

Florida International University
Knight Foundation School of Computing and Information Sciences

Computer Science Focus

Final Deliverable

Project Title: Predicting Stock Returns Using Machine Learning

Team Members: Erick Cevallos, Gavin Greene, Ryan Echevarria, Alexander Hernandez

Product Owner(s): Erick Cevallos

Instructor: Masoud Sadjadi

The MIT License (MIT)
Copyright (c) 2026 *Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

This document presents the final deliverable for the Predicting Stock Returns Using Machine Learning project, developed during the Spring 2026 Capstone II course at Florida International University. The project addresses the challenge of forecasting stock closing prices, a task complicated by market volatility and non-linear temporal dependencies that traditional statistical models struggle to capture. Our team developed a machine learning pipeline centered on a Long Short-Term Memory (LSTM) neural network to predict next-day closing prices for equities such as AAPL and NVDA. The pipeline encompasses data collection via Yahoo Finance, feature engineering with technical indicators (moving averages, RSI, MACD, and volatility), data preprocessing with MinMax scaling and sequence generation, and model training with an optimized two-layer LSTM architecture (128 to 64 units). We evaluated model performance through backtesting on unseen historical data using RMSE, MAE, and R-squared metrics, and compared results against a linear regression baseline. Key findings show that the LSTM model captures broader price trends and achieves approximately five percent average prediction error, while directional prediction accuracy remains near random at roughly 52 percent. This document details the problem definition, system architecture, implementation decisions, testing and evaluation results, lessons learned, and directions for future work including multi-day prediction horizons and sentiment analysis integration.

Table of Contents

INTRODUCTION	5
CURRENT SYSTEM	5
PURPOSE OF NEW SYSTEM	6
USER STORIES	
IMPLEMENTED USER STORIES	8
PENDING USER STORIES	9
PROJECT PLAN	
HARDWARE AND SOFTWARE RESOURCES	10
SPRINTS PLAN	11
<i>Sprint 1</i>	11
<i>Sprint 2</i>	11
<i>Sprint 3</i>	12
<i>Sprint 4</i>	12
<i>Sprint 5</i>	13
<i>Sprint 6</i>	13
<i>Sprint 7</i>	14
SYSTEM DESIGN	
ARCHITECTURAL PATTERNS	14
SYSTEM AND SUBSYSTEM DECOMPOSITION	15
DEPLOYMENT DIAGRAM	17
DESIGN PATTERNS	18
SYSTEM VALIDATION	20
GLOSSARY	27
APPENDIX	31
APPENDIX A - UML DIAGRAMS	31
<i>Static UML Diagrams</i>	31
<i>Dynamic UML Diagrams</i>	32
APPENDIX B - USER INTERFACE DESIGN	34
APPENDIX C - SPRINT REVIEW REPORTS	37
APPENDIX D - USER MANUALS, INSTALLATION/MAINTENANCE DOCUMENT, SHORTCOMINGS/WISHLIST DOCUMENT AND OTHER DOCUMENTS	41

INTRODUCTION

Stock price forecasting is a difficult problem in computational finance. Equity prices reflect a wide range of inputs, including earnings releases, macroeconomic data, geopolitical events, and shifts in investor sentiment, all combined into a single noisy time series. Decades of research have shown that the resulting price movements are non-linear and non-stationary, and only weakly predictable from past prices alone. For a typical retail investor or student researcher, the available options are usually limited to simple statistical models that ignore most of this information, or to proprietary trading platforms that do not expose how their forecasts are produced.

The Predicting Stock Returns Using Machine Learning project was developed during the Spring 2026 Capstone II course at Florida International University to address this gap with an open, reproducible forecasting pipeline. The system pulls historical price data for three technology stocks (Apple, Tesla, and NVIDIA), generates a set of common technical indicators, and trains a two-layer Long Short-Term Memory (LSTM) neural network to predict the next-day closing price. To put the model's output in context, the pipeline also trains a linear regression baseline on the same features, runs a GARCH-calibrated Monte Carlo simulation to produce 30-day probabilistic forecasts, and uses logistic regression to classify the next-day direction of the S&P 500 index. All settings are controlled from a single `config.py` file, and every run produces a structured set of outputs (trained models, metrics, plots, forecasts, and backtests) so that the work can be reproduced or extended without rebuilding the project from scratch.

Current System

There are several existing options for forecasting stock returns, but each has limitations that this project tries to address.

Traditional statistical models such as ARIMA, exponential smoothing, and standalone GARCH are still the textbook approach in most introductory finance and time series courses. These models assume linear relationships between past and future returns and cannot capture longer-range patterns that involve multiple input features. Their forecasts also tend to converge toward the recent mean after a small number of steps, which limits their usefulness for stocks like Tesla and NVIDIA, where volatility shifts can happen quickly.

Consumer trading platforms such as Yahoo Finance, TradingView, and Robinhood show technical indicators (moving averages, RSI, MACD, Bollinger Bands) as charting overlays, but they do not

generate model-driven forecasts. Users have to interpret the indicators on their own, and there is no built-in way to backtest a strategy or generate a price distribution. Paid services that do offer machine learning forecasts, such as Bloomberg Terminal and Refinitiv Eikon, are closed source and expensive, and they typically only return a single predicted value with no confidence range or methodology shared with the user.

Academic research on deep learning for stock prediction is plentiful but often fragmented. Most papers focus on a single model architecture applied to a single ticker, and many do not release their code. When a baseline is included, it is often a simple model that performs comparably to the headline model on next-day prediction, since daily prices are highly autocorrelated. Few papers provide a full pipeline that combines feature engineering, model training, backtesting, uncertainty estimation, and direction classification in one place. As a result, a student or junior analyst who wants to apply machine learning to real market data has no clear starting point: open source tools come without models, academic papers come without working tooling, and commercial products come without source code or comparable baselines.

Purpose of New System

The Predicting Stock Returns Using Machine Learning system is built to fill the gap described above with a single integrated and reproducible pipeline. The project has four main goals.

- 1. A single command runs the full pipeline.** Running python main.py downloads market data from Yahoo Finance, builds the feature set, trains and saves both the LSTM and the linear regression baseline, runs a rolling backtest, generates Monte Carlo forecasts, and trains the S&P 500 direction classifier. All outputs are written to a structured outputs directory and can be reviewed, version controlled, or used by other tools without manual setup.
- 2. Fair comparison against a baseline.** For each ticker, the pipeline trains both an LSTM and a linear regression on the same features and reports RMSE, MAE, MAPE, and R-squared on the same out-of-sample data. The goal is not to show that the LSTM is always better; it is to document where a deep learning model adds value over a simple baseline and where it does not. Rolling backtests are used to check whether the measured accuracy holds up across different periods.
- 3. Forecasts with uncertainty estimates.** A single point prediction is not very useful for risk management, since what matters in practice is the range of likely outcomes. The system feeds the LSTM's predicted return into a GARCH(1,1) volatility model and generates 300 Monte Carlo paths over a 30-day horizon. The result is a fan chart with 5th, 50th, and 95th percentile price bands for each stock, which gives a range of possible outcomes rather than a single number.

4. Broader market direction. Because individual stocks are influenced by the overall direction of the market, the system also trains a logistic regression model on lagged S&P 500 returns and technical indicators to predict whether the index will close higher or lower the next day. The reported accuracy is around 53 percent, which reflects how difficult short-term market direction prediction is and provides a reference point for future work.

The remainder of this document covers the user stories that drove development, the sprint plan, the system architecture and design patterns, the testing and validation results, and the limitations and future work identified during the project.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the Predicting Stock Returns Using Machine Learning project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

- #001:** As a developer, I want to set up a GitHub repository for the team so we can share code, review changes, and track the project's history throughout the semester.
- #002:** As a developer, I want all project settings (tickers, date ranges, model hyperparameters, Monte Carlo settings) consolidated into a single config.py file so the pipeline can be customized without editing the source code.
- #003:** As a user, I want to download historical OHLCV stock data from Yahoo Finance for AAPL, TSLA, and NVDA so the model has consistent and reliable training data.
- #004:** As a user, I want common technical indicators (moving averages, RSI, MACD, Bollinger Bands, and rolling volatility) computed automatically from the raw price data so they can be used as input features for the model.
- #005:** As a developer, I want preprocessing utilities for scaling features and building rolling-window sequences so the data is in the right shape for the LSTM.
- #006:** As a user, I want a two-layer LSTM model that takes a 30-day window of features and predicts the next-day closing price for each ticker.
- #007:** As a user, I want a linear regression baseline trained on the same features as the LSTM so I can compare the two models on equal footing.
- #008:** As a user, I want evaluation metrics (RMSE, MAE, MAPE, R-squared) computed for both models on the same out-of-sample test set so the comparison is fair.
- #009:** As a user, I want a rolling backtest that re-evaluates the model across multiple time windows so I can see how performance holds up across different market periods.
- #010:** As a user, I want a GARCH(1,1) volatility model fit to the LSTM's residuals so the forecasts can use realistic, time-varying volatility instead of a constant value.

- #011:** As a user, I want a Monte Carlo simulation that generates 300 price paths over a 30-day horizon so the forecast shows a range of possible outcomes rather than a single number.
- #012:** As a user, I want a logistic regression classifier that predicts the next-day direction (up or down) of the S&P 500 index from lagged returns and technical indicators.
- #013:** As a user, I want plots automatically generated for each ticker, including Actual vs. Predicted charts, Monte Carlo fan charts, confidence bands, ROC curves, and confusion matrices, so the results can be reviewed visually.
- #014:** As a developer, I want all outputs (trained models, fitted scalers, metrics, plots, forecasts, and backtest results) saved into a structured outputs directory so the results are reproducible and easy to share.
- #015:** As a user, I want a single command (python main.py) that runs the full pipeline end to end so the system can be executed without performing each step manually.
- #016:** As a developer, I want a README and a requirements.txt file so a new user can clone the repository, install the dependencies, and run the project on their own machine.

Pending User Stories

- #017:** As a user, I want the model to forecast multi-day horizons (for example 5-day and 30-day closing prices) directly from the LSTM, instead of relying only on the Monte Carlo simulation for longer horizons.
- #018:** As a user, I want news headline sentiment scores added as input features so the model can react to news and earnings events, not just historical price movements.
- #019:** As a user, I want the prediction pipeline deployed as a web service or interactive dashboard so I can request forecasts and view results without running the code locally.
- #020:** As a developer, I want automated hyperparameter tuning (such as grid search or Bayesian optimization) so the LSTM architecture is chosen based on validation performance rather than picked manually.

PROJECT PLAN

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

- **GitHub**. Source control and team collaboration.
- **Python 3.12**. Programming language used for the entire pipeline.
- **TensorFlow / Keras**. Deep learning framework for building and training the LSTM model.
- **scikit-learn**. Linear regression baseline and the S&P 500 logistic regression classifier.
- **pandas**. Data loading, cleaning, and tabular operations.
- **NumPy**. Numerical arrays and matrix operations.
- **yfinance**. Yahoo Finance API client used to download historical OHLCV data.
- **arch**. GARCH(1,1) volatility model fit to LSTM residuals.
- **matplotlib**. Generation of all plots (Actual vs. Predicted, Monte Carlo fan charts, ROC curves, confusion matrices).
- **joblib**. Saving and loading trained models, fitted scalers, and intermediate artifacts.
- **pip and venv**. Dependency management and isolated Python environments.
- **Visual Studio Code**. Primary code editor for the team.
- **Standard laptops**. Development and training hardware; no dedicated GPU is required, since training completes in under ten minutes on a typical laptop CPU.

Sprints Plan

Sprint 1

January 20 – February 1, 2026 | Project Setup and Research

Objective. Set up the development environment, agree on tooling, and review prior work on stock return prediction.

Tasks:

- Create the shared GitHub repository and define the initial folder structure
- Set up Python 3.12 environments and install the core dependencies
- Review prior work on LSTM-based time series forecasting and on GARCH volatility models
- Decide on target tickers (AAPL, TSLA, NVDA) and the historical date range for training
- Draft the initial backlog of user stories and assign sprint roles

Deliverables:

- GitHub repository with an initial README and folder structure
- requirements.txt covering the core dependencies
- Short research summary on LSTMs, technical indicators, and GARCH
- Initial product backlog and sprint plan

Sprint 2

February 2 – February 14, 2026 | Data Acquisition and Configuration

Objective. Build the data ingestion pipeline and the centralized configuration system.

Tasks:

- Implement data_loader.py to download OHLCV data from Yahoo Finance for each configured ticker
- Implement config.py with all tunable parameters (tickers, date ranges, hyperparameters, Monte Carlo settings)
- Add caching of downloaded data so the pipeline does not hit the API on every run
- Add basic data validation checks for missing values and date alignment
- Set up the outputs directory layout (models, metrics, plots, forecasts, backtests)

Deliverables:

- Working data download for AAPL, TSLA, NVDA, and the S&P 500 index

- config.py covering tickers, dates, model hyperparameters, and Monte Carlo settings
- Local CSV cache of historical prices
- Initial outputs directory scaffold

Sprint 3

February 15 – March 7, 2026 | Feature Engineering and Linear Regression Baseline

Objective. Generate the technical-indicator feature set and train the linear regression baseline.

Tasks:

- Implement features.py with simple/exponential moving averages, RSI, MACD, Bollinger Bands, and rolling volatility
- Implement preprocess.py with MinMax scaling and rolling-window sequence generation
- Implement baselines.py with a linear regression trained on the same features
- Compute initial RMSE, MAE, and R-squared for the baseline on each ticker
- Save fitted scalers alongside the baseline models so predictions can be inverse-transformed

Deliverables:

- Feature engineering module with all listed indicators
- Preprocessing module producing model-ready arrays for the LSTM
- Trained linear regression baseline for each ticker with reported metrics
- Persisted scaler objects for each ticker

Sprint 4

March 8 – March 22, 2026 | LSTM Model Development

Objective. Design, train, and evaluate the LSTM model on each ticker, and compare it against the baseline.

Tasks:

- Implement lstm_model.py with a two-layer (128, 64 units) LSTM architecture and dropout
- Add early stopping and model checkpointing to avoid overfitting
- Train and save LSTM models for AAPL, TSLA, and NVDA
- Generate the first round of Actual vs. Predicted plots for each ticker
- Compare LSTM metrics against the linear regression baseline and discuss the results as a team

Deliverables:

- Trained LSTM models saved for each ticker
- Side-by-side metrics tables for LSTM vs. baseline (RMSE, MAE, MAPE, R-squared)
- First Actual vs. Predicted plots saved to the outputs directory

Sprint 5**March 23 – April 5, 2026 | Backtesting and Evaluation Framework**

Objective. Add a rolling backtest and centralize all evaluation logic into reusable modules.

Tasks:

- Implement backtest.py to run rolling-window predictions across the full test period
- Implement metrics.py and evaluate.py for shared metric computation
- Save per-prediction CSVs for each ticker (Actual_Close, Predicted_Close, Absolute_Error, Percent_Error)
- Generate metrics JSON and CSV summaries for each ticker and model
- Review backtest results and identify periods where the LSTM under- or out-performs

Deliverables:

- Backtest prediction CSVs for AAPL, TSLA, NVDA
- Summary metrics JSON and CSV files for each ticker
- Combined backtest summary covering both models

Sprint 6**April 6 – April 12, 2026 | Monte Carlo Simulation and S&P 500 Direction Model**

Objective. Add probabilistic forecasting through Monte Carlo simulation, and build the S&P 500 direction classifier.

Tasks:

- Implement volatility.py with a GARCH(1,1) model fit to the LSTM residuals
- Implement monte_carlo.py to generate 300 paths over a 30-day horizon for each ticker
- Generate fan-chart and confidence-band plots (5th, 50th, 95th percentiles)
- Build the S&P 500 logistic regression classifier with lagged returns and indicators
- Generate ROC curve, confusion matrix, cumulative accuracy plot, and coefficient plot for the direction model

Deliverables:

- Monte Carlo path CSVs and forecast summary JSONs for each ticker

- Confidence band plots and Monte Carlo fan charts
- Trained S&P 500 direction classifier with metrics, ROC curve, and confusion matrix

Sprint 7

April 13 – April 26, 2026 | Documentation, Polish, and Final Deliverables

Objective. Wrap up the project: documentation, videos, poster, and final repository cleanup.

Tasks:

- Write the project documentation, README, installation guide, and user manual
- Record the introduction and user-guide videos
- Design the final poster (36 by 48 inches, horizontal) and the presentation slides
- Refactor the codebase, remove dead code, and verify reproducibility on a clean machine
- Compile daily scrum minutes and sprint review reports for the appendix

Deliverables:

- Project Documentation (this document)
- Recorded introduction and user-guide videos with YouTube descriptions
- Final poster and presentation slides
- Clean, reproducible source repository with installation guide and user manual

SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

The project follows three well-known architectural patterns from software engineering. Together they keep the codebase modular, easy to read, and easy to extend.

Pipeline (pipe-and-filter) architecture. The system is organized as a fixed sequence of processing stages: data download, feature engineering, preprocessing, model training, prediction,

evaluation, backtesting, and forecasting. Each stage takes the output of the previous one and passes its own output to the next. The `pipeline.py` module enforces this order, so the rest of the code does not have to. This makes it easy to swap one stage out, for example replacing the LSTM with a different model, without affecting any of the others.

Layered architecture. The modules are grouped into a small number of layers, each with a clear responsibility. The configuration layer sits at the top and defines all tunable values. Below that, the data and feature layers prepare inputs. The model layer holds the LSTM, the linear regression baseline, the GARCH(1,1) volatility model, the Monte Carlo simulator, and the S&P 500 direction classifier. The evaluation and visualization layers produce metrics and plots. The orchestration layer at the bottom of the stack ties everything together. Higher layers depend on lower layers, but the dependencies do not run in the other direction.

Configuration-driven design. Every parameter that affects a run, including tickers, date ranges, sequence length, LSTM hyperparameters, Monte Carlo path count, and S&P 500 lag days, is defined in a single Config dataclass in `config.py`. Modules import settings from this object rather than hardcoding values. This keeps every tuning decision in one place and lets the pipeline be reconfigured for new tickers or parameters without editing source files.

System and Subsystem Decomposition

The system is broken down into nine subsystems, each implemented as a small set of related Python modules. The diagram below shows how data flows through the layers, from the configuration object at the top to the outputs directory at the bottom.

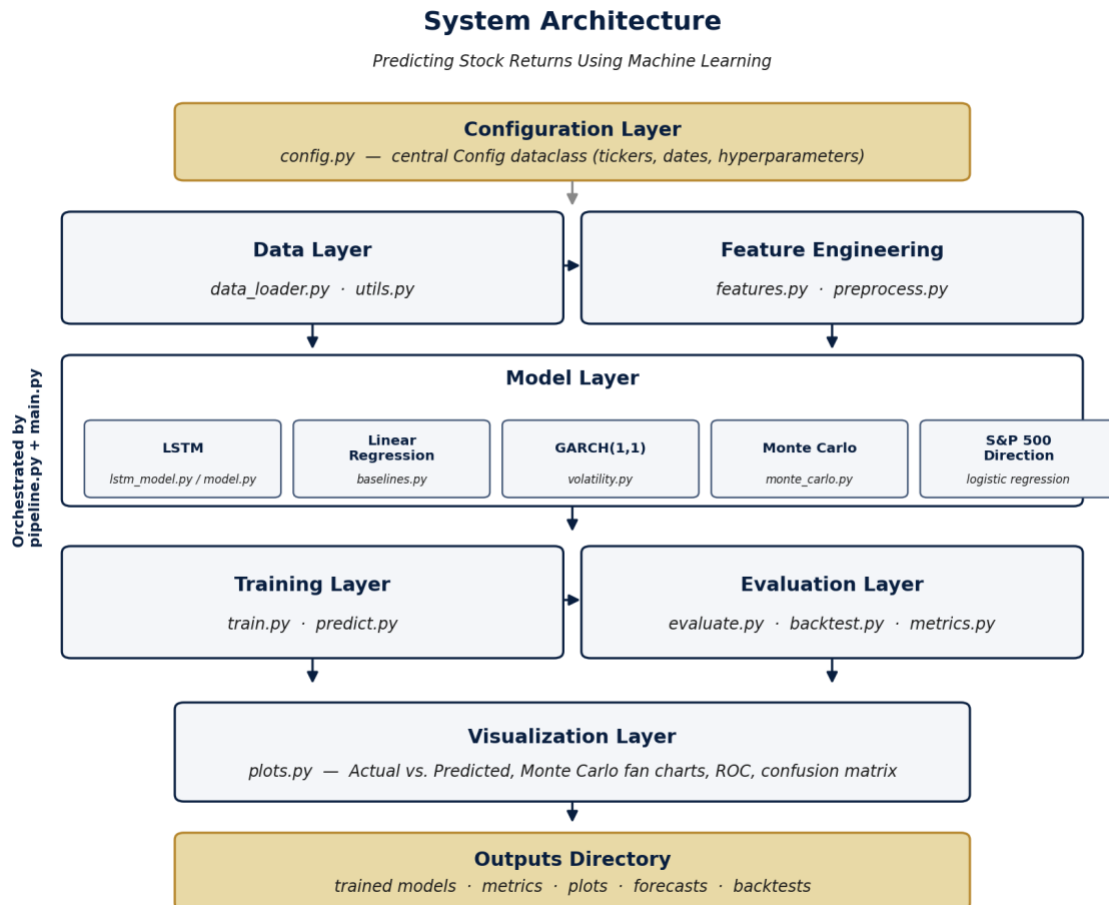


Figure 1. System architecture and subsystem decomposition.

- **Configuration.** Defines all tunable settings (config.py).
- **Data.** Downloads OHLCV data from Yahoo Finance and caches it locally (data_loader.py, utils.py).
- **Feature Engineering.** Computes technical indicators including moving averages, RSI, MACD, Bollinger Bands, and rolling volatility (features.py).
- **Preprocessing.** Applies MinMax scaling and builds rolling-window sequences ready for the LSTM (preprocess.py).

- **Model.** Holds the LSTM, the linear regression baseline, the GARCH(1,1) volatility model, the Monte Carlo simulator, and the S&P 500 logistic regression classifier.
- **Training.** Trains models, generates predictions, and saves trained artifacts (train.py, predict.py).
- **Evaluation.** Computes RMSE, MAE, MAPE, and R-squared, runs the rolling backtest, and writes summary files (evaluate.py, backtest.py, metrics.py).
- **Visualization.** Generates Actual vs. Predicted charts, Monte Carlo fan charts, ROC curves, and confusion matrices (plots.py).
- **Orchestration.** Calls each stage in sequence and writes outputs to disk (pipeline.py, main.py).

Deployment Diagram

The application is a single-machine command-line tool. There is no web server, no client, and no distributed component, which keeps the deployment topology small and easy to reason about.

Local environment. A team member runs the pipeline on a laptop or desktop with Python 3.12, pip, and the dependencies listed in requirements.txt. No GPU is required, since training completes in under ten minutes on a typical CPU.

External services. The only external dependency is Yahoo Finance, reached through the yfinance library over HTTPS. The pipeline sends requests for historical OHLCV data and receives JSON responses; the data is then cached locally so subsequent runs do not need network access.

Local storage. All artifacts live in the outputs/ directory on the user's machine. This includes trained models, fitted scalars, metrics files, plots, forecast paths, and backtest CSVs. The repository's .gitignore excludes this directory from version control.

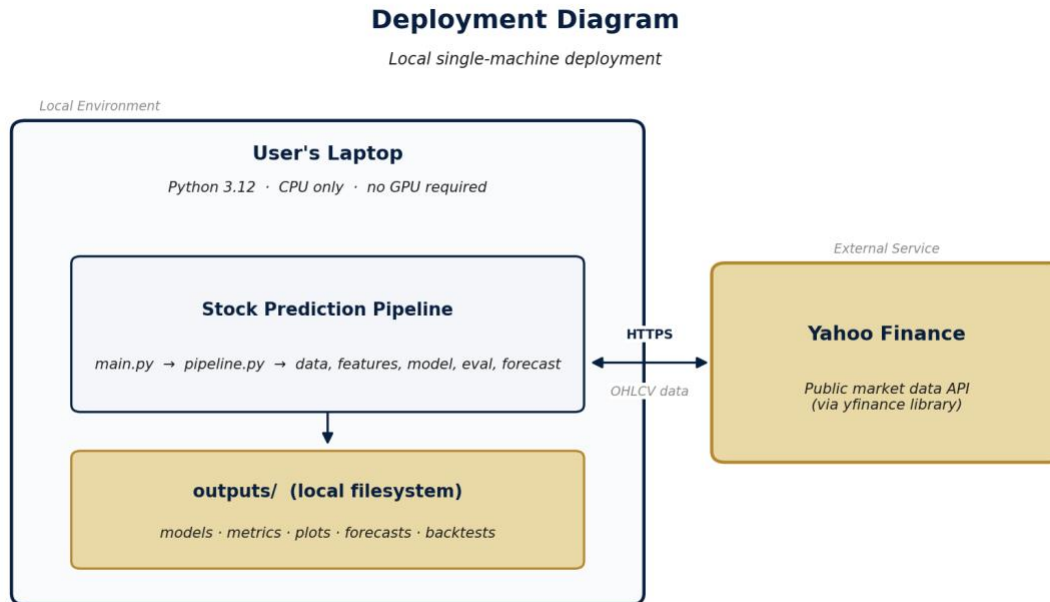


Figure 2. Deployment topology of the prediction pipeline.

Design Patterns

A few common software design patterns appear in the codebase. They are not used heavily, but where they appear they make the code easier to read and easier to extend.

Strategy pattern. The LSTM model and the linear regression baseline both expose the same small interface (fit, predict, save, load). Evaluation code can call either one without knowing which is in use, which is what allows the same backtest and metric functions to score both models on identical data.

Configuration object pattern. A single Config dataclass in config.py holds every tunable parameter. Modules import the configuration when they need it, instead of receiving long argument lists or hardcoding values. This is sometimes called the parameter object pattern.

Template method. The pipeline.py module defines the fixed order in which the stages run (data, features, preprocessing, training, evaluation, forecasting). Each stage delegates to its specialized module, but the surrounding template of stages does not change between tickers. Adding a new ticker means only changing the configuration, not the orchestration.

Persistence. Trained models, fitted scalers, and prediction tables are serialized to disk through joblib and the Keras save format. A re-run does not have to retrain from scratch when only the evaluation or plotting needs to change, and the intermediate artifacts stay available for inspection.

SYSTEM VALIDATION

This section reports how the system was tested and what the test results showed. Validation focused on two questions: how accurately the LSTM model can predict next-day closing prices for AAPL, TSLA, and NVDA compared with a simple linear regression baseline, and how well the logistic regression model can classify the next-day direction of the S&P 500 index. Both questions were answered using out-of-sample data that the models had not seen during training.

Backtesting Methodology

Each ticker's historical OHLCV data was split into a training set and a held-out test set, with the test set always covering the most recent dates so the evaluation respects the time order of the data. A 30-day rolling window was used as the input sequence for the LSTM, and the same engineered feature set was given to the linear regression baseline. Both models were trained, predicted, and evaluated on the same test window so the comparison is fair.

All three tickers produced 349 out-of-sample predictions. For each prediction the system records the actual closing price, the predicted closing price, the absolute error, and the percent error. From these per-prediction records the system computes RMSE, MAE, MAPE, and R-squared, as well as a directional accuracy that measures how often the predicted next-day move agrees in sign with the actual move.

LSTM vs. Linear Regression Baseline

Table 1 reports the four error metrics for the LSTM and the linear regression baseline on each ticker. Lower is better for RMSE, MAE, and MAPE; higher is better for R-squared.

Ticker	Model	RMSE	MAE	MAPE (%)	R ²
AAPL	LSTM	15.03	12.33	5.13	0.578
AAPL	Linear Reg.	5.03	3.70	1.63	0.953
NVDA	LSTM	19.65	16.52	10.52	0.503
NVDA	Linear Reg.	5.87	4.46	3.20	0.956
TSLA	LSTM	27.23	21.89	6.19	0.881
TSLA	Linear Reg.	14.87	11.34	3.46	0.965

Table 1. LSTM and linear regression metrics on the held-out test set (349 predictions per ticker).

On all three tickers and on every metric, the linear regression baseline outperforms the LSTM. The gap is largest on the technology stocks with the strongest short-term autocorrelation, AAPL and NVDA, where the baseline reaches an R-squared above 0.95 while the LSTM stays around 0.50. On TSLA, where intraday volatility is much higher, both models do better in relative terms but the baseline still has lower error. This result is consistent with the broader literature on next-day equity prediction: when today's closing price is highly correlated with tomorrow's, a model that essentially predicts "close to today's price" is hard to beat.

Visual Comparison

Figure 3 plots the actual closing price for AAPL against both the LSTM prediction and the linear regression baseline across the 349-day test window. The baseline tracks the actual price closely, while the LSTM lags and undershoots during the rising periods on the right side of the chart. The shape of the disagreement is similar for NVDA and TSLA.



Figure 3. AAPL actual closing price compared with the LSTM and linear regression predictions on the test set.

Directional Accuracy

Beyond price-level error, the system also measures whether the predicted next-day move agrees in direction with the actual move. Table 2 reports the directional accuracy of the LSTM on each ticker.

Ticker	Directional Accuracy	Number of Test Predictions
AAPL	52.0%	349
NVDA	48.3%	349
TSLA	49.4%	349

Table 2. LSTM directional accuracy on the held-out test set.

All three values sit close to 50 percent, which means the LSTM does not have a useful edge in predicting the sign of next-day moves. This confirms a well-known result in financial machine learning: a low MAPE on price levels does not automatically translate into useful directional signals, because most of the predictability comes from the slow-moving level of the price rather than from the small daily changes around it.

30-Day Probabilistic Forecast

After the LSTM produces a next-day return estimate, that estimate is fed into a GARCH(1,1) volatility model fit to the residuals, and the system simulates 300 Monte Carlo paths over a 30-day horizon. From these paths it builds a 90 percent confidence band (5th to 95th percentile) and a median forecast. Figure 4 shows the resulting confidence interval for AAPL.

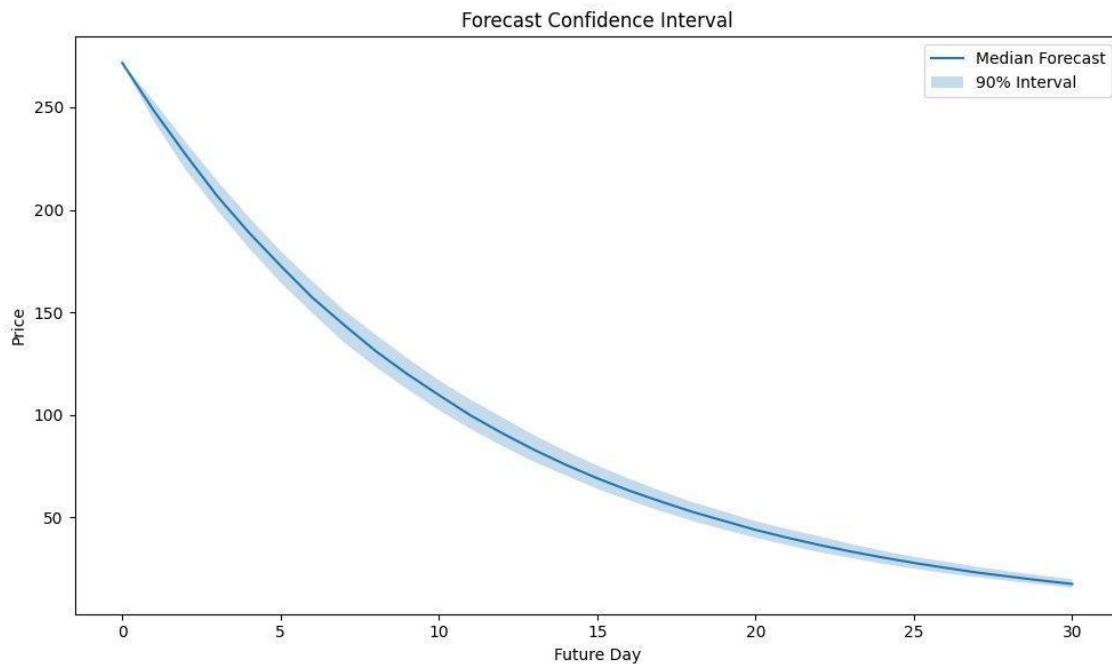


Figure 4. AAPL 30-day Monte Carlo forecast: median path and 90 percent confidence band.

The Monte Carlo simulation correctly produces a widening band over time, but the shape of the median path reveals a limitation of the current setup. Because the LSTM's first-step return estimate is recursively re-applied as the simulation moves forward, any bias in that estimate compounds across the 30-day horizon. In the AAPL case the LSTM predicts a sharply negative next-day return, and that negative drift dominates the volatility term in the simulation, producing a steeply declining median path. This is an honest finding: the probabilistic framework around the model works as intended, but the LSTM itself is not yet calibrated well enough for long-horizon recursive forecasting. The same limitation appears, with different magnitudes, on NVDA and TSLA.

S&P 500 Direction Classifier

The third validation target is the S&P 500 next-day direction classifier. This model is a logistic regression trained on lagged index returns and the same family of technical indicators used by the price-prediction models. The test set covers 1,053 trading days. Table 3 reports the standard classification metrics.

Metric	Value
--------	-------

Accuracy	53.0%
Precision (Up class)	53.5%
Recall (Up class)	89.5%
F1 score	0.669
ROC AUC	0.504
Naive baseline (always predict Up)	53.2%
Number of test predictions	1,053

Table 3. S&P 500 next-day direction classifier metrics on the test set.

The headline accuracy is 53.0 percent, which on its own looks slightly above random. However, the test set itself contains 53.2 percent up-days, so a naive model that always predicts "Up" would score 53.2 percent and slightly outperform the trained model. The confusion matrix in Figure 5 shows what the trained model is actually doing: it predicts "Up" on 937 of the 1,053 test days, which gives it a high recall on up-days (89.5 percent) but very poor recall on down-days. The model is not discovering an edge over the base rate; it is essentially defaulting to the majority class.

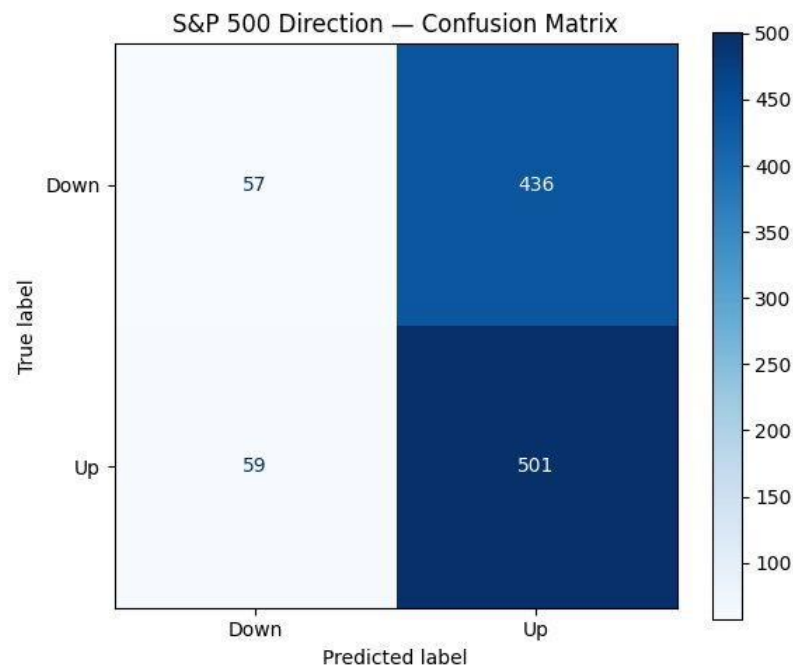


Figure 5. Confusion matrix for the S&P 500 direction classifier.

The ROC curve in Figure 6 confirms the same conclusion. The curve sits almost on top of the random diagonal, and the area under the curve is 0.504, which is indistinguishable from random.

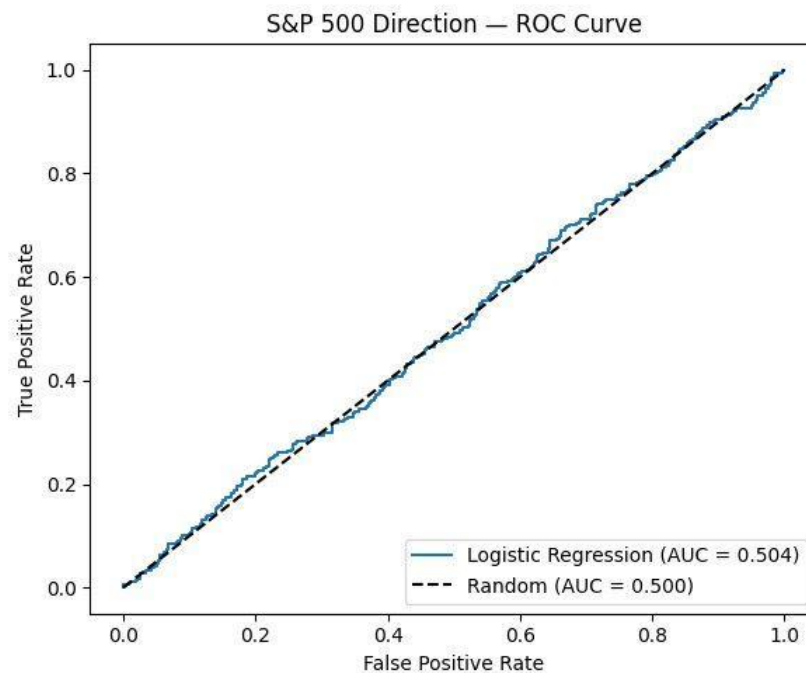


Figure 6. ROC curve for the S&P 500 direction classifier ($AUC = 0.504$).

Discussion

Three findings stand out from the validation results. First, the linear regression baseline outperforms the LSTM on every ticker and every error metric. This was not the team's expectation going into the project, and it was the most useful thing the validation pipeline produced: without an honest baseline, the LSTM's MAPE numbers in the 5 to 10 percent range would have looked respectable in isolation.

Second, directional accuracy hovers near 50 percent across all tickers, which means low absolute price error does not translate into a useful signal for trading or hedging. The system's best results are most useful as a description of the general level of the price, not as a guide to short-term moves.

Third, the Monte Carlo and direction classification components both behave as their underlying methodology predicts they should. The Monte Carlo confidence band widens over the horizon, exactly as a GARCH-driven simulation should; the issue is that the LSTM's recursive return estimate biases the median path. The direction classifier reaches naive-baseline performance, exactly as the efficient-market hypothesis would predict for a model trained only on lagged price-derived features. These results give the team a clear list of priorities for future work, including alternative loss functions for the LSTM, multi-day forecast targets that do not require recursive application, and additional feature sources such as news sentiment.

GLOSSARY

This glossary defines key terms used throughout the document, covering financial concepts, statistical and machine-learning methods, and project-specific terminology. Entries are listed alphabetically.

AUC (Area Under the Curve). A single-number summary of an ROC curve. AUC of 0.5 means the classifier performs no better than random; 1.0 means perfect separation. The S&P 500 direction model in this project has an AUC of 0.504.

Autocorrelation. The correlation of a time series with a lagged copy of itself. Daily stock prices are highly autocorrelated, which is why a model that predicts something close to today's price tends to do well on next-day error metrics.

Backtest. The process of running a trained model on historical data the model did not see during training, to estimate how it would have performed in practice. Each ticker in this project has 349 backtest predictions.

Bollinger Bands. A technical indicator built from a moving average and bands placed a fixed number of standard deviations above and below it. Used as a feature in the price-prediction models.

Confidence Band. A range of forecast values that, with a chosen probability, is expected to contain the actual outcome. The Monte Carlo simulation in this project produces a 90 percent confidence band (5th to 95th percentile).

Confusion Matrix. A table that shows how a classifier's predictions compare with the true labels, broken down by class. Used to diagnose how the S&P 500 direction model distributes its predictions across Up and Down.

Directional Accuracy. The fraction of predictions where the sign of the predicted next-day move matches the sign of the actual next-day move. Sits near 50 percent across tickers in this project.

F1 Score. The harmonic mean of precision and recall. Used as a single summary statistic when both false positives and false negatives matter.

Feature Engineering. The process of constructing input variables for a model from the raw data. In this project, feature engineering converts OHLCV price data into moving averages, RSI, MACD, Bollinger Bands, and rolling volatility.

GARCH (Generalized Autoregressive Conditional Heteroskedasticity). A statistical model for time-varying volatility. The project uses a GARCH(1,1) model fit to the LSTM residuals to drive the Monte Carlo simulation.

Hyperparameter. A model setting chosen before training rather than learned from data. Examples include the LSTM's number of layers, the number of units per layer, the sequence length, the number of training epochs, and the Monte Carlo path count.

Keras. A high-level deep learning API that runs on top of TensorFlow. The LSTM model in this project is built with Keras layers.

Linear Regression. A statistical model that fits a linear relationship between input features and a target value. Used in this project as the baseline against which the LSTM is compared.

Logistic Regression. A model that estimates the probability of a binary outcome. Used in this project to predict the next-day direction of the S&P 500 index.

LSTM (Long Short-Term Memory). A type of recurrent neural network designed for sequence data. Each LSTM cell maintains an internal state that lets the network capture longer-range dependencies than a plain recurrent network.

MACD (Moving Average Convergence Divergence). A technical indicator computed from the difference between two exponential moving averages of a price series, used to identify shifts in trend.

MAE (Mean Absolute Error). The average of the absolute differences between predicted and actual values, reported in the same units as the target.

MAPE (Mean Absolute Percentage Error). The average of the absolute percentage differences between predicted and actual values, reported as a percentage.

Monte Carlo Simulation. A method for estimating uncertainty by generating many random sample paths from a probabilistic model and summarizing the resulting distribution. The project generates 300 paths over a 30-day horizon for each ticker.

Moving Average (MA). The mean of a price series over a rolling window of a given length. Simple moving averages and exponential moving averages are both used as features.

Naive Baseline. A trivially simple comparison model. The S&P 500 direction baseline predicts Up every day, which scores 53.2 percent accuracy on the test set.

OHLCV. Open, High, Low, Close, and Volume. The standard set of daily fields for a stock price series, and the format in which Yahoo Finance returns historical data.

Out-of-Sample. Data that was not used during model training. Reporting metrics on out-of-sample data is necessary for an honest evaluation, since in-sample metrics tend to overstate performance.

Overfitting. When a model memorizes patterns in the training data that do not generalize to new data, leading to poor out-of-sample performance. Mitigated in this project through early stopping and dropout.

Pipeline. The full sequence of stages that take raw data to final outputs. In this project the pipeline runs data download, feature engineering, preprocessing, model training, evaluation, backtesting, and forecasting.

Precision. Of the cases where a classifier predicts the positive class, the fraction that are actually positive.

R-squared (R^2). The fraction of the variance in the target variable explained by the model. R^2 of 1.0 is perfect; 0.0 means the model does no better than predicting the mean.

Recall. Of the cases where the actual class is positive, the fraction the classifier correctly identifies as positive.

RMSE (Root Mean Squared Error). The square root of the mean squared difference between predicted and actual values. Like MAE, reported in the same units as the target. RMSE penalizes large errors more heavily than MAE.

ROC Curve (Receiver Operating Characteristic). A plot of true positive rate against false positive rate as the classification threshold varies. The diagonal from (0,0) to (1,1) corresponds to random performance.

RSI (Relative Strength Index). A technical indicator that measures the speed and magnitude of recent price changes on a 0 to 100 scale. Values above 70 are conventionally treated as overbought and below 30 as oversold.

S&P 500. A market-capitalization-weighted index of 500 large publicly traded U.S. companies. The project includes a logistic regression model that predicts the next-day direction of this index.

scikit-learn. A Python library for classical machine learning. Used in this project for the linear regression baseline, the logistic regression direction classifier, and standard preprocessing utilities such as MinMax scaling.

Sequence Length. The number of past time steps fed into the LSTM as input for a single prediction. The project uses a sequence length of 30 trading days.

TensorFlow. An open-source deep learning framework developed by Google. The LSTM model is built and trained on top of TensorFlow through the Keras API.

Ticker Symbol. The short alphabetic code that identifies a publicly traded security. The project covers AAPL (Apple), TSLA (Tesla), and NVDA (NVIDIA).

Train / Test Split. The division of historical data into a training set, used to fit the model, and a test set, used to evaluate it. The split in this project respects the time order of the data so the test set always covers later dates than the training set.

Volatility. A measure of the variability of returns over time. The project estimates daily volatility from past returns and models its dynamics with GARCH(1,1) for the Monte Carlo simulation.

yfinance. A Python library that provides a convenient interface to Yahoo Finance. Used in this project to download historical OHLCV data for the configured tickers.

APPENDIX

Appendix A - UML Diagrams

This appendix contains the UML diagrams that describe the structure and behavior of the system. The static view (a class diagram) shows the main classes and their relationships. The dynamic view (a sequence diagram) shows the interactions between the components during a single pipeline run.

Static UML Diagrams

Figure 7 is the class diagram. It shows the main classes that drive the system, organized into three tiers. The Config dataclass at the top provides every tunable parameter and is consumed by the Pipeline class. The Pipeline owns the four model classes: LSTMModel for sequence-based price prediction, LinearBaseline for the linear regression comparison, DirectionClassifier for the S&P 500 logistic regression model, and Forecast for the GARCH plus Monte Carlo probabilistic forecast. The residuals from the LSTM are passed into the GARCH component of Forecast, which is shown as a gold data-flow arrow. Helper classes such as DataLoader, FeatureEngineer, Preprocessor, and Backtester are not repeated in this diagram because they are already described in the System and Subsystem Decomposition section (Figure 1).

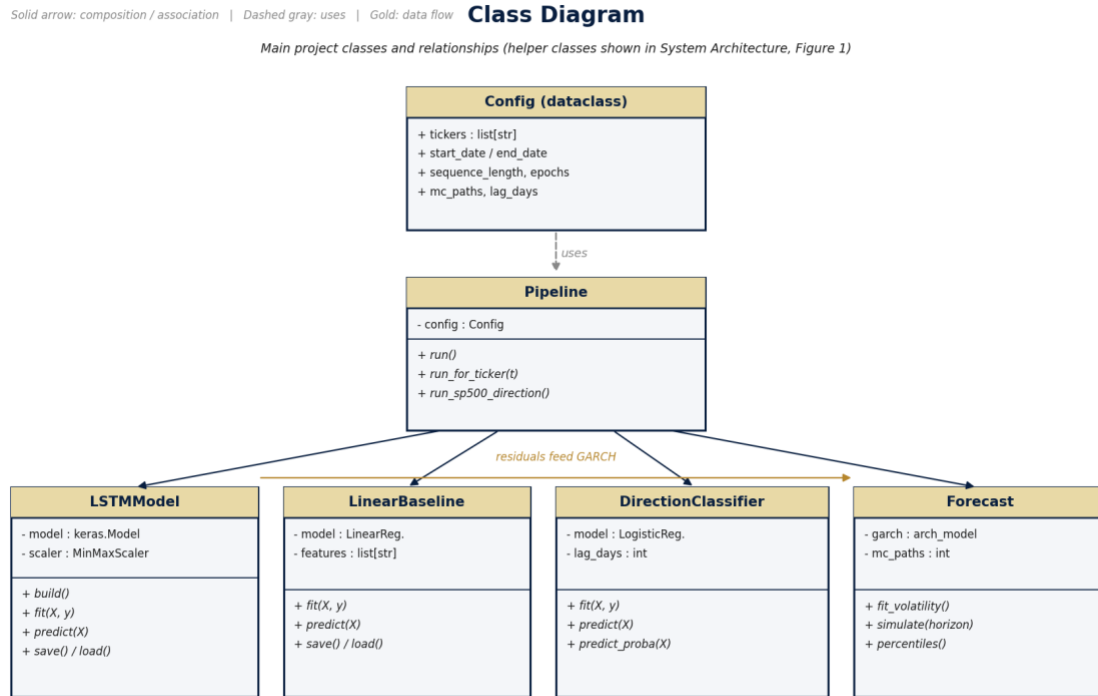


Figure 7. Class diagram showing the main classes and their relationships.

Dynamic UML Diagrams

Figure 8 is the sequence diagram. It shows what happens, in order, when a user runs python main.py for a single ticker. The user starts the program from the command line; main.py calls Pipeline.run(); the pipeline downloads OHLCV data from Yahoo Finance, computes technical indicators, scales the data and builds rolling-window sequences, trains the LSTM and the linear regression baseline, runs the backtest, fits the GARCH model on the LSTM residuals, simulates 300 Monte Carlo paths over a 30-day horizon, and writes plots, metrics, and forecast files to the outputs directory. Solid arrows are synchronous calls; dashed arrows are returns. The same flow runs once per configured ticker.

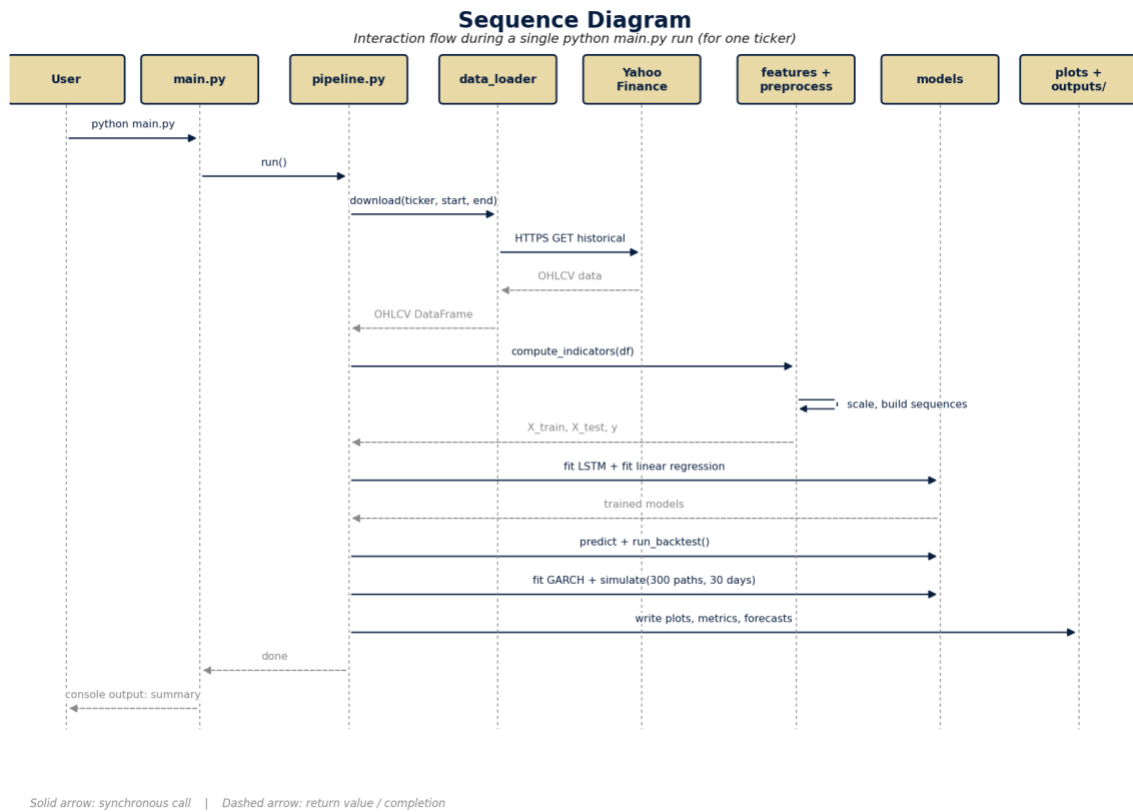


Figure 8. Sequence diagram for a single python main.py run.

Appendix B - User Interface Design

This appendix describes the user interface for the system. The system is a command-line application written in Python, so the user-facing surface consists of three things: a configuration file that the user edits, a command-line entry point that the user runs, and a structured outputs directory where the results are written. There is no graphical user interface in the current version of the project.

Configuration Interface

All tunable parameters live in a single Config dataclass in config.py. The user edits this file directly to choose tickers, date ranges, model hyperparameters, and forecast settings. A simplified view of the configuration is shown below.

```
# config.py
from dataclasses import dataclass, field

@dataclass
class Config:
    tickers: list = field(default_factory=lambda: ["AAPL", "NVDA", "TSLA"])
    start_date: str = "2019-01-01"
    end_date: str = "2026-01-01"

    sequence_length: int = 30 # LSTM input window in trading days
    epochs: int = 20 # max LSTM training epochs
    batch_size: int = 32

    mc_paths: int = 300 # Monte Carlo simulation paths
    forecast_days: int = 30 # forecast horizon in trading days

    sp500_lag_days: int = 5 # lagged returns for direction model
```

Command-Line Interface

After editing the configuration, the user runs the full pipeline with one command from the project root:

```
python main.py
```

The program prints progress to the console as it advances through each stage. A representative run produces output similar to the following.

```
[1/6] Loading configuration from config.py
[2/6] Downloading data: AAPL, NVDA, TSLA, ^GSPC
      AAPL: 1,762 rows (2019-01-02 -> 2025-12-31)
```

```

NVDA: 1,762 rows
TSLA: 1,762 rows
[3/6] Engineering features: SMA, EMA, RSI, MACD, Bollinger, vol
[4/6] Training models for each ticker
AAPL LSTM (epoch 12/20, val_loss=0.0021) done
AAPL LinearRegression done
...
[5/6] Running backtest and computing metrics
AAPL LSTM RMSE=15.03 MAE=12.33 MAPE=5.13% R2=0.578
AAPL LinReg RMSE=5.03 MAE=3.70 MAPE=1.63% R2=0.953
NVDA LSTM RMSE=19.65 ...
...
[6/6] Generating forecasts and plots
AAPL Monte Carlo: 300 paths x 30 days done
Plots saved to outputs/plots/
Pipeline complete. See outputs/ for results.

```

Output Files

Every run writes its artifacts to a structured `outputs/` directory at the project root. The layout is shown below. Files are organized by artifact type, so a user looking for a specific plot or metric can find it without scanning the entire directory.

```

outputs/
|
|-- models/
|   |-- AAPL_lstm.keras
|   |-- AAPL_linear.joblib
|   |-- NVDA_lstm.keras
|   |-- ...
|
|-- metrics/
|   |-- AAPL_metrics.csv
|   |-- NVDA_metrics.csv
|   |-- TSLA_metrics.csv
|   |-- SP500_direction_metrics.csv
|
|-- backtests/
|   |-- AAPL_backtest_predictions.csv
|   |-- AAPL_backtest_summary.json
|   |-- NVDA_backtest_predictions.csv
|   |-- ...
|
|-- forecasts/
|   |-- AAPL_monte_carlo_paths.csv
|   |-- AAPL_forecast_summary.json
|   |-- ...
|
|-- plots/
|   |-- AAPL_actual_vs_predicted.png

```

```
|-- AAPL_monte_carlo.png  
|-- AAPL_confidence_band.png  
|-- SP500_direction_confusion_matrix.png  
|-- SP500_direction_roc_curve.png  
|-- ...
```

Adding a new ticker requires only adding the symbol to the tickers list in config.py and rerunning the pipeline. Adjusting the LSTM training time, the number of Monte Carlo paths, or the forecast horizon similarly requires only editing the relevant Config field.

Appendix C - Sprint Review Reports

This appendix presents the sprint review reports for each of the seven two-week (and in two cases, one and three week) sprints in the development cycle. Each review summarizes the goals the team set at the start of the sprint, the outcomes achieved by the end, and the most important things the team learned during the sprint.

Sprint 1 Review

Dates: January 20 – February 1, 2026

Theme: Project Setup and Research

Sprint goals. Set up the development environment, agree on tooling, and review prior work on stock return prediction.

Outcomes. The team created the shared GitHub repository, set up Python 3.12 environments with the core dependencies installed, and decided on AAPL, TSLA, and NVDA as the target tickers. Initial research notes were collected on LSTMs for time series, the standard set of technical indicators, and on GARCH volatility models. The product backlog was drafted and roles were assigned.

Learnings and observations. The team learned that the project scope should focus on a small number of well-known tickers rather than trying to cover a broad market index, because training and evaluation per ticker take real wall-clock time. The team also agreed early to commit to a comparison against a simple baseline, which turned out to be one of the most valuable decisions made in the entire project.

Sprint 2 Review

Dates: February 2 – February 14, 2026

Theme: Data Acquisition and Configuration

Sprint goals. Build the data ingestion pipeline and the centralized configuration system.

Outcomes. Implemented `data_loader.py` to download OHLCV data through `yfinance`, with local caching so subsequent runs do not hit the API. Implemented the `Config` dataclass with every tunable parameter in one place. Built the initial outputs directory scaffold and basic data-validation checks.

Learnings and observations. Yahoo Finance occasionally returns missing rows for partial trading days, and the team wrote validation utilities to flag these. Centralizing the configuration in `config.py` paid off immediately: every later sprint touched the file but no other module needed to be edited to change tickers, dates, or hyperparameters.

Sprint 3 Review

Dates: February 15 – March 7, 2026

Theme: Feature Engineering and Linear Regression Baseline

Sprint goals. Generate the technical-indicator feature set and train the linear regression baseline.

Outcomes. Built `features.py` with simple and exponential moving averages, RSI, MACD, Bollinger Bands, and rolling volatility. Built `preprocess.py` with MinMax scaling and rolling-window sequence generation. Trained the linear regression baseline for each ticker and recorded the first metrics.

Learnings and observations. This was the longest sprint of the semester at three weeks, which absorbed the Spring Break gap. The linear regression baseline was much stronger than the team expected, with R-squared values above 0.95 on AAPL and NVDA. This set a high bar for the LSTM that ultimately the LSTM did not clear, but it made the eventual evaluation honest and useful.

Sprint 4 Review

Dates: March 8 – March 22, 2026

Theme: LSTM Model Development

Sprint goals. Design, train, and evaluate the LSTM model on each ticker.

Outcomes. Implemented `lstm_model.py` with a two-layer (128, 64 units) LSTM architecture with dropout, early stopping, and model checkpointing. Trained and saved models for AAPL, TSLA, and NVDA. Generated the first round of Actual vs. Predicted plots.

Learnings and observations. Training on the original closing prices produced unstable LSTMs; switching to MinMax-scaled inputs was important. The team also discovered that the LSTM consistently lagged the actual price during rallies, which is visible in Figure 3. This was a real result, not a bug, and led the team to keep the linear regression baseline in the comparison rather than discarding it.

Sprint 5 Review

Dates: March 23 – April 5, 2026

Theme: Backtesting and Evaluation Framework

Sprint goals. Add a rolling backtest and centralize all evaluation logic.

Outcomes. Implemented `backtest.py` for rolling-window predictions across the full test window. Centralized RMSE, MAE, MAPE, R-squared, and directional accuracy computation in `metrics.py` and `evaluate.py`. Per-prediction CSVs and summary JSON files were saved for each ticker and each model.

Learnings and observations. Once metrics computation was centralized, side-by-side comparison between the LSTM and the linear regression baseline became straightforward, and the team confirmed the baseline outperformed the LSTM on every metric for every ticker. Directional accuracy hovered near 50 percent, which made it clear that low MAPE does not automatically translate to a useful directional signal.

Sprint 6 Review

Dates: April 6 – April 12, 2026

Theme: Monte Carlo Simulation and S&P 500 Direction Model

Sprint goals. Add probabilistic forecasting through Monte Carlo simulation, and build the S&P 500 direction classifier.

Outcomes. Implemented `volatility.py` with a GARCH(1,1) model fit to the LSTM residuals and `monte_carlo.py` to generate 300 paths over a 30-day horizon. Generated fan-chart and confidence-band plots. Built the S&P 500 direction classifier with ROC curve, confusion matrix, and cumulative accuracy plot.

Learnings and observations. This was a one-week sprint, and the schedule was tight. The Monte Carlo machinery worked correctly, but the team observed that any bias in the LSTM's first-step return estimate compounded over 30 days. The S&P 500 direction model converged to a 53 percent accuracy that almost exactly matched the naive Always-Up baseline, which was an important honest finding for the validation section.

Sprint 7 Review

Dates: April 13 – April 26, 2026

Theme: Documentation, Polish, and Final Deliverables

Sprint goals. Wrap up the project: documentation, videos, poster, and final repository cleanup.

Outcomes. Wrote the project documentation (this document), the README, the installation guide, and the user manual. Recorded the introduction and user-guide videos, designed the final poster, and prepared the presentation slides. Refactored the codebase, removed dead code, and verified reproducibility on a clean machine.

Learnings and observations. Producing the documentation forced the team to articulate the validation results as a story rather than as a dump of numbers, which clarified the team's view of where the project succeeded and where it fell short. The team also confirmed that the system reproduces from a clean checkout in under fifteen minutes on a typical laptop.

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

This appendix collects the user-facing documents for the project: the installation guide, the user manual, and a list of known shortcomings and future work items.

Installation Guide

The system is a Python application that runs on Windows, macOS, and Linux. It does not require a GPU; training completes in under ten minutes on a typical laptop CPU.

Prerequisites. Python 3.9 or higher, pip, and an internet connection for the initial data download.

Step 1: Clone the repository

```
git clone https://github.com/ecevallos264/Captone_II_Project.git
cd Capstone_II_Project
```

Step 2: Create a virtual environment (recommended)

```
python -m venv venv
```

Activate the environment.

```
# Windows
venv\Scripts\activate

# macOS / Linux
source venv/bin/activate
```

Step 3: Install dependencies

```
pip install -r requirements.txt
```

If requirements.txt is unavailable, install the core dependencies directly:

```
pip install numpy pandas matplotlib scikit-learn tensorflow yfinance arch joblib
```

Step 4: Verify the installation

```
python main.py
```

A successful run downloads the configured stock data, trains the models, writes metrics and forecasts, and saves plots into the outputs directory. The full run typically takes under fifteen minutes.

Notes. TensorFlow GPU is not supported on native Windows for versions 2.11 or later; the project runs on CPU. Internet access is required only for the first run, since data is cached locally afterward.

User Manual

This manual describes how to use the system after it has been installed.

Overview

The system predicts next-day closing prices for a configurable list of tickers using two models in parallel: a two-layer LSTM neural network and a linear regression baseline. It also runs a 30-day Monte Carlo simulation for each ticker, producing a probabilistic price forecast, and it includes a separate logistic regression model that predicts the next-day direction of the S&P 500 index. All results are saved as CSV, JSON, and PNG files in the outputs directory.

Running the program

From the project root, run:

```
python main.py
```

The program runs the full pipeline once and exits. To rerun with different tickers or hyperparameters, edit config.py and run the command again.

Configuration

The Config dataclass in config.py exposes every tunable parameter. The most commonly edited fields are listed below.

- `tickers`: list of stock symbols to process. Defaults to AAPL, NVDA, TSLA.
- `start_date` / `end_date`: training window in YYYY-MM-DD format.
- `sequence_length`: number of trading days fed to the LSTM (default 30).
- `epochs`: maximum number of LSTM training epochs (default 20). Early stopping cuts training short when validation loss stops improving.
- `mc_paths`: number of Monte Carlo simulation paths (default 300).
- `forecast_days`: forecast horizon in trading days (default 30).

Understanding the results

After a run, the metrics, plots, and forecasts produced for each ticker are saved under outputs/. The most important files are:

- outputs/metrics/<TICKER>_metrics.csv: LSTM and linear regression error metrics (RMSE, MAE, MAPE, R-squared) on the held-out test set.
- outputs/backtests/<TICKER>_backtest_summary.json: directional accuracy and number of test predictions for each ticker.
- outputs/plots/<TICKER>_actual_vs_predicted.png: chart comparing actual prices against LSTM and linear regression predictions on the test window.
- outputs/plots/<TICKER>_confidence_band.png: 30-day Monte Carlo forecast with 90 percent confidence band.
- outputs/plots/SP500_direction_confusion_matrix.png and outputs/plots/SP500_direction_roc_curve.png: classification metrics for the S&P 500 direction model.

Customization

The most common customizations are changing the tickers list (any symbol supported by Yahoo Finance works), adjusting sequence_length and epochs to trade off accuracy against training time, and changing forecast_days to shorten or lengthen the Monte Carlo horizon.

Adding more advanced changes (different LSTM architectures, additional technical indicators, or alternative volatility models) requires editing the relevant module: lstm_model.py, features.py, or volatility.py.

Troubleshooting

If a Python package import fails on first run, install it with `pip install <package>`. If TensorFlow refuses to load on Windows, fall back to CPU only or run the project under WSL2. If yfinance returns no data, verify the ticker symbol and the internet connection.

Shortcomings and Wishlist

This section is an honest list of the system's known limitations and the items the team would prioritize in a follow-up project.

Known shortcomings

- The LSTM does not outperform the linear regression baseline on any of the three tickers. Linear regression has lower RMSE, MAE, and MAPE, and a higher R-squared, on AAPL, NVDA, and TSLA.
- Directional accuracy is near 50 percent across all tickers, which means low absolute price error does not translate into a useful trading signal.

- The 30-day Monte Carlo forecast is dominated by any bias in the LSTM's first-step return estimate, because the estimate is recursively re-applied. When the LSTM predicts a sharply negative next-day return, the median path declines steeply over the horizon.
- The S&P 500 direction model reaches 53.0 percent accuracy, which is essentially identical to the naive 53.2 percent Always-Up baseline. The ROC AUC of 0.504 is statistically indistinguishable from random.
- The system only consumes price-derived features. It cannot react to news, earnings releases, or macroeconomic surprises.
- There is no graphical user interface. Configuration is done by editing a Python file rather than through a form or a settings panel.

Wishlist for future work

- Replace the recursive single-step forecast with a multi-day-target LSTM that predicts the entire 30-day path in one forward pass, removing the bias-compounding problem.
- Try alternative loss functions for the LSTM (Huber loss, quantile loss) so the model is less dominated by large absolute errors.
- Add news-sentiment features from a public news API as additional inputs, particularly for the S&P 500 direction model.
- Add a small web dashboard (Plotly Dash or Streamlit) so a non-technical user can pick tickers, dates, and hyperparameters from a form, and view the resulting plots without opening files.
- Add a continuous-evaluation mode that re-runs the pipeline weekly on freshly downloaded data and tracks model drift over time.
- Cover additional tickers and a broader market index, with a job runner that processes them in parallel.