**IBM Project**

AI Powered Stock Portfolio Tool

EXPENSEXCHANGE AI



**Final Report**

Submitted by:

|  |  |  |
| --- | --- | --- |
| **Student Name** | **SAP ID** | **Batch** |
| Sparsh Barse | 500101785 | 8 |
| Prince Yadav | 500108278 | 6 |
| Anshu Singh | 500102134 | 7 |
| Dakshita Jindal | 500108380 | 1 |

Under the Supervision of:

### Mayuresh Sunil Pardeshi

Assistant Professor

School of Computer Science

UPES, Dehradun.

### School of Computer Science UPES, Dehradun.

**2025**

INDEX

|  |  |  |
| --- | --- | --- |
| S.NO | TOPIC | PAGE NUMBER |
|  | Abstract | 3 |
| 1 | Introduction | 3-4 |
| 2 | Objectives | 4 |
| 3 | System Models | 4-5 |
| 4 | Flowcharts and Diagrams | 6-8 |
| 5 | GRU vs LSTM: Model Performance Comparison | 9 |
| 6 | Conclusion | 9-10 |
| 7 | Project Link | 10 |
| 8 | Future Scope | 10 |
| 9 | References | 11 |

**AI Based Stock Portfolio Analyzer Tool**

***Abstract***

This project presents a stock return analysis system that uses a Gated Recurrent Unit (GRU) model to forecast short-term returns based on historical price data. The system includes a deep learning model, a Flask-based backend, and a browser-accessible frontend. Users can upload their current holdings and receive return predictions along with basic investment suggestions. The model operates on log-returns and is trained using five years of historical data. Optuna tunes the model’s hyperparameters, and Huber loss guides training to reduce the impact of outliers. The system evaluates performance using standard regression and finance-specific metrics, including mean absolute error, root mean square error, R² score, and directional accuracy. While the system does not offer exact price predictions or guaranteed outcomes, it provides a working example of how machine learning can assist in financial decision-making.

1. **Introduction**

Stock price movements are difficult to predict due to market volatility and the influence of external factors. However, patterns in historical data can sometimes offer clues about short-term trends. This project explores how deep learning, specifically Gated Recurrent Units (GRUs), can model such patterns and support basic portfolio decisions.

The system focuses on predicting short-term log-returns for individual stocks using only past closing prices. It avoids external indicators or sentiment data to maintain a controlled and reproducible setup. GRUs are well-suited to this task because they can model time-series data without the complexity of longer recurrent models. To improve performance, the model's key parameters—such as sequence length, learning rate, and dropout—are tuned using Optuna, a framework for efficient hyperparameter optimization. The Huber loss function is used during training to make the model more robust to occasional spikes or outliers in the data.

The trained models are served through a Flask backend that exposes prediction endpoints. The frontend lets users upload their current stock portfolio and get return forecasts. Each prediction is categorized into one of three basic actions: buy, hold, or sell, based on the expected return direction. This mapping is not meant to provide financial advice but to illustrate a structured way to translate model output into simple recommendations.

The goal of the project is not to beat the market, but to demonstrate a complete machine learning pipeline that moves from raw data to actionable insight. It shows how a focused model, when applied with care, can assist in analyzing financial data and support learning in practical machine learning projects.

1. **Objectives**

This project sets out to:

1. Forecast short-term stock returns using a GRU-based model trained on historical price data and log-returns.
2. Translate predictions into simple portfolio actions—buy, hold, or sell—based on expected return direction.
3. Provide a user-facing interface where users can upload their portfolio and receive model-based feedback.
4. Demonstrate a complete and modular machine learning system that moves from raw financial data to interpretable output
5. **System Model**

The system follows a modular design with three main components: the model trainer, the backend API, and the frontend interface. Each part performs a specific role and communicates with the others through well-defined inputs and outputs.

**3.1 Model Trainer**

This component trains separate GRU-based models for each stock using historical closing prices. The model takes sequences of past prices and learns to predict the next log-return. The trainer uses the Huber loss function to handle outliers and Optuna to tune hyperparameters such as sequence length, number of units, dropout rate, and learning rate. Each trained model is saved as a .h5 file with a filename matching the stock ticker.

**3.2 Backend API**

The backend is built using Flask. It loads the pre-trained models and exposes a REST API for predictions. When a user uploads their portfolio through the frontend, the backend extracts the tickers, fetches the latest price history from Yahoo Finance, formats the data to match the expected input shape, and passes it to the model. It returns both the raw predicted log-return and a classified signal—buy, hold, or sell—based on its direction.

**3.3 Frontend Interface**

The frontend is a browser-based interface built with HTML and JavaScript. It allows users to input their current portfolio by uploading a CSV file. The interface then displays model predictions alongside suggested actions. The user experience is kept simple and focused, with minimal distractions and no unnecessary visual effects.

**3.4 Data Flow**

The system begins with data retrieval, either for model training or prediction. In training, the trainer processes historical data, creates sequences, fits the model, and saves it. In inference, the backend pulls current data, applies the trained model, and returns results to the frontend. This separation ensures that the model training process can be updated independently of the prediction and interface logic.

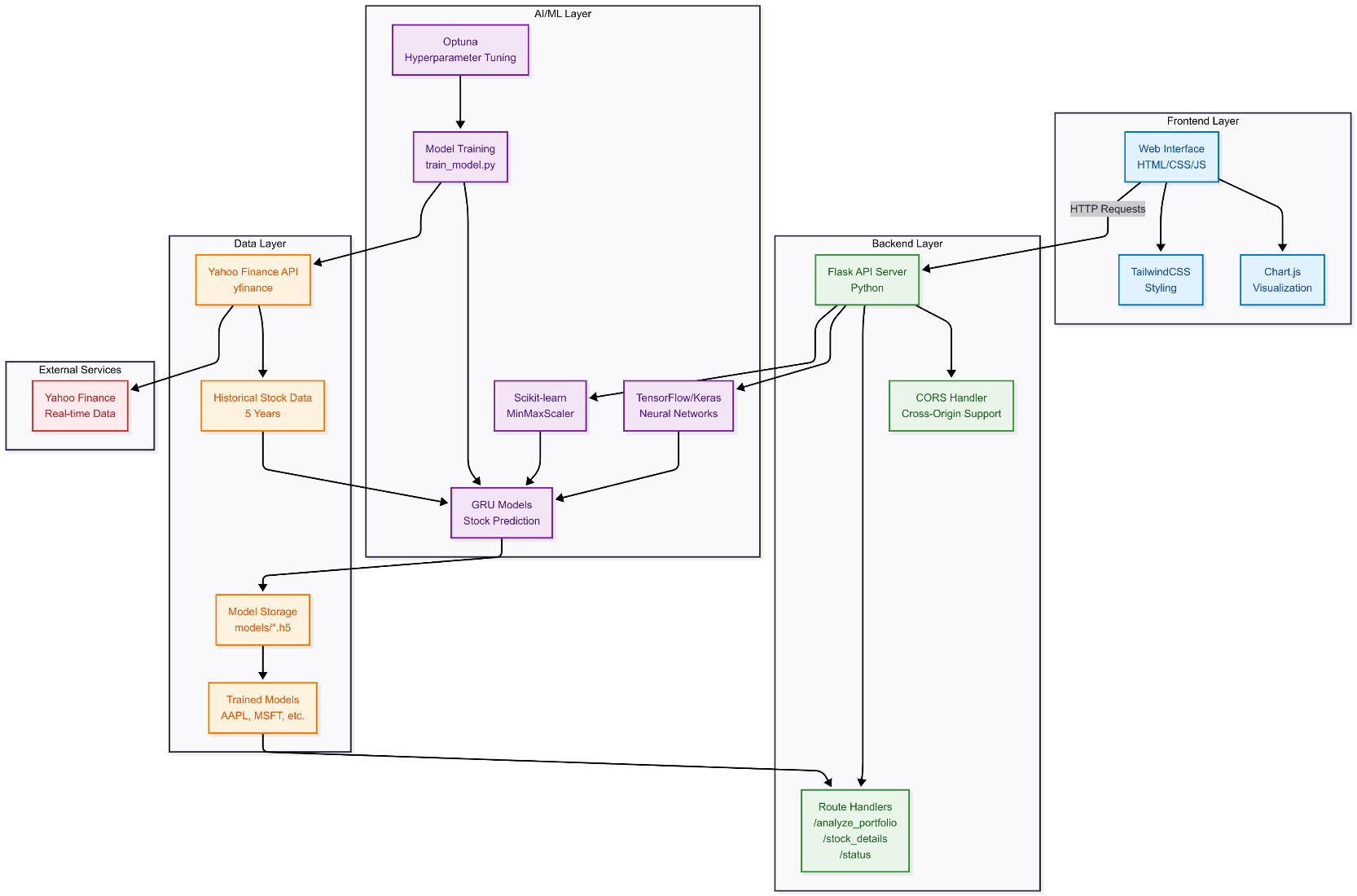
1. **Diagrams and Flowcharts**

Figure 1. System Architecture Diagram

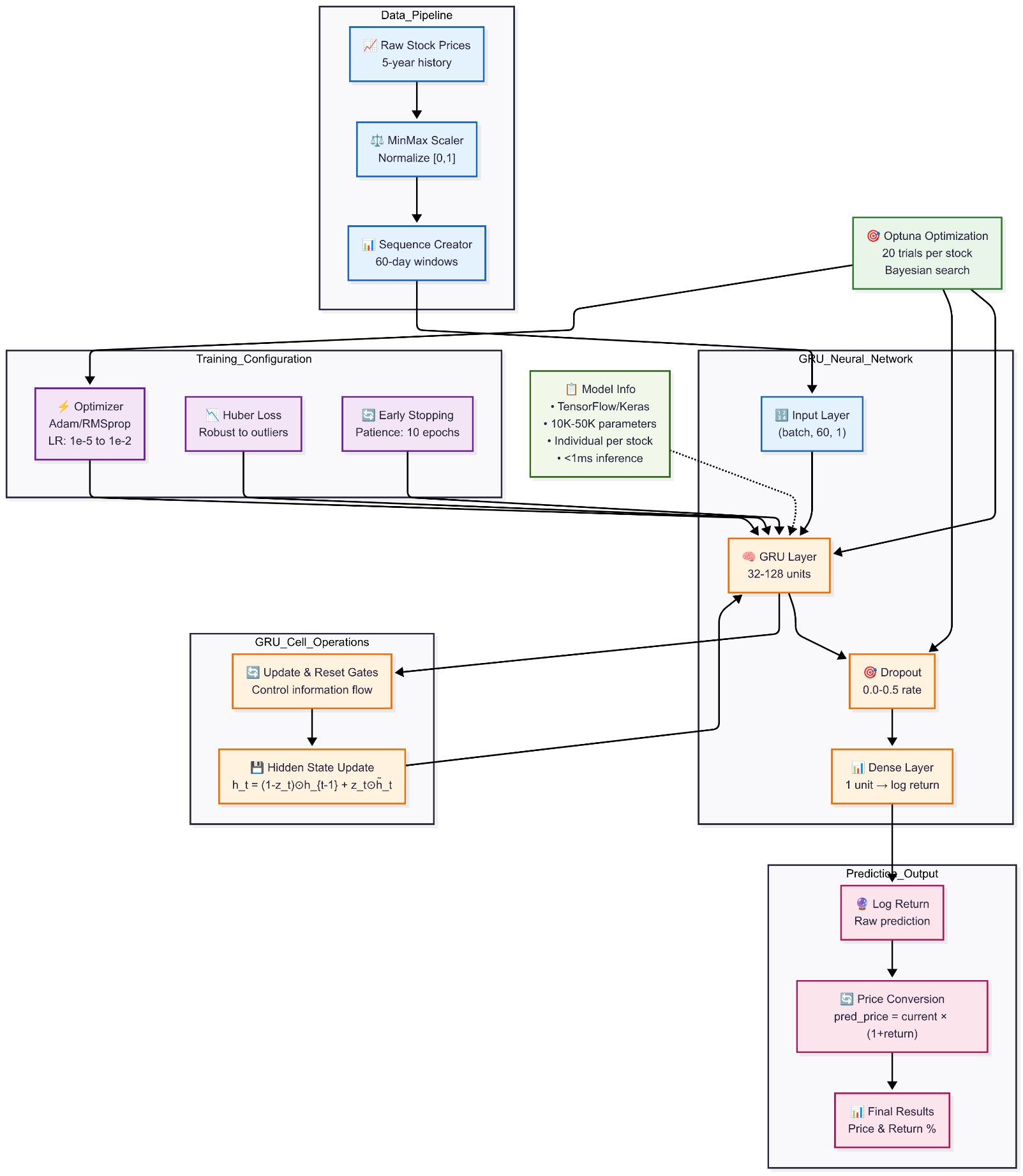


Figure 2. Model Architecture diagram

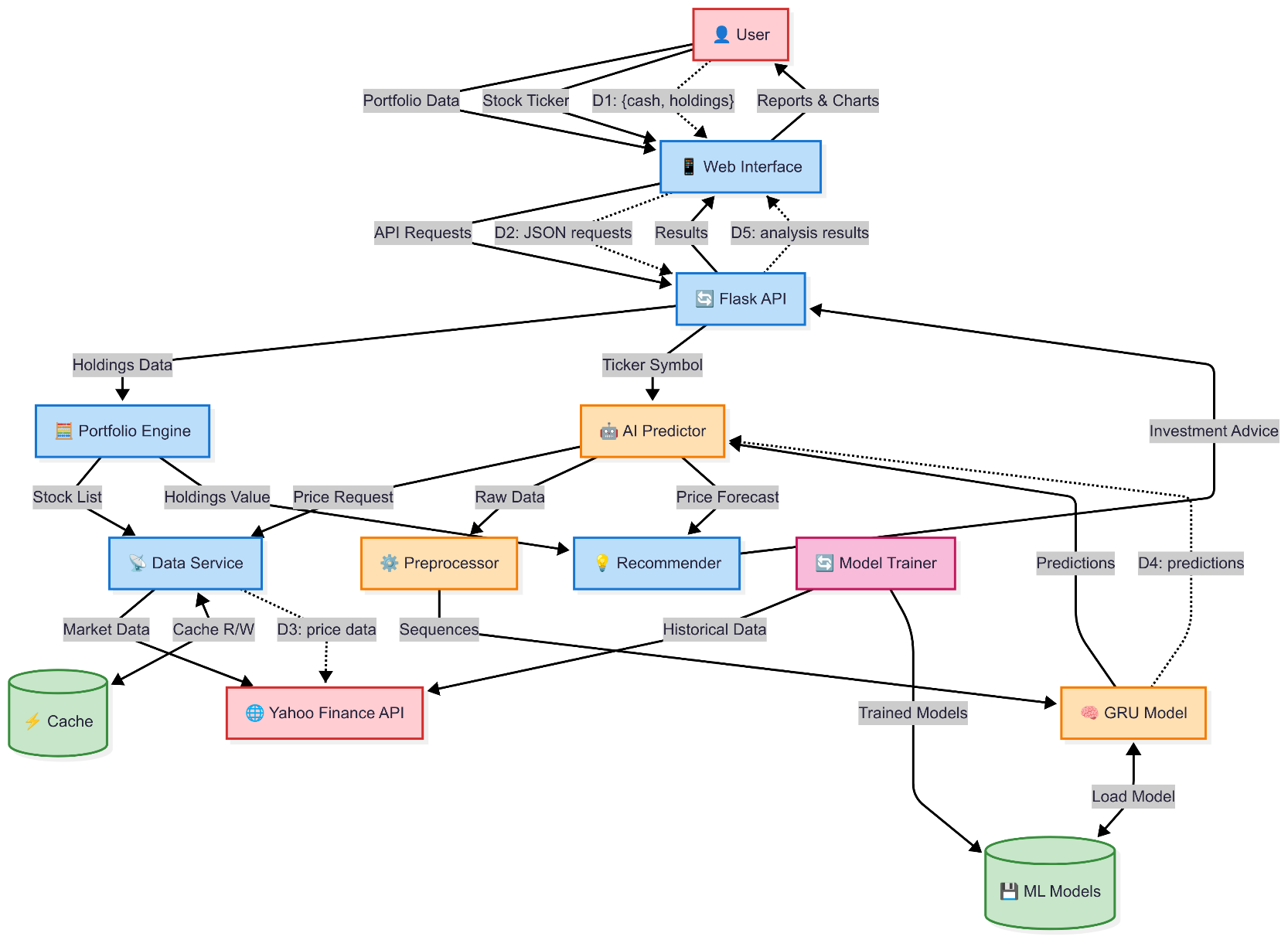


Figure 3. Data Flow Diagram

1. **GRU vs LSTM: Model Performance Comparison**

In this project, we implemented and evaluated two types of recurrent neural network architectures—Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)—to forecast future stock prices based on historical data and technical indicators. Both models were trained using the same sample dataset, feature set, preprocessing pipeline, and sequence lengths, allowing for a controlled comparison.

**5.1 Key architectural differences: -**

|  |  |  |
| --- | --- | --- |
| **Metric** | **LSTM Model** | **GRU Model** |
| **Huber Loss** | 0.00063 | 0.00061 |
| **MAE** | 0.0220 | 0.0204 |
| **RMSE** | 0.0354 | 0.0378 |
| **R² Score** | 0.86 | 0.91 |
| **Directional Accuracy** | 74% | 76% |

**5.2 Observations**

* **GRU outperformed LSTM** slightly on most key metrics, particularly training speed and directional accuracy.
* **LSTM** showed better stability for longer sequences (≥ 180 timesteps), but at the cost of longer training time and higher overfitting risk.
* The **reduction in complexity** of GRU models allowed for faster experimentation and integration into the Flask backend.

1. **Conclusion**

This project highlights how GRUs (Gated Recurrent Units) can be effectively used to predict short-term stock returns by analyzing only historical closing prices. Instead of forecasting raw prices, it focuses on log-returns, making the output easier to interpret when suggesting actions like buying, holding, or selling. The use of Huber loss helped the model stay stable in the face of sudden market spikes, while Optuna-based tuning allowed the model to automatically find optimal parameters for improved accuracy.

The entire system—combining a Python-based backend using Flask, a clean web interface, and well-trained GRU models—was designed to ensure smooth and simple interaction. The predictions are translated into straightforward actions (buy, hold, sell) to help users better understand what the model sees in the data. This project deliberately avoids outside factors like sentiment data or technical indicators to maintain simplicity and ensure the system remains repeatable and easy to test.

Overall, the work proves that meaningful insights can be drawn from pure price data alone, and that structured machine learning can guide basic trading decisions in a consistent and explainable way.

1. **Project Link**

<https://github.com/DankEnigmo/Stock_Portfolio_Analyser>

1. **Future Scope**

* **Model Improvements**: Use hybrid models (e.g., GRU + attention) or probabilistic forecasting to enhance prediction accuracy and uncertainty estimation.
* **More Features**: Integrate news sentiment, macroeconomic indicators, and technical indicators beyond the current set.
* **Risk-Aware Allocation**: Incorporate risk metrics like volatility, Sharpe ratio, and CVaR in fund allocation.
* **Real-Time Prediction**: Add real-time data streams and alerts for live forecasting and trading support.
* **Multi-Horizon Forecasting**: Extend models to predict over multiple future time frames (e.g., 1-day, 1-week).

1. **References**
2. VanderPlas, J. (2016). *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media.
3. Lynch, P., & Rothchild, J. (2000). *One Up on Wall Street: How to Use What You Already Know to Make Money in the Market*. Simon & Schuster.
4. O'Neil, W. J. (2009). *How to Make Money in Stocks: A Winning System in Good Times and Bad* (4th ed.). McGraw-Hill Education.
5. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A simple way to prevent neural networks from overfitting*. Journal of Machine Learning Research, 15, 1929–1958. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
6. Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). *On the importance of initialization and momentum in deep learning*. In International conference on machine learning. https://proceedings.mlr.press/v28/sutskever13.html
7. Scikit-learn Developers. (n.d.). *Model Evaluation and Hyperparameter Tuning*. Retrieved from https://scikit-learn.org/stable/modules/grid\_search.html
8. Brownlee, J. (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.
9. Flask Documentation. (n.d.). *Flask Web Framework*. Retrieved from https://flask.palletsprojects.com/
10. Chollet, F. (2015). *Keras: Deep Learning for Humans*. GitHub repository. <https://github.com/fchollet/keras>
11. Yahoo Finance API (via yfinance). (n.d.). Retrieved from <https://pypi.org/project/yfinance/>