_loss_curve()` helps interpret convergence behavior over epochs.
- **Comparison Table:** Finally, MSEs for all model configurations (models 2–5) are reported in a table to assess which variable combination yields the best predictive performance.

Loss Function: We use `mean squared error (MSE)` as the primary loss function during training. MSE is compatible with gradient descent and can effectively penalize large deviations, fitting with our time series model(especially the high volatility observed during financial crises). Although Huber loss was explored in preliminary tests, it was ultimately excluded to maintain consistency across trials and the baseline model.

5.3 Hyperparameter tuning

5.3.1 Hyperparameter tuning in preliminary stage

For the preliminary stage we manually tuned the hyperparameters to observe the model improvement step by step, which really helpful for us to narrow down the RandomSearch Hyperparameter tuning later. Our model development focuses

on tuning three key architectural hyperparameters for LSTM: **sequence length**, **number of layers**, and **number of nodes**. The objective is to identify a simple yet effective model configuration using validation set MSE as the evaluation metric.

During the first three stages, we fixed the training settings to:

- Optimizer: Adam
- Learning rate: 1e-3
- Activation function: ReLU

Step 1: Tuning Sequence Length. We began by testing three different sequence lengths: 21, 56, and 126, with the number of layers fixed at 2 and nodes at 64. This allows the model to determine its own memory horizon, as suggested by the instructor, instead of using the MA lags from the Corsi (2009) HAR model.

Step 2: Tuning Number of Layers. Using the best sequence length from Step 1, we tested 1, 2, 4 and 5 LSTM layers. Increasing depth may help the model capture more complex temporal dynamics, though with a risk of overfitting.

Step 3: Tuning Number of Nodes. Next, we tuned the number of LSTM units per layer. Since nodes represent the hidden memory size for each time step, we chose values that were **not too small relative to the sequence length**. Specifically, we tested:

- 64 (baseline)
- 128 (approx. double the initial sequence length)
- 256 (larger capacity for complex patterns)

Our intention was to avoid setting the node number below the sequence length, as a small hidden size can limit the model's learning capacity, especially in financial volatility forecasting.

Step 4: Tuning Optimizer, Learning Rate, and Activation Function. After identifying the best architecture (sequence length, layers, nodes), we conducted a second-stage tuning with:

- Learning rates: 1e-3, 1e-4
- Optimizers: Adam, SGD
- Activation functions: ReLU, Tanh

This step focused on improving convergence and model generalization.

The best-performing model will be selected based on test MSE and used for comparison with the HAR baseline in the final report.

Conclusion from preliminary stage After identifying the optimal values for `sequence_length`, number of layers, and node scaling, we further tested various combinations of learning rate, activation function, and optimizer. The results are summarized in the following table:

Learning Rate	Optimizer	Activation	Train MSE	Valid MSE	Test MSE
0.001	adam	relu	2.8078	1.2466	0.7970
0.001	adam	relu	5721.4052	2.0627	1.9893
0.001	adam	tanh	3.7587	1.4311	0.6329
0.001	sgd	relu	2.0674	2.1469	2.0809
0.001	sgd	tanh	1.8610	1.4658	1.2232
0.0001	adam	relu	2.7691	1.1390	0.6516
0.0001	adam	tanh	2.9578	1.1558	0.7551
0.0001	sgd	relu	2.8143	2.9923	2.9894
0.0001	sgd	tanh	2.4509	2.4485	2.3468

Table 2: Manual Hyperparameter Tuning Results (Sequence Length = 56, Layers = 2, Nodes = 64). Best validation MSE is highlighted.

5.3.2 Hyperparameter tuning using RandomSearch

Based on the preliminary results, which provided useful insights, we were able to narrow down the list of hyperparameters for RandomSearch:

- **Sequence Length:** Explored from 7 to 70 with a step of 7, enabling finer resolution near the optimal lag window.
- **Node Scaling:** Since in the previous optimal model, I found that the node should be close to sequence length, so setting the node scale to 1.0 or 1.5 times the sequence length.
- **Number of Layers:** Ranged from 1 to 4 stacked LSTM layers, covering both single-layer and multi-layer configurations.
- **Activation:** Fixed to ReLU for its computational efficiency and ability to mitigate vanishing gradients.
- **Optimizer:** Adam was selected due to its adaptive learning rate and robustness across settings.

5.4 Performance Evaluation and Visual Results

Table 3 summarizes our final performance of the four tuned LSTM models using the best hyperparameters obtained from the previous RandomSearch strategy. I applied hyperparameters tuning simultaneously across to all four univariate and multivariate models to ensure a fair and systematic comparison. We didn't use manually selected best hyperparameters to find the best multivariable configuration first and only see it as a guideline to find effective RandomSearch hyperparameters lists, as such an approach may lead to biased conclusions.

From the results above, **Model3** introducing both VIX and interest rate achieved the best validation performance with a MSE of 0.7614, closely followed by Model5. While Model5 achieved the best test MSE (0.5846), our model selection prioritizes validation performance as it is the main basis for early stopping and hyperparameter tuning. Adding external variables (VIX and interest rates) in Model 3 led to improved validation performance, suggesting that incorporating

Model	Train MSE	Valid MSE	Test MSE	Seq Len	Layers	LR	Activation	Units	Optimizer
Model2	1.6811	0.9821	0.6257	7	1	0.0010	relu	7	adam
Model3	4.1691	0.7614	0.6306	7	1	0.0010	relu	7	adam
Model4	3.1587	3.2748	3.0788	56	4	0.0001	relu	56	adam
Model5	4.3062	0.9785	0.5846	7	1	0.0010	relu	7	adam

Table 3: Performance of final LSTM models using tuned hyperparameters. Best validation MSE is highlighted.

macroeconomic features enhances predictive accuracy. Additionally, as you can the hyperparameter of LSTM ses the lower bounds of the hyperparameter ranges, which may indicate that if we introduce variables, simpler architecture is preferred. For comparison, the baseline HAR model yields the following MSE values:

- Train MSE: 3.7840
- Validation MSE: 2.0474
- Test MSE: 0.2347

Although HAR test MSE is better than LSTM, it suffers from relatively poor validation performance. However, our LSTM models better capture temporal dependencies and exhibit more stable validation performance, even if slightly underperforming on the test set.

Figure 4 shows the standardized loss curve of Model3. The training loss (red dashed) and validation loss (blue) both decrease and stabilize, showing good convergence. The validation curve flattens after around 10 epochs, and there's no major gap between train and validation loss, suggesting good generalization and a well-balanced model. The green dotted line marks the test loss (standardized) at 0.0356, confirming strong generalization.

Figure 5 visualizes predicted vs actual volatility values across training, validation, and test periods. The model captures the overall trend of realized volatility (rv5) across training (red), validation (blue), and test (green) periods. Predictions align well with the actual values (black), especially during stable periods. Even though slight deviations appear during extreme spikes, it is typical in volatility modeling.

Lastly, to validate is single-layer LSTM with external data performs best, I also tried adjusting the lower range of the LSTM layer to 2 to see what would happen, if a multi-layer LSTM would outperform. The outcomes are as follows:

Table 4: Multi-layer LSTM Final Model Comparison Table

Model	Train MSE	Valid MSE	Test MSE	Seq Len	Layers	LR	Activation	Units	Optimizer	Loss
Model2	1.633676	0.960410	0.472885	28	3	0.001	relu	28	adam	mse
Model3	4.879426	1.035481	1.152986	7	2	0.001	relu	7	adam	mse
Model4	48.268534	1.265244	1.876565	28	3	0.001	relu	28	adam	mse
Model5	2.468798	1.230120	0.842472	7	2	0.001	relu	7	adam	mse

For this situation, I did not observe a better validation result, and the univariate model outperforms in multi-layer case.

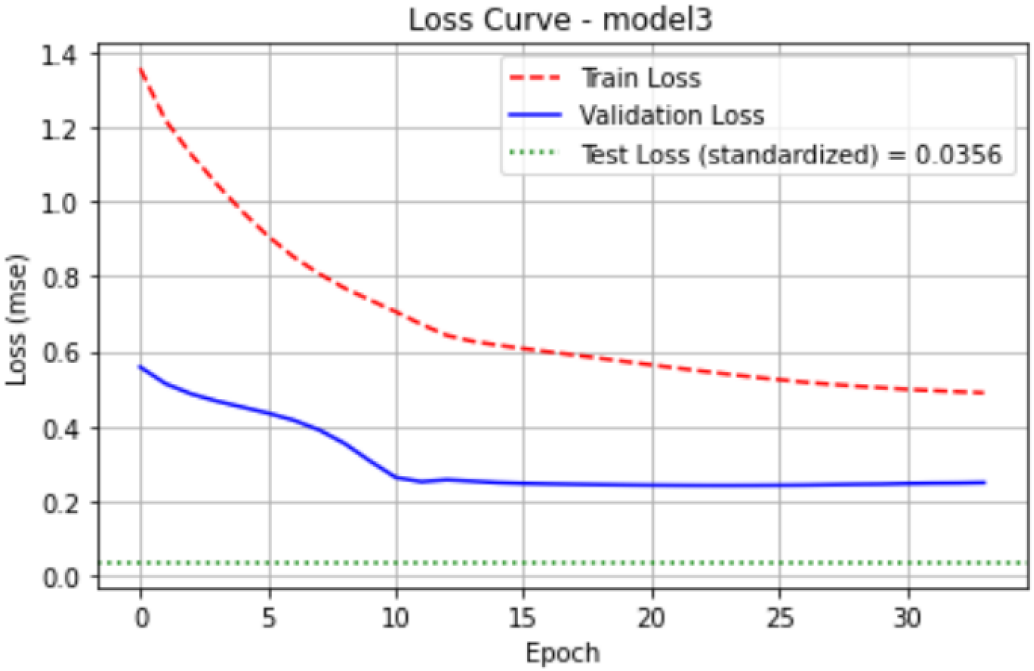

Figure 4: Loss curve of Model3

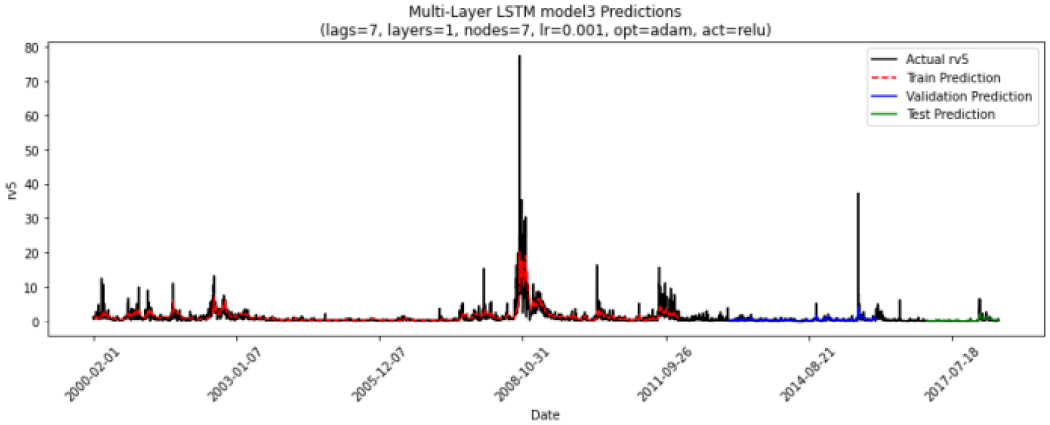

Figure 5: Prediction results of Model3 (lags=7, layers=1, nodes=7, lr=0.001, opt=adam, act=relu).

5.5 Conclusion

Among the four univariate and multivariate LSTM models, Model 3 demonstrated the best validation performance by effectively leveraging both VIX and interest rates. While Model 5 achieved the lowest test error, Model 3 showed the most stable and generalizable results across all datasets, highlighting the value of incorporating comprehensive macroeconomic features.