Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Institute of Computer Science

Master's diploma thesis

in the field of study Computer Science
and specialisation Computer Systems and Networks

Development of Artificial Intelligence Support for the Cryptocurrency Market

Mehmet Hanifi ISIK
student record book number 324325

thesis supervisor
Prof. dr hab. inż. Jan Mulawka

WARSAW 2025

Development of Artificial Intelligence Support for the Cryptocurrency Market

Abstract. The difficulty in predicting cryptocurrency prices is due to the instability of the currency market and interconnected sentiment, regulatory, and technological trends. This thesis aims to enhance accuracy in price prediction using deep learning architecture. This thesis intends to increase the likelihood of success in price prediction by implementing various deep learning architectures. These include Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), and hybrid models such as LSTM-CNN and CNN-LSTM, which are chosen for their ability to capture spatial and temporal patterns in data. A complete data pipeline is constructed from data processing and normalization to feature engineering to Optuna for hyperparameter tuning for optimal prediction results. Prediction experiments yield accuracies and performance validated through performance metrics. The LSTM architecture achieves the best prediction performance on the Solana coin dataset, $R^2 = 0.9614$ (MSE = 0.000627, MAE = 0.018803). The CNN-LSTM architecture produces the most consistent prediction performance, $R^2 = 0.9596$ (MSE = 0.000655, MAE = 0.019019); the CNN architecture obtains $R^2 = 0.9582$ (MSE = 0.000678, MAE = 0.019855). The LSTM-CNN architecture produces the most reliable performance on the Ethereum coin dataset, $R^2 = 0.9544$ (MSE = 0.000761, MAE = 0.019163). Thus, the results suggest that the LSTM architecture best captures the dependencies and trends over time regarding fluctuations in cryptocurrency price to yield the highest accuracy when compared to competing architectures. As for model development and deployment in a production environment with Flask backend and WebSocket for real-time updates, performance optimizations via Redis caching, Docker, and CI/CD pipeline ensure deployment, efficacy, and performance.

Keywords: Cryptocurrency Price Prediction, Deep Learning, CNN, LSTM, Hyperparameter Optimization, Web-Based Forecasting, Financial Technology

ROZWÓJ WSPARCIA DLA RYNKU KRYPTOWALUT METODAMI SZTUCZNEJ INTELIGENCJI

Streszczenie. Prognozowanie cen kryptowalut jest trudnym zadaniem ze względu na niestabilność rynku walutowego oraz wzajemne powiązania trendów sentymentalnych, regulacyjnych i technologicznych. Niniejsza praca ma na celu poprawę dokładności prognozowania cen poprzez zastosowanie architektur głębokiego uczenia. W tym celu wdrożono różne modele głębokiego uczenia, w tym konwolucyjne sieci neuronowe (CNN), sieci długiej pamięci krótkoterminowej (LSTM) oraz modele hybrydowe, takie jak LSTM-CNN i CNN-LSTM, które zostały wybrane ze względu na ich zdolność do uchwycenia wzorców przestrzennych i czasowych w danych. Zaprojektowano kompletny pipeline przetwarzania danych, obejmujący przetwarzanie i normalizację danych, inżynierię cech oraz optymalizację hiperparametrów przy użyciu Optuna, w celu uzyskania optymalnych wyników predykcji. Wyniki eksperymentów prognostycznych zostały ocenione za pomocą miar skuteczności modeli. Najlepszą dokładność prognozowania na zbiorze danych Solana osiągnęła architektura LSTM (R^2 = 0.9614, MSE = 0.000627, MAE = 0.018803). Architektura CNN-LSTM wykazała największą stabilność (R^2 = 0.9596, MSE = 0.000655, MAE = 0.019019), natomiast model CNN osiągnął R^2 = 0.9582 (MSE = 0.000678, MAE = 0.019855). Model LSTM-CNN wykazał najwyższą niezawodność na zbiorze danych Ethereum (R^2 = 0.9544, MSE = 0.000761, MAE = 0.019163). Wyniki te sugerują, że architektura LSTM najlepiej uchwyca zależności i trendy czasowe w zakresie fluktuacji cen kryptowalut, osiągając najwyższą dokładność w porównaniu z innymi architekturami. Wdrożenie modeli w środowisku produkcyjnym obejmuje backend oparty na Flask, aktualizacje w czasie rzeczywistym przy użyciu WebSocket oraz optymalizację wydajności za pomocą Redis caching. Ponadto Docker i pipeline CI/CD zapewniają efektywność oraz niezawodne wdrożenie modeli predykcyjnych.

Słowa kluczowe: Prognozowanie cen kryptowalut, Głębokie uczenie, CNN, LSTM, Optymalizacja hiperparametrów, Prognozowanie internetowe, Technologia finansowa

załącznik nr 10 do zarządzenia
nr 46 /2016 Rektora PW

Politechnika Warszawska
Warsaw University of Technology

…….....
miejscowość i data
place and date

……..
imię i nazwisko studenta
name and surname of the student
……..
numer albumu
student record book number
…….…….
kierunek studiów
field of study

OŚWIADCZENIE

DECLARATION

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Under the penalty of perjury, I hereby certify that I wrote my diploma thesis on my own, under the guidance of the thesis supervisor.

Jednocześnie oświadczam, że:
I also declare that:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- *this diploma thesis does not constitute infringement of copyright following the act of 4 February 1994 on copyright and related rights (Journal of Acts of 2006 no. 90, item 631 with further amendments) or personal rights protected under the civil law,*

- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- *the diploma thesis does not contain data or information acquired in an illegal way,*

- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- *the diploma thesis has never been the basis of any other official proceedings leading to the award of diplomas or professional degrees,*

- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- *all information included in the diploma thesis, derived from printed and electronic sources, has been documented with relevant references in the literature section,*

- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.
- *I am aware of the regulations at Warsaw University of Technology on management of copyright and related rights, industrial property rights and commercialisation.*

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.
I certify that the content of the printed version of the diploma thesis, the content of the electronic version of the diploma thesis (on a CD) and the content of the diploma thesis in the Archive of Diploma Theses (APD module) of the USOS system are identical.

.....
czytelny podpis studenta
legible signature of the student

Contents

1. Introduction

1.1. Motivation

The initial step of our process was to explore the nature of cryptocurrency and what it has done to financial markets. At its core, cryptocurrency is a decentralized method of implementing currency without turning to traditional means of cash. With the advent of digital currencies like Bitcoin, Ethereum, and Solana, opportunities for international transactions and a lessened need for intermediary third parties have created more channels for accessing capital that some may not previously have had. Yet for investors and analysts looking to make sense of the cryptocurrency market, this is a problem. Unlike standard, conventional financial assets that have a predictable response to macroeconomic developments and events, the currency of cryptocurrency fluctuates for many reasons [1]. Therefore, price prediction is required to improve the selection process. The price dynamics of cryptocurrency are nonstationary and not driven by a stochastic process, which creates difficulty in forecasting currency fluctuations. For instance, traditional time-series forecasting with ARIMA and GARCH has shown limited success in forecasting; however, these techniques do not account for the nonlinear fluctuations associated with cryptocurrency. Yet, with access to more and more high-frequency trading data and the worldwide interest in artificial intelligence, approaches based on Such an approach exploits the changes in price and other non-financial indicators over time to potentially increase prediction accuracy aside from more traditional methods [2], [3].

1.2. Challenges in Cryptocurrency Price Prediction

However, several challenges hinder the efficacy of deep learning solutions for cryptocurrency price forecasting. While we recognize a few challenges, the most stubborn one is that the trading of the cryptocurrency market is volatile and speculative; prices fluctuate from erratic spurts and with frenzied, reactionary trends [4]. Such features of the currency trading environment present formidable challenges for deep learning systems, which require perfect situational awareness for generalization across fluctuating market environments over time. Furthermore, fluctuating intra-market factors like legal regulations, technological changes, and social sentiments/impacts also contribute to these challenges [5]. through implementation. It's not enough to just implement a model; there are literally hours upon hours of hyperparameter tuning adjusting learning rates, batch sizes, epochs, etc. For example, in the regression models used for predicting financial forecasts, there's an activation function that is important - for open and close, you use the linear function; for up and down, you use the sigmoid function. If you don't, you get bad returns meaning that even if you predict well, the purpose of the prediction fails based upon the error that is not predictive. Therefore, it's not only challenging to justify proper utilization of a model, but an error to utilization that has nothing to do with it can still

ruin the purpose. necessitated dataset-dependent hyperparameter tuning and feature selection strategies [6].

1.3. Contributions

This research intends to establish a definitive model for cryptocurrency prediction by exploring various deep-learning approaches and adjusting them based on a comparative analysis. The deep learning approaches are CNNs, which detect short-term trends based on the features of specific data sets within the market; LSTMs, which detect trends based on long-term data dependencies across larger periods; and a blended CNN-LSTM approach. In addition, hyperparameter adjustment takes place through Bayesian optimization with Optuna to achieve the most fruitful execution and predictions. The predicted models will be executed in the real world via a deployed full-stack web application for cryptocurrency prediction with extensive documentation. api. The Application is a Flask backend that communicates with the Python logic that operates the prediction generator and a separate package for the socket implementation. The application frontend was created with Next.js, and WebSocket is utilized to guarantee that price feeds are sent in real-time. All dependencies both domestically and internationally ensure proper functionality between both sides of the prediction and trading operation. Outcome generation through data visualization using Recharts and TradingView aids traders in comprehending and acting upon model outputs[7]. The original contribution of this research is the extensive analysis of deep learning architectures for cryptocurrency price prediction and the conclusive seamless hyperparameter tuning and extensive evaluation of model performance via monetary values of R-squared, Mean Squared Error, and Mean Absolute Error. Furthermore, the ability to deploy deep learning models in a live web application for a production environment connects AI-driven financial predictions to practical use with applicable information for traders and investors alike.

1.4. Thesis Structure

The subsequent chapters are as follows. Chapter 2 contains the literature review of anticipated cryptocurrency value and a review of deep learning for economic forecasting. Chapter 3 contains the methodology of the experiment from data collection and processing to model development and processing. Chapter 4 contains the backend development from API development to WebSocket integration to execution of predictions. Chapter 5 contains the frontend development for human interaction from UI rendering to live updating and state management. Chapter 6 contains the deployment and implementation from Docker installation to CI/CD pipeline construction to stability testing. Chapter 7 contains the results and assessment of the system and potential future work.

2. Related Work

The issue surrounding time-series forecasting to predict cryptocurrency values is the same as it is with any other. Cryptocurrency market fluctuations are temperamental, volatile, and non-stationary. Yet, similar to the stablecoin research above, many variables exist that influence cryptocurrency pricing—from token supply and demand to legislative concerns about the assets and interest by investors to macroeconomic fluctuations. Yet much of the research below uses deep learning equations to achieve time-stable, reasonable answers to these pricing projection concerns. The highest outcome of the significances within the field is as follows.

Table 2.1. Summary of Key Studies in Cryptocurrency and Stock Market Forecasting

Lp	Study (Authors)	Model	Best Performance
1	[8]	LSTM, RNN, GRU	LSTM outperforms RNN and GRU
2	[3]	Deep Learning with Technical Indicators	Improved accuracy with technical indicators
3	[9]	Deep Learning Models	Enhanced trend prediction
4	[10]	CNN-LSTM	Superior performance in stock price prediction
5	[11]	Deep Learning Methods	Effective stock market price forecasting
6	[12]	Deep Learning with Interpretability Enhancement	Improved model interpretability
7	[13]	Deep Learning with Historical Prices and News	Enhanced prediction accuracy
8	[14]	LSTM Model	High accuracy in stock price prediction
9	[15]	Hybrid Convolutional Recurrent Model	Effective Bitcoin price prediction
10	[16]	Machine Learning Time Series Analysis	Accurate financial trend prediction
11	[17]	CNN-LSTM Hybrid	Enhanced prediction performance
12	[18]	CNN-LSTM vs. LSTM-CNN	Comparative performance analysis
13	[19]	Deep Learning Comparison	Comprehensive model evaluation

Lp	Study (Authors)	Model	Best Performance
14	[20]	CNN-LSTM Models	Effective financial market trend analysis
15	[21]	Deep Learning Models	Comparative study on stock price prediction
16	[22]	Hybrid Deep Learning Models	Improved stock market prediction
17	[23]	Deep Learning Models	Effective crypto price prediction

Many studies have been done with machine learning and deep learning models to predict trading in cryptocurrency and stocks. For instance, [8] analyzes the efficacy of LSTM, RNN, and GRU models to predict stock prices. The models were trained on historical price data, and with the acquired findings, LSTM was assessed as the most effective and accurate model in comparison to RNN and GRU.

Also, [3] examines a trained deep learning prediction model using technical indicators to predict stock price returns. This was effective in showing that using other market indicators apart from just price history could yield positive results in accuracy. The advantages of applying CNN-LSTM models for stock price prediction are evaluated by [10], stating that the advantages of CNN and LSTM over a plethora of other models exist to render the detection of spatial and temporal dependences of financial time series data. Therefore, the application of such complex deep learning models in a hybrid form possesses the possibility of higher prediction accuracy.

Likewise, [17] constructs a CNN-LSTM hybrid model and found positive results in prediction accuracy because of the effectiveness of combined convolutional and recurrent networks. In addition to certain deep learning architectures, externally derived features and salient stock features have been leveraged for training. For instance, [13] trained a unique deep learning architecture that integrates historical prices and significant trends in volatility and stock related news over time to greatly enhance prediction quality.

In addition, [14] used an LSTM algorithm to predict stock price and found it to be highly effective using merely historical price data and its time-series relationship. Additionally, [15] present a convolutional recurrent approach to Bitcoin price prediction which suggests we require both convolutional and recurrent layers to analyze trend fluctuations on minuscule and vast scales. This model achieves an accuracy greater than comparable model types, which implies that time series prediction works when utilizing blended approaches.

While much research limited itself to deep learning and the constraints of the data it generated, there were studies that included other variables from outside the dataset. For example, one study that determined macroeconomic indicators are important to

the stability and robustness of the trained models utilized machine learning time series analysis for stock price prediction with investment firms and macroeconomic trends. [16] Also, a study devoted to explainable AI for financial time series forecasting found that the use of interpretability techniques helps determine what aspects contribute to deep learning predictions. [12]

There are also recent findings about hyperparameter tuning. Reported by Ti (2024), after assessing various deep learning models, he concluded that trained models without hyperparameter tuning yield subpar results—meaning automatic hyperparameter tuning does seem to enhance the accuracy and reliability of the model. When it comes to deployment, the shift from testing to real-time applications went without a hitch. Therefore, [23] convened a conference on the deep learning applications and the assessment of real-time deployment was justified. Similarly, for deployment [11] utilized deep learning at stock market price prediction and subsequent assessment effective in real time.

But the pertinent research suffers, too. The vast majority of papers assess merely calibrating a model to achieve predicted performance irrespective of good quality and diverse input data. There are options upon options for one cryptocurrency or one stock price as if it's not related to anything else in the world and capricious assignment of data over time. Even worse, when predictions are correct, it's erroneous that they are correct based on the convoluted, nonstationary nature of financial time-series data. This study builds on similar research because it utilizes a deep learning model with a comprehensive cryptocurrency and stock dataset collated in one location. Where the aforementioned studies create a model to predict one asset at a time, our model predicts many assets simultaneously, meaning it considers cross-asset correlations that could enhance predictions. Furthermore, instead of testing for price prediction ability or directional prediction ability, as done in many studies like the one by Zhang, Chan, and Lin (2024) [10] and Liu, Li, and Li (2023) [24], we test both to reveal the advantages of multidimensional data for profitable outcomes.

3. System Architecture

Our capstone project for software engineering is a full stack web application that utilizes machine learning to predict cryptocurrency prices in real time. We have utilized system design concepts to create an architecture that is modular, scalable, and maintainable. Layers of the system connect to one another for communication yet serve independent purposes for effective functionality, performance, and future expansion ease. Ultimately, the architecture includes multiple tiers from client interface to API gateway to application layer to ML pipeline to database/storage. The client interfaces with the system through a web-based dynamic UI that issues RESTful API calls to the server side and establishes a WebSocket connection for server push updates.

The API gateway is Nginx, a reverse proxy for API request redirection, SSL termination, and load balancing, which enhances application performance and security. Flask is the framework used on the back-end. The back-end operates like a motherboard, processing API requests and WebSocket interactions while communicating with our app and the machine learning models. It requests the historical pricing, cleans it up, and sends it down the machine learning pipeline convolutional neural networks (CNN), long short-term memory (LSTM), CNN-LSTM, LSTM-CNN. It sends back to our app the derivative data used to create a prediction matrix on the dataset and relays the output back to the back-end to be sent to the user. Additionally, since Redis is the caching service used in the application, the back-end can perform actions without having to reprocess for quicker results.

We utilize Optuna for hyperparameter tuning to train and adjust our models to guarantee we're always using suitable parameters. As for our layers of storage, we use SQLite to store optimization results, Redis to store cached predictions, and we use the filesystem as well for model outputs and visualizations. This creates a level of save efficiency of storage for immediate rapid processing and long-term saves. We ensure security by using TLS, CORS, and rate limiting policies to ensure that only authenticated users can access and maintain data integrity. We ensure consistent functionality and reliability by using asynchronous processing with websockets for live streaming so that we can ensure responsiveness even under heavy load. Therefore, our application is operated reliably and performs effectively for real-time cryptocurrency price prediction due to scientifically based architecture developments and machine learning trained developments.The system data flow is displayed in Figure 3.2

The system architecture is displayed in Figure 3.1. While this section only provides a brief overview of the layers composing the architecture, sections three and four delve deeper into each layer client-server communication protocol, backend layers responsible for processing and prediction, and the trained machine learning validation pipeline constructed for validation of a successfully trained model. We assess how the frontend fulfills requests, how the backend composes with third-party repositories to fulfill data processing, and how trained machine learning models are reintegrated into the architecture to provide prediction.

From this flowchart 3.2, we can deduce that a machine learning structure is followed. There is a client interfacing with a Flask API for training and predicting. The notion that the API operates via WebSocket is encouraging, meaning that real-time renderings and results can be communicated back to the client during training and predicting. For training, the Data Processor gets input from an external source, cleans and transforms the data to pass along to the Model Pipeline, which trains the model based upon epochs. When it finishes, the model is saved. For predicting, the client pings the API to get predictions. The API loads the saved model and gets transformed data to give back predictions. Those predictions return to the client along with any visual renderings.

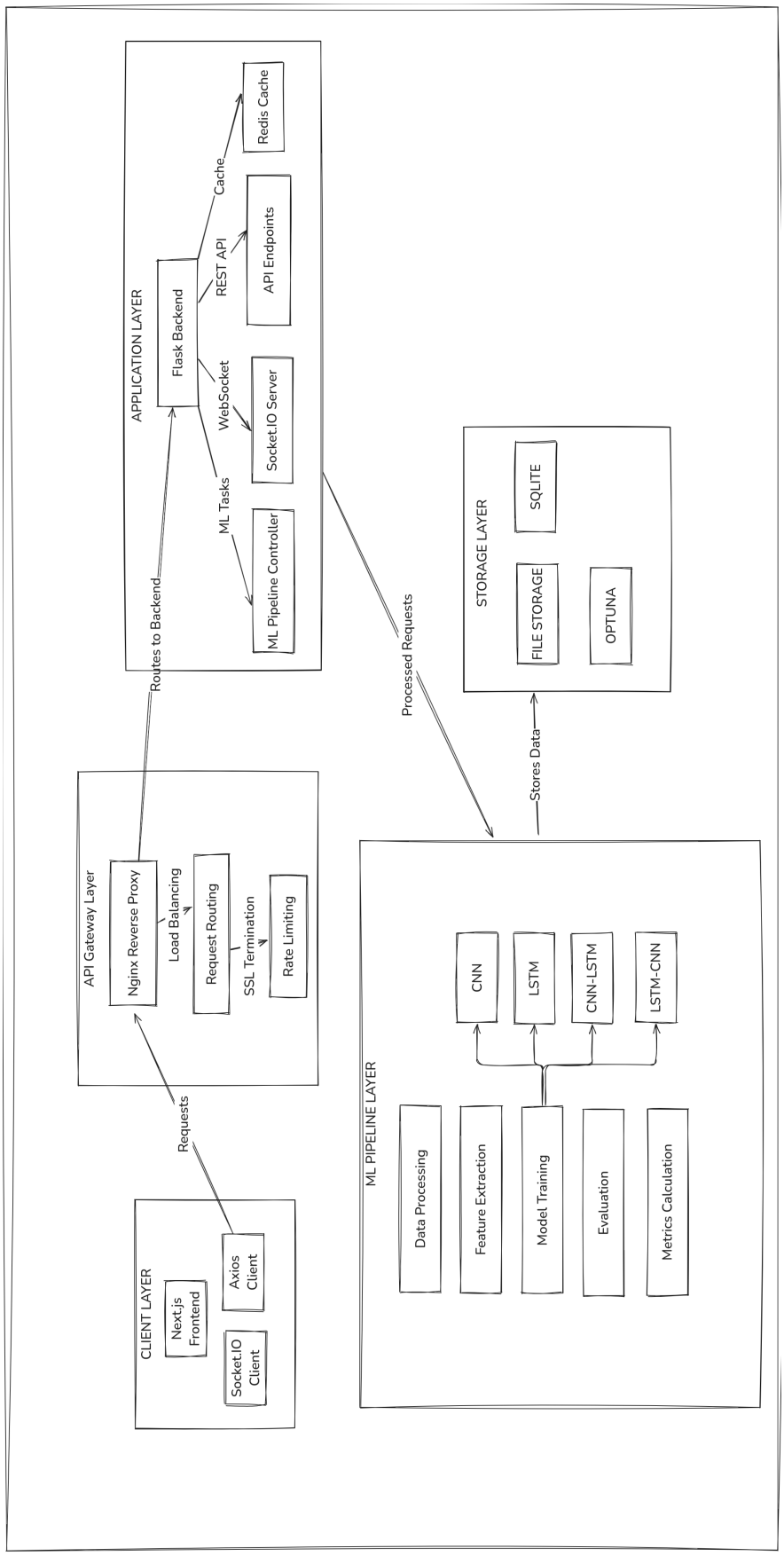

Figure 3.1. System Architecture Overview

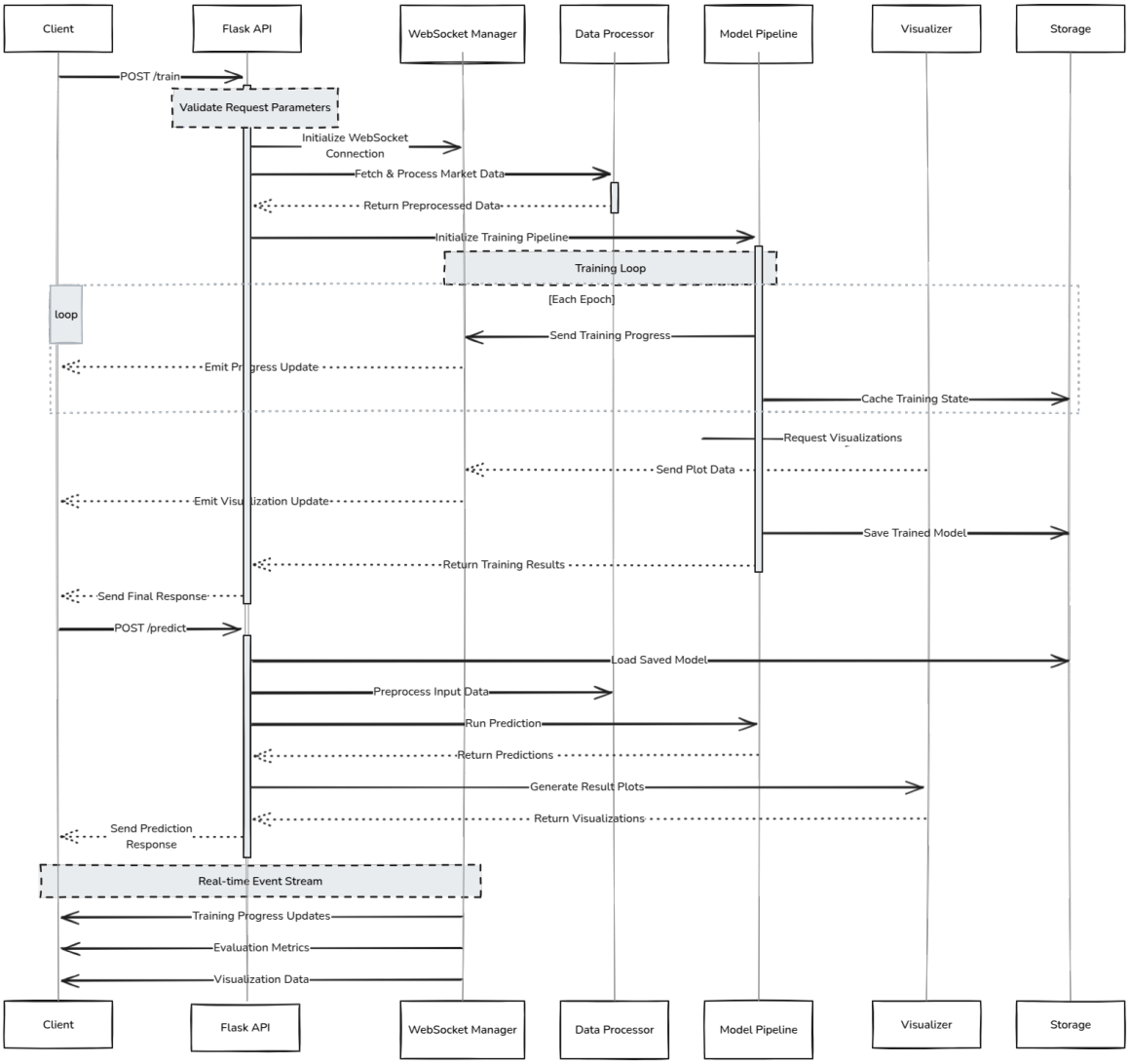

Figure 3.2. System Data Flow Diagram

4. Machine Learning Development

The task of predicting cryptocurrency prices is a broad and complex undertaking, as it requires assessing the unpredictable condition of the economy, and the nonlinear association with determining factors and relative transactions are not correctly assessed by typical financial models. Thus, a study of deep learning approaches is necessary. Therefore, the subsequent use of CNNs, LSTMs, and a hybrid of both (CNN-LSTM, LSTM-CNN) evaluates deep learning advancements for an effective and accurate predictive approach. In addition, the models undergo hyperparameter tuning with the Optuna hyperparameter tuning engine for optimization. Error metrics and predictive accuracy. This chapter outlines the dataset and relative data transformations, model configuration and training, and evaluation of experimental results.

4.1. Data

The source of the data for this project is daily historical data from 1.1.2021 to present. The three cryptocurrencies analyzed are Cardano, Ethereum, and Solana. These coins were selected based upon their market cap and volume traded so that the modeling would be more stable and any forecasting from such data would be applicable. The source of the data is from Yahoo Finance API (yfinance), which is credible for this currency. The test set was essentially three sets training, validation, and test set with $\tilde{6}0\%$ for training within 2021–2022. The validation set was 2023 in real-time data which determined if forecasting output was accurate. Therefore, the testing was to forecast 2023 values based upon the training/validation window. From July 2023 onward until the available timeline cut-off date, 20% of the testing data was reserved for the project. This was to guarantee sufficient unseen out-of-sample data to test for model generalization.

4.1.1. Descriptive Statistics and Data Characteristics

Table 4.1 provides summary statistics of the dataset, including key statistical measures such as mean, median, standard deviation, skewness, and kurtosis for each cryptocurrency.

Table 4.1. Descriptive statistics for ADA-USD, ETH-USD, and SOL-USD.

Asset	Minimum	Maximum	Mean	Std. Dev.	Median	Skewness
ADA-USD	0.175	3.098	0.892	0.512	0.895	0.48
ETH-USD	730.37	4,891.70	2,487.34	1,243.56	2,510.65	-0.32
SOL-USD	1.84	260.98	58.43	73.1	45.21	1.15

Figure 4.1 illustrates the price trends of the three cryptocurrencies over the study period, highlighting the volatility and price fluctuations that characterize these assets.

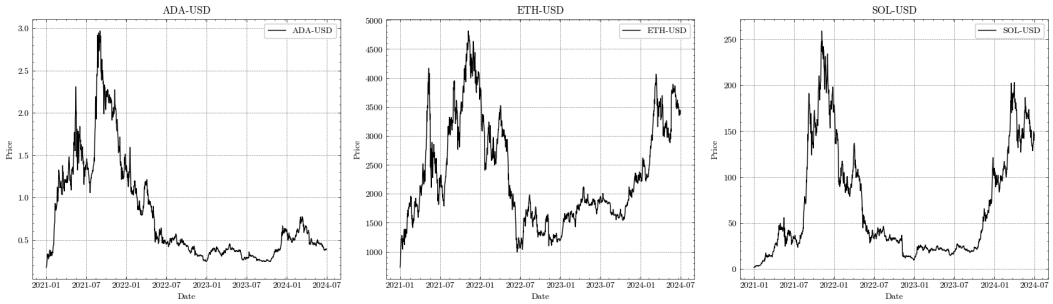

Figure 4.1. Daily Price Trends of Cardano (ADA-USD), Ethereum (ETH-USD), and Solana (SOL-USD)

4.1.2. Dataset Splitting and Normalization

We utilized transformation of the data set via MinMaxScaler to train the model because transforming each value to be situated within the [0, 1] range would not only normalize but also allow for no value to overpower model training based on a value larger than any other that was left in place, thus allowing for more equalized training. Following normalization, we established a train/validation/test split to ensure test performance was assessed on non-trained data. 60% constituted the training set, 20% the validation set, and 20% the testing set. This was a stratified sample and enables appropriate learning and performance assessment on unseen sampled data.Figure 4.2 is important to assess relative to how the data split for ADA-USD, ETH-USD, and SOL-USD. It presents a visual for series data trained, validated, and tested, with a dashed line representing where train/validation exists and a dotted line for validation/testing. Ultimately, however, the relevant sections where validation and testing exist are shaded in direct contrast to the lines so that it's clear where the focus resides for validation and testing to better assess generalization in an external/uncontrollable market situation apart from trained.

4.1.3. Stationarity and Data Transformations

To assess the stationarity of the time series data, we conducted the Augmented Dickey-Fuller (ADF) test. Table 4.2 presents the test statistics and corresponding p-values for the original and transformed series. The results indicate that the original time series for ADA-USD and ETH-USD are non-stationary, as evidenced by p-values greater than 0.05. However, after applying the log returns transformation, all series became stationary (p-value < 0.05), making them suitable for deep learning model training.

4.2. Preprocessing Techniques

The following preprocessing steps occurred:

- Missing Data Imputation: Linear interpolation is used to impute missing data. If gaps are too big, the entry for that observation will be dropped for the entire timestep.

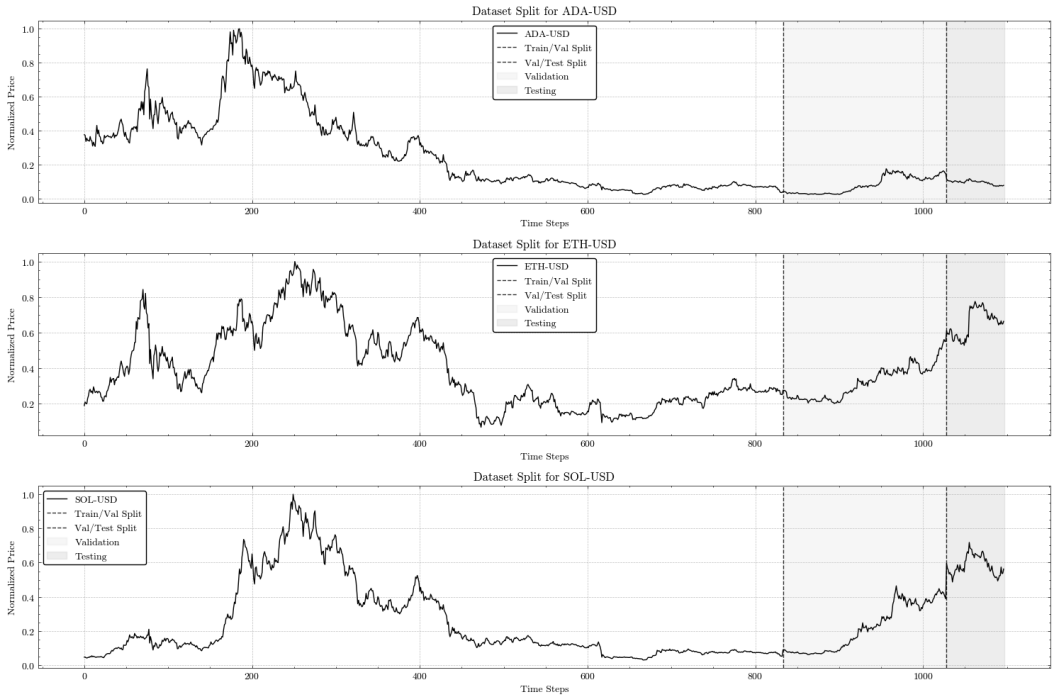

Figure 4.2. Dataset Splits for Cardano (ADA-USD), Ethereum (ETH-USD), and Solana (SOL-USD), Highlighting Training, Validation, and Testing Partitions

Table 4.2. ADF Unit Root Test Results for Cryptocurrency Time-Series

Time-Series	**ADF Statistic**	**p-value**
ADA-USD (Level)	-1.831	0.364
ETH-USD (Level)	-2.058	0.261
SOL-USD (Level)	-3.880	0.002*
ADA-USD (Returns)	-36.83	0.000*
ETH-USD (Returns)	-36.83	0.000*
SOL-USD (Returns)	-36.83	0.000*

- Outlier Detection and Correction: Outliers are detected using Interquartile Range (IQR). If an observation is above $1.5 \cdot IQR$, then it is either capped at the floor value or substituted with the median of that feature's distribution.
- Feature Normalization: MinMaxScaler transformation to ensure all numeric features are in a comparable [0, 1] range across various cryptocurrencies.
- Feature Engineering: In addition to the basic OHLCV features, additional technical indicators were created through the Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Exponential Moving Averages (EMA) for increased prediction efficacy.
- Data Creation: A 60-day sliding window created the dataset, which ensured time

series integrity as each window was used as input features to predict the target of the next day's closing price.

4.3. Model Architectures and Layers

This section provides a detailed mathematical explanation of the model architectures employed in our study. The architectures leverage convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to capture both spatial and temporal dependencies in time-series data.

4.3.1. Convolutional Neural Network (CNN)

CNNs are designed to extract spatial features from input sequences through convolutional operations, detecting local patterns such as upward or downward trends, crucial for financial forecasting. The convolution operation is defined as:

$$(f * g)(t) = \sum_{i=0}^{k} f(i)g(t-i), \tag{1}$$

where:

- $f(i)$ represents the input sequence,
- $g(t-i)$ represents the convolutional kernel,
- k is the kernel size.

If padding is applied, the effective input size changes as:

$$L_{out} = \frac{L_{in} - k + 2p}{s} + 1, \text{ (See, e.g., [25])} \tag{2}$$

where L_{in} is the input length, p is the padding, and s is the stride.

The activation function used is the Rectified Linear Unit (ReLU) [26]:

$$ReLU(x) = \max(0, x).$$

MaxPooling is applied to reduce dimensionality by taking the maximum value over a pooling window:

$$y_j = \max_{i \in P_j} x_i.$$

4.3.2. Long Short-Term Memory (LSTM)

LSTM networks specialize in sequential data processing by maintaining memory over long sequences, which helps capture long-term dependencies. Unlike traditional RNNs, LSTMs mitigate the vanishing gradient problem by introducing gated units [27]:

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

where i_t, f_t, and o_t are the input, forget, and output gates, respectively, and c_t is the cell state.

Here, the logistic sigmoid function $\sigma(x)$ is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

4.3.3. CNN-LSTM Hybrid Model

The CNN-LSTM model integrates CNN's ability to extract spatial features with LSTM's capability to model temporal dependencies. The convolutional layers first extract meaningful patterns before passing the processed sequence to LSTM layers.

Table 4.3. CNN-LSTM Model Architecture

Layer	**Description**
Input	Accepts a 60-day time-series sequence (shape: (60,1))
Conv1D	37 filters, kernel size 2, ReLU activation. Detects localized trends.
MaxPooling1D	Pool size: 1. Reduces complexity while maintaining key features.
LSTM (Layer 1)	103 units, return sequences=True, tanh activation. Captures longer-term dependencies.
LSTM (Layer 2)	16 units, return sequences=False, tanh activation. Finalizes sequential representation.
Dense (Hidden Layer)	50 units, ReLU activation. Combines extracted features.
Output	Single unit (linear activation) for price prediction.

4.3.4. LSTM-CNN Hybrid Model

The LSTM-CNN hybrid model starts with LSTM layers to capture temporal dependencies before applying convolutional layers for spatial feature extraction.
Parameters: The model contains a total of 52,934 trainable parameters, optimized for efficient learning and prediction.

Table 4.4. LSTM-CNN Hybrid Model Architecture

Layer	Description
Input	Accepts a 60-day time-series sequence (shape: (60,1))
LSTM (Layer 1)	81 units, return sequences=True, tanh activation. Captures long-term dependencies.
LSTM (Layer 2)	46 units, return sequences=True, tanh activation. Refines temporal representations.
Conv1D	95 filters, kernel size 4, ReLU activation. Detects localized trends and patterns.
MaxPooling1D	Pool size: 3. Reduces dimensionality while preserving key features.
Flatten	Converts feature maps into a one-dimensional vector.
Dense (Hidden Layer)	50 units, ReLU activation. Combines learned representations.
Output	Single unit (linear activation) for price prediction.

4.4. Hyperparameter Optimization

Hyperparameter tuning is essential to improve predictive accuracy and generalizability of deep learning. Several hyperparameter-related changes to the deep learning models are made to determine the best-suited configuration to avoid overfitting. Therefore, a hyperparameter tuning experiment was conducted in this context. The most effective automated hyperparameter optimization library was used—Optuna. The sampling parameter used to investigate the hyperparameter space was the Tree-structured Parzen Estimator (TPE). For pruning, median-based pruning was chosen to eliminate any trials that failed. The objective function to evaluate the The best hyperparameter selections from all the iterations by currency. Each model/cryptocurrency pairing went through hyperparameter tuning for 15 iterations for each cryptocurrency. Each iteration was the best parameter found by maximum accuracy. As for the combined baseline model, there were 15 iterations of the hyperparameter tuning per cryptocurrency, which equals 120 iterations in total. Also, for ETH-USD, there was a combined baseline model created with 15 more iterations per model, bringing its total to 240 iterations. The range for iterations was based upon batch sizes (16, 32, 64, 128), dropout (0.1 to 0.5), learning rate (1e-4 to 1e-2), dimensions of the convolutional filters (20 to 128), kernel size (2, 5), and LSTMs. The CNN-LSTM and LSTM-CNN used LSTM-related hyperparameters specific to both CNN and LSTM to ensure the hybridized model was done in the most efficient way possible. 4.3 illustrates where the hyperparameters chosen were applied. Histograms for batch size, dropout, learning rate, and total parameters across different models. These frequencies show the tuning tendencies and which parameters within what ranges yielded the best results.

The effectiveness of the hyperparameter tuning is illustrated in Figure 4.4.

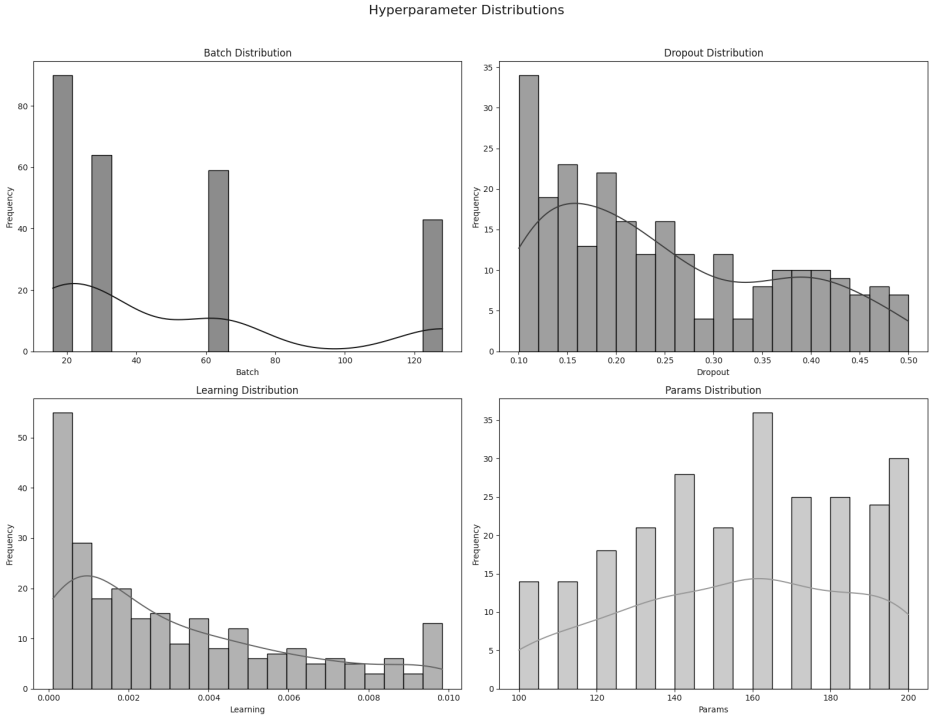

Figure 4.3. Hyperparameter Distributions Across Trials.

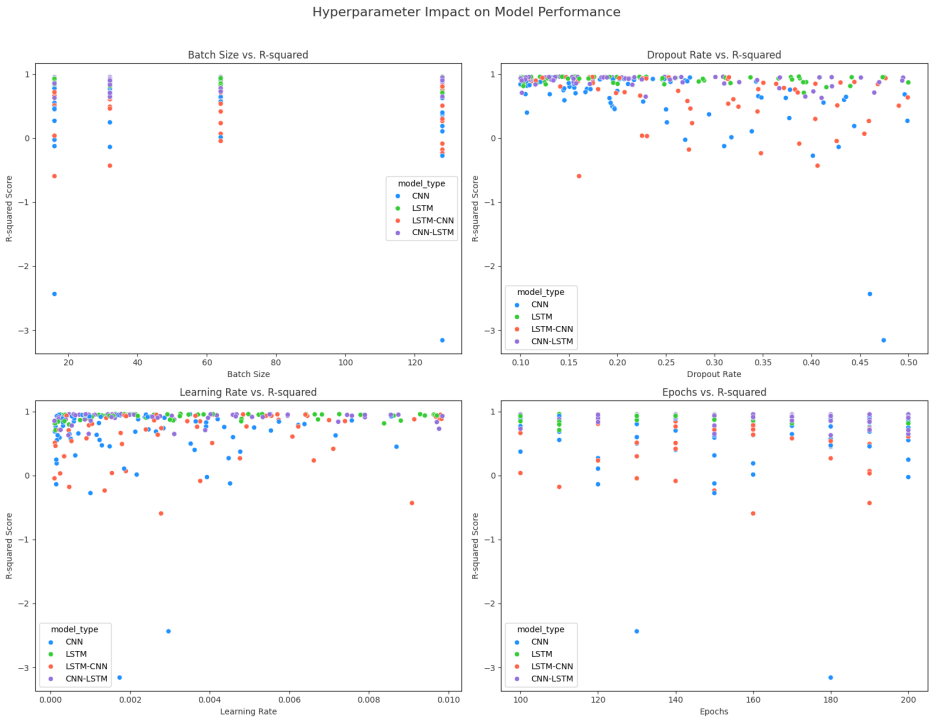

Figure 4.4. Hyperparameter Impact on Model Performance.

To evaluate model performance, three key metrics were employed: the R^2 score, Mean Squared Error (MSE), and Mean Absolute Error (MAE). The performance of the best models across different architectures is summarized in Table 4.5.

Table 4.5. Model Performance with Optimized Hyperparameters

Model	MSE	MAE	R^2
LSTM	2208.65	32.98	0.9542
CNN	7114.40	64.24	0.8524
CNN-LSTM	2229.16	33.65	0.9538
LSTM-CNN	6693.68	66.15	0.8611

Figure 4.5 provides a visual representation of the best R^2 scores achieved for each cryptocurrency ticker across different architectures, demonstrating that the LSTM-based models consistently outperform pure CNN models.

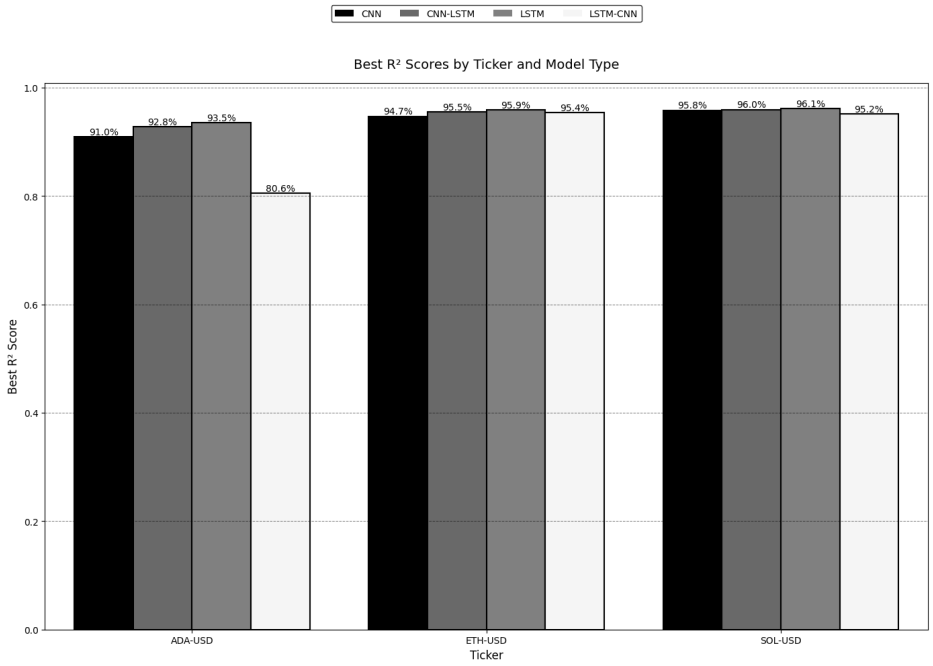

Figure 4.5. Best R^2 Scores by Ticker and Model Type.

Furthermore, all other regularization and optimization techniques were applied to guarantee the model was as precise and consistent as possible. For instance, early stopping was used with a 10-epoch patience to avoid overfitting. The ReduceLROnPlateau scheduler reduced the learning rate when increases and decreases of validation loss were observed. Batch normalization was used for stabilized training; dropout regularization was used to avoid overfitting and gradient clipping prevented overly powerful gradients—and there would be many, for example, with LSTM-based models to prevent exploding gradients.

Performance trends over time were also analyzed to assess the effectiveness of these strategies. Figure 4.6 illustrates the temporal evolution of R^2 scores and Mean Squared Error across different models, further validating the robustness of the hyperparameter optimization process.

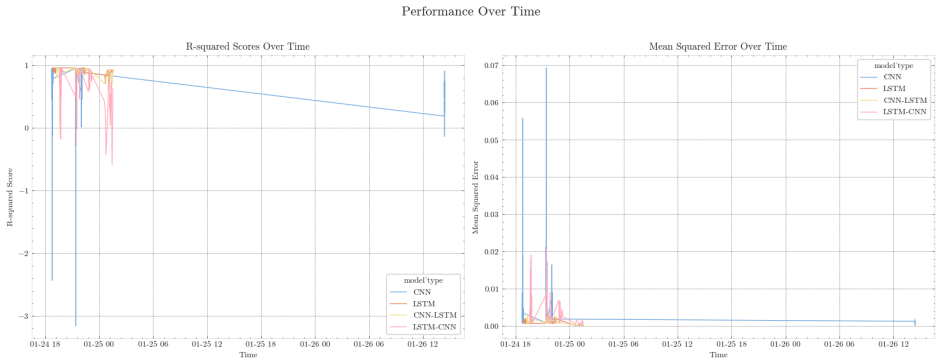

Figure 4.6. Performance Trends Over Time for Optimized Models.

These findings suggest that aside from improved forecasting performance of the LSTM-based architectures for cryptocurrency price prediction, other factors are at play. Standardized hyperparameter tuning makes the outcomes more translatable to reality. When manual and automatic tuning create consistency, the validity and correctness of the findings suggest that deep learning methods can be successfully used for financial time series prediction especially in cryptocurrency. To detailed study outputs are documented and added into apendicies 14 and optuna results for each dataset with all trials.

5. Backend Development

The cryptocurrency application server in this thesis is designed with modular, scalable, and maintainable considerations on the backend. The backend implementation is based on the Flask framework to provide RESTful API endpoints for model training and predictions, in addition to WebSocket for real-time feed. Docker is used for containerization. The system architecture consists of the following technologies. The web framework is Flask, and two-way real-time communication is created with Flask-SocketIO. The machine learning is performed via TensorFlow and Keras. Data processing and feature extraction are done through Pandas, NumPy, and Scikit-learn. Redis was used for cache and message queuing, and Pydantic for request validation and schema.

5.1. API Design and Implementation

The backend of the thesis project consists of RESTful API endpoints, well documented, and follows software development practices. The specific endpoints include one to train the model, one to create predictions, and one to check status. They include input validation, uniform logging, and access control mandated for enhanced security and dependability in addition to literally following backend bests for server creation.The architecture is modular and scalable, ensuring that future expansions or enhancements do not require a complete overhaul. Additionally, caching mechanisms are implemented to minimize lag and prevent server overload caused by redundant API calls

Table 5.1. REST API Endpoints

Method	Endpoint	Parameters	Description
POST	/train	{"ticker": str, "model": str, "start_date": str, "end_date": str, "lookback": int, "epochs": int}	Initiate model training.
POST	/predict	{"start_date": str, "days": int}	Generate predictions.
GET	/health	None	Check system health.

```
curl -X POST -H "Content-Type: application/json" -d '{
  "ticker": "BTC-USD",
  "model": "LSTM",
  "start_date": "2023-01-01",
  "end_date": "2023-01-10",
  "lookback": 60,
  "epochs": 50
}' http://localhost:5000/train
```

Listing 1. Request Example for /train

```
{
  "request_id": "20240328153045",
  "status": "success",
  "message": "Training started successfully",
  "data": {
    "train_metrics": {
    "mae": 123.45,"rmse": 156.78, "r2": 0.89,mape": 2.34,
    "mse": 24571.23 },
    "test_metrics": { "mae": 145.67, "rmse": 178.90,
    "r2": 0.86, "mape": 2.67, "mse": 32000.10 },
    "history": { "loss": [0.0234, 0.0156, 0.0123, ...],
    "val_loss": [0.0256, 0.0178, 0.0145, ...] },
    "training_duration": 145.67,
    "plots": {
      "training_history": "base64_encoded_image_data...",
      "predictions": "base64_encoded_image_data...",
      "residuals": "base64_encoded_image_data...",
      "error_distribution": "base64_encoded_image_data...",
      "qq_plot": "base64_encoded_image_data...",
      "prediction_scatter": "base64_encoded_image_data...",
      "bollinger_bands": "base64_encoded_image_data...",
      "momentum": "base64_encoded_image_data...",
      "volume_profile": "base64_encoded_image_data...",
      "drawdown": "base64_encoded_image_data..."
```

```
    }
  }
}
```

Listing 2. Response Example for `/train`

```
curl -X POST -H "Content-Type: application/json" -d '{
  "start_date": "2023-01-01",
  "days": 10
}' http://localhost:5000/predict
```

Listing 3. Request Example for `/predict`

```
{
  "request_id": "20240328153145",
  "status": "success",
  "data": {
    "predictions": [
      {
      "timestamp": "2023-01-02T00:00:00",
      "predicted_value": 42150.25,
      "lower_bound": 41950.75,
      "upper_bound": 42349.75,
      "confidence": 0.95 }
    ],
    "model_config": {
      "model": "LSTM", "lookback": 60, "batch_size": 64,
      "train_test_split": 0.8, "random_seed": 42,
      "ticker": "BTC-USD"
    },
    "data_range": {
      "start_date": "2023-01-01T00:00:00",
      "end_date": "2023-01-10T00:00:00",
      "prediction_start": "2023-01-11T00:00:00"
    }
  }
}
```

Listing 4. Response Example for `/predict`

```
curl http://localhost:5000/health
```

Listing 5. Request Example for `/health`

```
{"status": "healthy", "timestamp": "2024-10-27T10:00:00"}
```

Listing 6. Response Example for `/health`

5.2. Real-time Communication and WebSockets

Real-time updates are essential for actively monitoring the training process and visualizing model performance. We use WebSockets to transmit periodic updates regarding training loss trends, validation performance, and estimated completion times. This ensures the frontend displays the latest model progress.

Table 5.2. WebSocket Events

Event	Description	Example Payload
`training_update`	Training progress updates, including epoch, loss, and progress percentage.	`{"epoch": 10, "loss": 0.23, "progress": 20}`
`validation_update`	Validation metrics like MSE, MAE, and R-squared.	`{"mse": 2208.65, "mae": 32.98, "r2": 0.954}`
`visualizing_update`	Visualization plots (e.g., loss curves) as base64 encoded images.	`{"plot": "loss_curve", "image": "data:image/png;base64"}`
`error_log`	Error notifications with details.	`{"error": "Training failed", "details": "Insufficient data"}`

The following visualizations are streamed via WebSockets in real-time: Training History, Model Predictions, Residual Analysis, Error Distribution, Q-Q Plot, Actual vs. Predicted Scatter Plot, Rolling Performance Metrics, Price Action (Candlestick Chart), Bollinger Bands Analysis, Momentum Indicators, Volume Profile Analysis, and Drawdown Analysis.

5.3. Request Validation with Pydantic

We utilize Pydantic models to validate all incoming API requests, ensuring data integrity and preventing common vulnerabilities. This enforces schema consistency and contributes to the robustness of the backend.

```
1  from pydantic import BaseModel, Field, field_validator
2  from typing import Optional
3  from datetime import date
4
5  class PredictionRequest(BaseModel):
6      start_date: date
7      days: Optional[int] = Field(default=15, ge=1, le=30)
8
9      @field_validator('start_date')
10     def start_date_must_be_before_today(cls, v):
11         if v > date.today():
```

```
            raise ValueError('start_date cannot be in the future')
        return v

class TrainingRequest(BaseModel):
    ticker: str = Field(..., min_length=1, max_length=10)
    model: str = Field(..., min_length=1, max_length=10)
    start_date: date
    end_date: date
    lookback: Optional[int] = Field(default=8, ge=1, le=60)
    epochs: Optional[int] = Field(default=100, ge=1, le=1000)

    @field_validator('end_date')
    def end_date_must_be_after_start_date(cls, v, info):
        start_date = info.data.get('start_date')
        if start_date and v <= start_date:
            raise ValueError('end_date must be after start_date')
        return v

    @field_validator('ticker')
    def validate_ticker(cls, v):
        if not all(char.isalnum() or char == '-' for char in v):
            raise ValueError('ticker must contain only letters,
            numbers, and hyphens')
        return v.upper()
```

Listing 7. Pydantic Validation Examples

5.4. Redis Integration

Redis plays a crucial role in optimizing the backend by serving as both a caching layer and a message broker.

Caching: We use Redis to store frequently accessed data, such as historical price data and intermediate model predictions. This reduces redundant computations and database queries, improving latency and response times. Incoming requests first check the Redis cache. A cache hit retrieves data directly from Redis, bypassing slower data sources. A cache miss triggers fetching from the source, performing calculations, and then storing the result in Redis for future hits. A well-designed caching strategy, including appropriate cache

6. Frontend Development

This section outlines the development of the cryptocurrency thesis project's front end. Ultimately, the front end would allow a user to realistically predict prices in real-time, observe trained models, and receive incremental updates via WebSocket in a responsive, user-friendly, and accessible setting. Frontend development focused on UI/UX, state

control, and data visualization to ensure the smoothest experience for anyone attempting to trade or just analyze the information. To ensure ease of development and a uniform user-facing experience, much of the component library Shadcn UI was used with Nextjs.

6.1. Architecture and Design

Frontend is constructed with Next.js and a component-based architecture for modular, maintainable, and scalable endeavors. Axios is used to fetch API data, and React Query is used for caching and establishing server state in the background and in the foreground. Zod is used throughout the application for schema validation for universal data use. State management is a scalable effort, and the integration of React Query and React Hook Form provides application logic and form validation, respectively. WebSockets enable real-time updates to guarantee instantaneous results of training efforts and seen predictions. The user interface is responsive with a mobile-first approach and adjustability across screen sizes, dark/light themes, and accessibility features for any user need.

6.2. Dashboard Integration

Instead of separate applications and screens for analysis, a dashboard was created to house the various widgets for prediction market analysis, prediction accuracy, and live training status for a unified, cohesive critical experience. Implemented with a grid layout where elements take only what is needed on the screen or device to display, it is highly responsive no matter the sizing. Yet, one of the best features of the dashboard is the ability to resize panels and dictate how segmented one's screen should be by the various widgets. ResizablePanel and ResizablePanelGroup components give dedicated space so live market fluctuation metrics can take up an entire pane of the screen without overlap from machine learning accuracy metrics. If one insight is more important than another at a specific time, it's easily adjustable without losing access to critical information. As for the aesthetic nature of the dashboard, ShadCN UI components were integrated into a consistent, modern inclusive experience across varying hierarchies to ensure a seamless flow from navigation to forms to animated components for the most proactive and retroactive use of these systems. Furthermore, static data generates in real time to toggle from historical data trends to current pricing with conditional filtering for interest in specific cryptocurrencies or certain windows of time. Thus, these functionality-driven additions create a financial analytical hub that encourages versatility and ease of use for real-time assessment opportunities. Figure 6.1 shows a glimpse of the entire system with major market analyses and machine learning observance functionality.

6.3. Widgets

Our frontend consists of multiple widgets and reusable components, each designed to provide real-time insights into cryptocurrency trends and model performance.

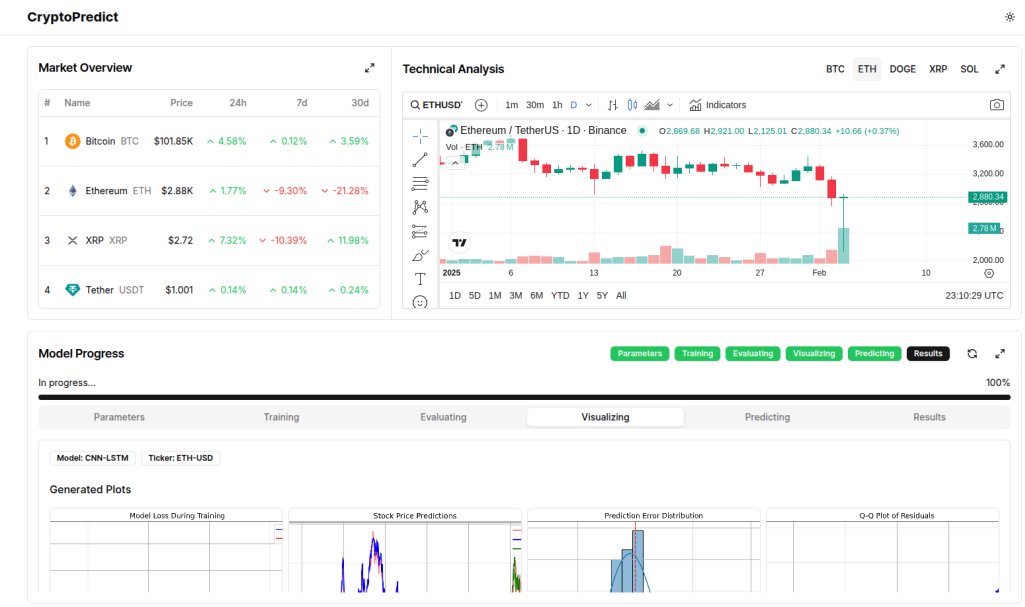

Figure 6.1. Cryptocurrency Prediction Dashboard

6.3.1. Market Overview Widget

We build the front end in Next.js. Therefore, we adopt a modular component-based approach for better maintainability and scalability. We make API calls using Axios and employ React Query for server state management and caching, including background-fetching refetches automatically. We use Zod for schema validation. We apply a similar state management approach to ensure consistent state management methodology throughout the app. We utilize React Query and React Hook Form for form validation and optimal state management. We employ WebSockets for real-time updates so users see their training status and predictions updated as they work on their training. Finally, we have a mobile-first responsive design. We ensure theme accessibility as well and supports the dark themes and light themes.

Figure 6.2 displays the Market Overview Widget, highlighting key market indicators fetched from the CoinGecko API.

6.3.2. Technical Analysis Widget

The Technical Analysis widget provides a summary of trading indicators such as RSI, MACD, and moving averages. In addition, we selected the **TradingView** Advanced Charting Widget integration because it offers users the opportunity to use and discover a plethora of technical trading options and features—configurable indicators, annotation tools, and configurations for chart types. These indicators help to determine various trends to make proper trading decisions. Notable features include live buy and sell recommendations based on technical trading indicators, the ability to create support and resistance lines, and customizable settings for various trading indicators.

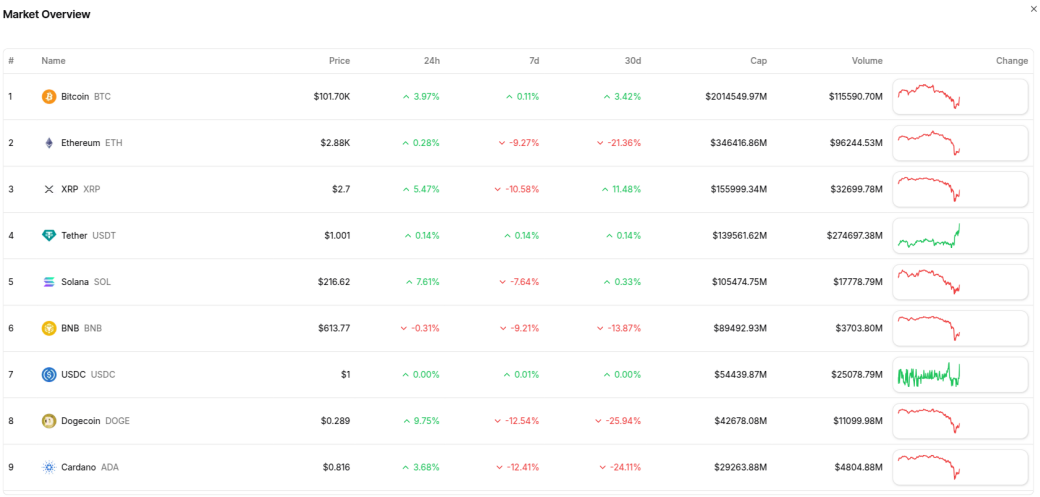

Figure 6.2. Market Overview Widget

Figure 6.3. Technical Analysis Widget

Figure 6.3 demonstrates the Technical Analysis Widget, showcasing the integrated TradingView charting tools.

6.3.3. Machine Learning Model Widget

The Machine Learning Model Widget is a main widget in dashboard that can train and test selected machine learning models to dynamically predict cryptocurrency prices and render such predictions visually as it operates. Furthermore, the web application includes a guided step-by-step process to ensure you can engage successfully with every part of making the model and rendering the prediction visualization. For example, the initial step involves the training parameters, which include which cryptocurrency ticker (BTC-USD, ETH-USD, etc.), lookback period (How many days back will be analyzed to predict price?), type of model (CNN, LSTM, CNN-LSTM, LSTM-CNN), batch size , and epochs. In addition, the provided configurations permit the tuning of the learning rate,

batch size, and epochs. Moreover, by selecting the custom beginning and ending dates option, the user can specify which historical data he or she wants to utilize. Yet after all of this is set, the final action is to push the training button, which executes a POST request to the '/train' route in the back-end to start the modeling training pipeline.

As the model trains, for example, the widget updates the user in real-time via Web-Sockets a live, predictable, continuous increments of where they are within the epoch, decreased training/validation loss, and predicted time remaining. These increments provide awareness to the user of how their adjustments and rectifications affect over time regarding subsequent epochs and whether they've appropriately trained their model. In addition, receiving real-time updates brings the user up to speed with every achievement instantly. Subsequently, after the widget trains, it enters the testing phase whereby indicators of model success are offered including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R^2, Mean Absolute Percentage Error (MAPE), and Mean Squared Error (MSE) during both the training and testing stages. The thorough assessment of success aids in modeling reliability and optimal choice for application. In addition to being able to visualize training loss by epoch and in graphic and rolling averages/st. deviations, one can see predicted vs. actual increases/decreases and related distributions and Q-Q plots of residuals visually. There are also candlestick charts with Bollinger Bands for additional technical analysis. Ultimately, forecasting is always an option. By selecting a forecasting horizon (i.e., 15 days) and a starting point of where to begin forecasting, for example, the widget employs the fitted model to generate a forecast and displays the findings with corresponding upper and lower bounds to gauge variability in forecasting to ensure accurate budgeting.

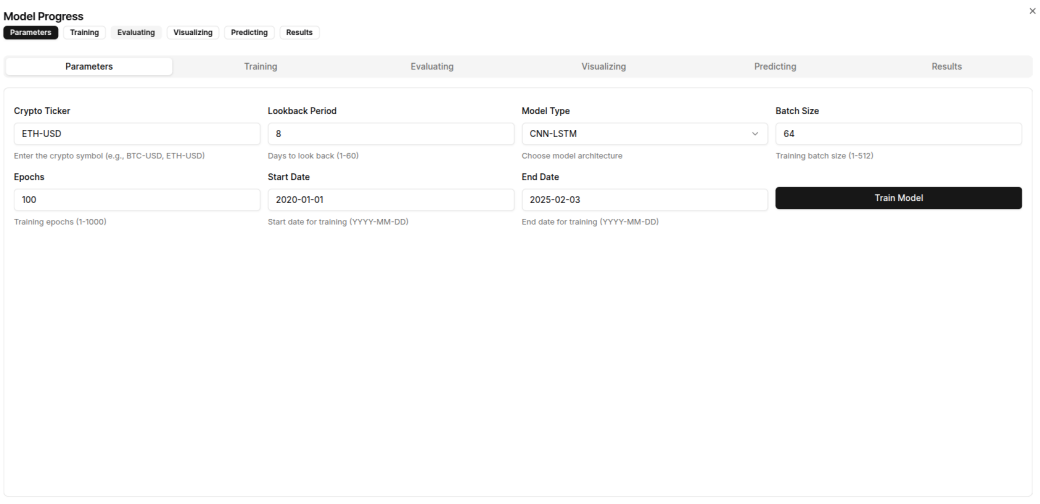

Figure 6.4. Phase 1: Training Form

Figures 6.4 through 6.9 illustrate the various stages of the Machine Learning Model Widget, from training initiation to prediction results.

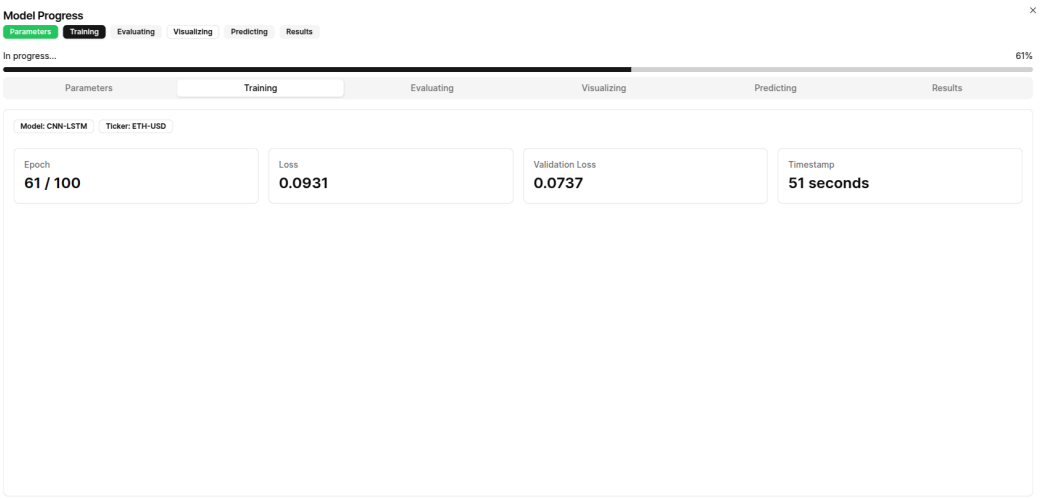

Figure 6.5. Phase 2: Training in Progress

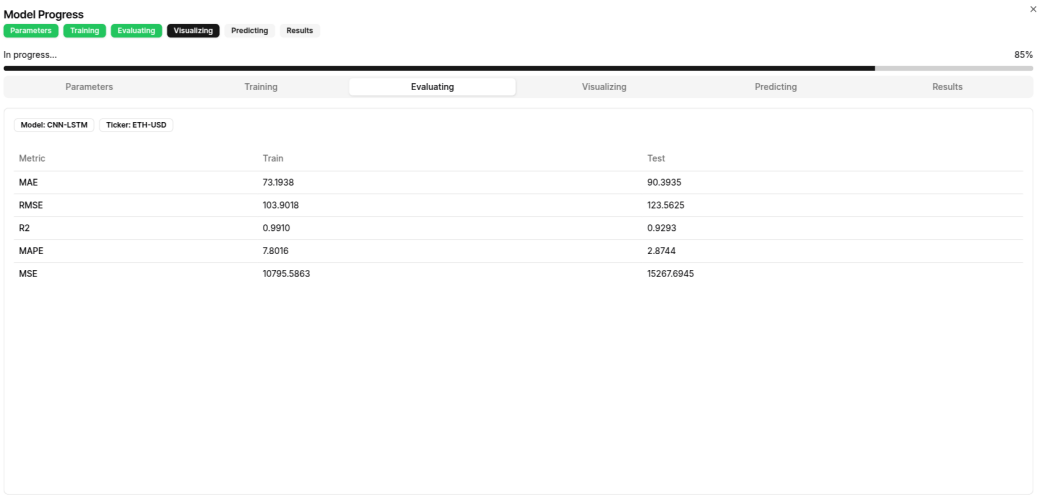

Figure 6.6. Phase 3: Evaluation Metrics

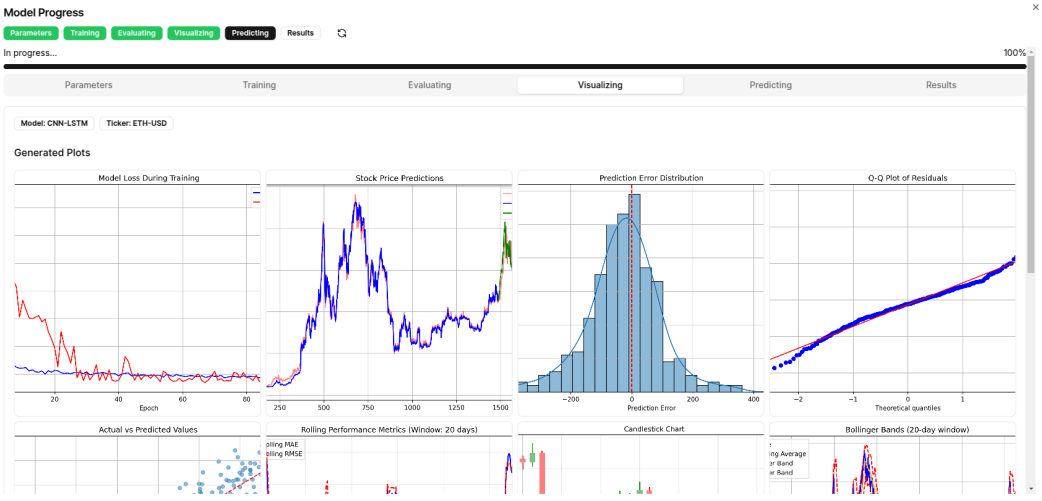

Figure 6.7. Phase 4: Generated Plots

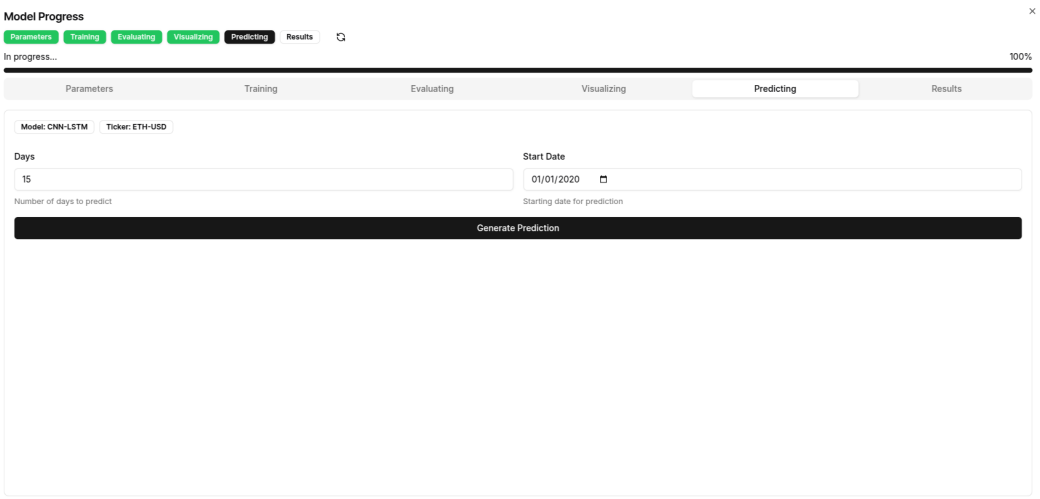

Figure 6.8. Phase 5: Prediction Form

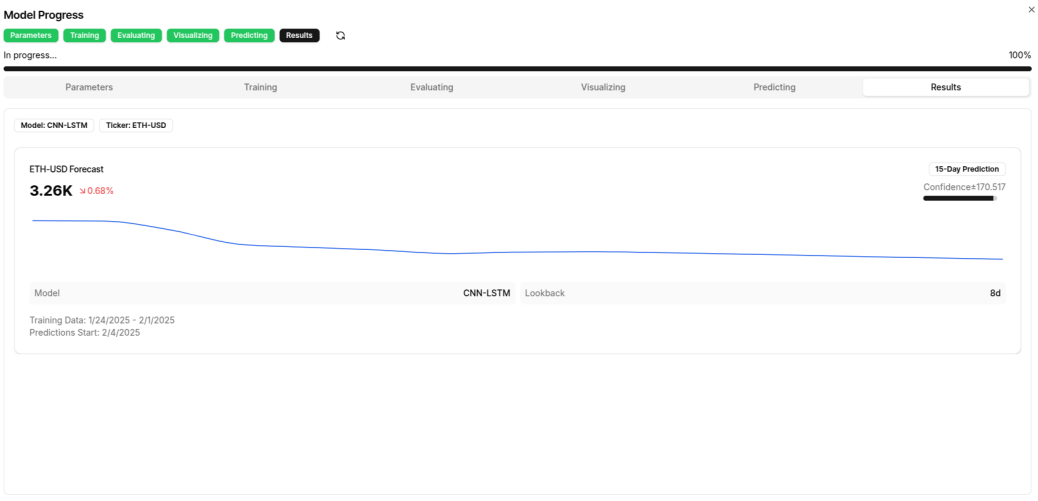

Figure 6.9. Phase 6: Prediction Results

6.4. WebSocket Communication

Socket.IO enabled WebSocket communication on the front end; a custom React hook was developed to handle the WebSocket connection for functionality and implementation 15. WebSocket communication on the front end is essential because this portion of the project needs real-time updates during model training and subsequent price prediction and service retrieval. Therefore, having separate WebSocket streams allows the user to receive real-time updates on critical functions and enhances overall functionality. For instance, during model training, loss metrics are displayed to the end user in real time along with training progress and estimated time of arrival. Prediction works the same way. As visualizations happen, more data is constantly being inputted into the model and similarly, losses are calculated on dynamic graphs in real time. With real-time accuracy seeming through the roof, there was little to no latency. This reduction in latency comes from the use of WebSockets and the ability to give end users real-time data without having to refresh, pause, or restart progress ever.

6.5. API Integration

Our frontend application communicates with the backend through REST API endpoints. We employ React Query for our API calls, meaning we benefit from caching, error boundary creation, and background refetching. Thus, the chances of duplicate network requests are minimized and our application runs more seamlessly. Furthermore, we employ Zod for schema validation of returned API response data so that we experience type safety on the frontend instead of dealing with unexpected issues. All API calls are validated in order to ensure accurate data entry in addition to error handling. The application executes asynchronous calls with loaders and error messages. In addition, the use of WebSockets enables the sending of real-time updates to the client side as well without the need for a page refresh.

6.6. Theme

We also have a dark and light theme rendered on the frontend for user-selection viewing, which is crucial relative to the user preferences. We use the next-themes library to achieve theme switching. The ThemeProvider defaults to reading the system theme. With respect to styling, Tailwind CSS and CSS Modules are utilized to create a consistent, responsive, and low-maintenance look across the application. CSS Modules provide a level of conflict avoidance between components with scoping when appropriate, while Tailwind provides a utility-based approach for applying style. With regards to the persistence of user theme choices, the theme state is stored in local storage so that theme preferences are consistent across use. Also, we utilize React's Context API straight in our ThemeProvider, so state is exposed globally.

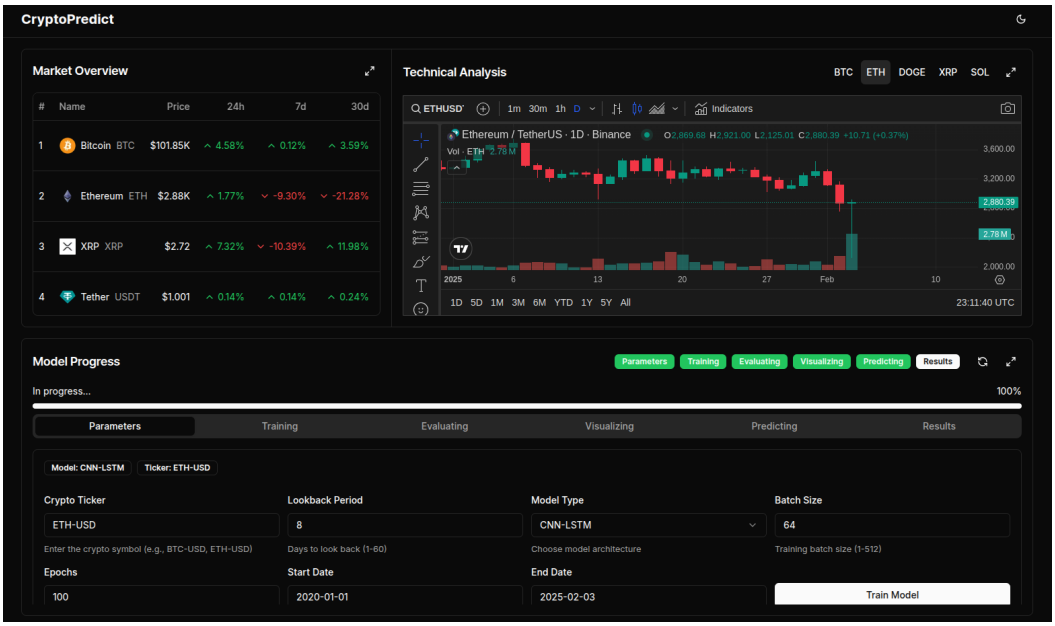

Figure 6.10. Dark Theme

Figure 6.10 demonstrates the dark theme option, providing an alternative visual experience. The dark theme is designed to reduce eye strain in low-light environments, while the light theme is suitable for well-lit settings. Users can easily toggle between the two themes using a designated control within the application's settings.

6.7. Environment Configuration

We use .env files to guarantee our environment variables for configuration variables such as API endpoints, WebSocket links, and integration third-party URLs. Thus, irrespective of the configuration established, we can interchangeably use dev, staging, and production environments with no need to adjust any code. As far as .env file deployment is concerned, each environment has its own configuration.

```
1 NEXT_PUBLIC_BASE_URL=http://localhost:5000
2 NEXT_PUBLIC_CRYPTO_URL=  % URL for additional crypto data
3 NEXT_PUBLIC_TRADING_VIEW_URL= % URL for TradingView widget
```

Listing 8. Environment Variables for Client

```
1 DOCKERHUB_USERNAME=username
2 DOCKERHUB_TOKEN=token
3 VERCEL_TOKEN=vercel_token
4 VERCEL_ORG_ID=org_id
5 VERCEL_PROJECT_ID=project_id
6 NEXT_PUBLIC_BASE_URL=http://localhost:5000
```

Listing 9. Environment Variables for CI/CD Github Actions

Listing 8 and 9 illustrates an example of how environment variables are configured. This separation of configuration from code promotes security best practices, as sensitive information is not embedded directly within the application's source code. It also simplifies the deployment process, as configurations can be easily adjusted for different environments.

7. DevOps and Deployment

This chapter is devoted to the DevOps methodology and deployment options to create this thesis study application in a sustainable, scalable, and stable manner. For ease of coupling operation during development and production—and within the GitHub repository , containerization, CI/CD tools, cloud hosting options, and security considerations are all addressed.

7.1. Containerization

Containerization is the primary deployment mechanism since it ensures that wherever the application runs, it'll run in the same environment. We use Docker to deploy a container, which is a lightweight, portable version of the application backend and all pertinent dependencies. Our backend container is simply the Python runtime, running the Flask app and its necessary libraries. Furthermore, we also run a Redis container on its own, but we only access it for caching and WebSocket event handling. We create our container from a Dockerfile, following containerization and security best practices, which efficiently leads to a minimal footprint image, necessary dependencies, and security precautions. For example, our installation consists of installing system dependencies, setting up environmental variables, and creating a non-root user to avoid permission issues later. However, this means that the application can run independently of the operating system and required packages, which, for further portability and repeatability, is a good thing. Furthermore, running under a non-root user is more secure because a smaller footprint exists should intrusions occur.

7.2. CI/CD Pipeline

We implemented a full CI/CD pipeline for automated testing, building, and deployment. Our CI/CD pipeline guarantees that code commits are checked against approved automated testing before deployment, reducing the likelihood of sending bugs to production. We operated in a monorepo setting, meaning the client and server applications were in one repo with all files, folders, and assets, all of which our CI/CD pipeline could quickly process.

Our CI/CD pipeline is orchestrated using GitHub Actions and consists of several interconnected workflows:

1. **Main CI/CD Workflow:** This is the overall global workflow that drives the operation behind everything and acts as the central orchestrator. It runs upon pushes and pull requests to main. It uses path filtering to run when changes occur within the client or server folder. It then starts the respective client or server workflows. This is a more effective testing and deployment approach because it can hone in on specifically what changed
2. **Client CI/CD Workflow:** This is the CI/CD workflow that executes for the client application. It runs tests (lint, type checking, unit tests) and upon successful completion, along with a successful push to the 'main' branch, builds the client application and deploys it to Vercel. It uses the required environment variables and uses 'vercel-action' to complete the task of deployment
3. **Server CI/CD Workflow:** This is the CI/CD workflow that executes for the server application. It runs linting, unit testing, and coverage testing. If there is a push to the 'main' branch, it builds the server application into a Docker image, tags the image, and pushes it to Docker Hub. This gives this workflow the opportunity to set up deployment on Render for the server app.

Automated monitoring tools are integrated within the pipeline to track system health and notify administrators in case of failures. This automated approach streamlines our development workflow, allowing us to rapidly iterate and deploy new features with confidence.

7.3. Cloud Deployment

We deployed our application on cloud platforms to ensure high availability and scalability. Two primary cloud hosting services are used:

Vercel We chose to deploy the frontend application to Vercel because of ease of deployment, scalability, and usage. Vercel allows for Git Integration, which means deployment happens automatically with Git commits as well; a developer does not need to push the latest version in production to implement it. The latest version goes live with a commit. Furthermore, Vercel has great performance relative to frontends with a global CDN. It doesn't matter where in the world the user is; they can access the site with high-speed load times. Vercel does well for scaling with traffic and demand; however, it can get costly. As usage increases, scaling must be taken into consideration. Lastly, Vercel, like AWS, has serverless functions and edge caching. Latency is reduced, and performance is enhanced.

Render Render is host to our backend application. Automatic scaling, managed databases, and SSL ensure that deployment and maintenance of the backend service are seamless. The backend, of which our CI/CD pipeline builds and deploys, operates in a compartmentalized and isolated environment on Render, making it a safe bet and functioning well for

our needs. Additionally, Render requires management features that absolve much of the necessary daily maintenance that would be required to sustain a backend.

7.4. Security Considerations

Application and user data security was embedded at various levels of the continuous integration/continuous deployment pipeline. The following represent the key security features generated: Frontend and backend communicate via TLS, meaning that no one but the application and the user can see what information is transferred; there's no third-party access. CORS is configured to ensure that only specified domains can access the API. There are API rate limits configured to ensure that nefarious users can't spam the system and take it down via DoS attempts. The API key and database credentials are stored as environment variables instead of hardcoded into the application.

7.5. Docker Compose Setup

We utilized a 'docker-compose.yml' file to manage multi-container deployments locally, ensuring seamless communication between services during development and testing. The setup defines the backend service, Redis container, and networking configurations.

```
version: "3.8"

services:
  redis:
    image: redis:alpine
    ports:
      - "63790:6379"
    volumes:
      - redis_data:/data
    networks:
      - app-network
  ml-server:
    build: .
    container_name: crypto-predictor-server
    ports:
      - "5000:5000"
    volumes:
      - ./models:/app/models
      - ./logs:/app/logs
    environment:
      - FLASK_APP=app.py
      - FLASK_ENV=production
      - PYTHONUNBUFFERED=1
      - CORS_ORIGINS=*
    depends_on:
      - redis
```

```
    networks:
      - app-network
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
networks:
  app-network:
    driver: bridge
volumes:
  redis_data:
```

Listing 10. Docker Compose Configuration

Listing 10 shows our Docker Compose configuration. This configuration ensures efficient service orchestration, allowing each component to communicate securely and reliably during development. We use named volumes for data persistence and health checks to ensure the backend service is running correctly.

7.6. Dockerfile

The Dockerfile for the server application was created with an eye towards efficiency, security, and maintainability. For the study it was followed much of the best practice suggestions provided to create a container that would be secure, stable, and quickly scalable.

```
FROM python:3.9-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    build_essential \
    python3-dev \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

RUN pip install gunicorn eventlet

COPY . .

RUN mkdir -p models

ENV PYTHONUNBUFFERED=1
ENV FLASK_APP=app.py
```

```
ENV FLASK_ENV=production
ENV PORT=5000
ENV REDIS_URL=redis://redis:6379/0

EXPOSE 5000

RUN useradd -m myuser
RUN chown -R myuser:myuser /app
USER myuser

CMD ["gunicorn", "--config", "gunicorn_config.py",
"--worker-class", "eventlet", "app:app"]
```

Listing 11. Dockerfile for Server Application

The Dockerfile for the server application is attached in Listing 11. It uses a Python 3.9-slim image as its base so that the container is as lightweight as possible with only the necessary requirements to operate—even though it has the capacity for all requirement-based dependencies. The RUN command for the setup installs necessary system and Python dependencies, the optimal use of COPY moves application code, and ENTRYPOINT and CMD ensure that appropriate functionality occurs when in production. Ultimately, a non-root user is created, and ownership of permissions is altered upon execution for security and separation of concerns. The app runs on Gunicorn within the server using eventlet as the class of workers to facilitate asynchronous processing. Ultimately, this is a containerized application based on minimal security and maintainable, scalable containerized environments for the necessary cryptocurrency price prediction application. The CI/CD pipeline ensures rapid deployment with frill features operational under low and high stress. Therefore, to cloud containerize with CI/CD to containerize with security is to ensure everything runs secure, stable, scalable, and efficient.

7.7. GitHub Repository

The source code for this cryptocurrency price prediction system is publicly available as an open-source project and can be accessed at **here**.

8. Conclusion

This thesis study offers a novel hybrid neural network modeling technique of hyperparameter tuning via Optuna and application across multiple deep learning architectures to forecast cryptocurrency values. This is in relation to enhanced value forecasting precision as all models met an R-squared of 0.9582 for CNN, 0.9614 for LSTM, 0.9544 for LSTM-CNN, and 0.9596 for CNN-LSTM. The various trained models are put through a robustness experiment of 100+ hyperparameter combinations to produce four different architectures of the four models with subsequent retraining validating the increases in

accuracy and decreases in error. The relative results of this study indicate that the LSTM model had the Minimum Squared Error (MSE) of 0.000627 and Achieved the lowest Mean Absolute Error (MAE) of 0.018803 on SOL while no architecture surpassed the stability of the CNN-LSTM hybrid (dropout: 0.140; learning rate: 0.00178). Moreover, we observed that the optimal configurations vary per architecture: CNN requires a batch size of 128 and 150 epochs (filters 74, 54; kernel sizes 3, 5); LSTM needs a batch size of 64 and 200 epochs (units 113, 55); CNN-LSTM requires a batch size of 32 and 160 epochs (LSTM units 103, 16; filters 37). Lastly, our findings indicate that smaller kernels and the use of both batch normalization and In terms of results, achieving momentum with the LeakyReLU activation functions ensured the architectures arrived at a more stabilized, better-fitted model. The SOL cryptocurrency was also the most predictable, as the architecture kept an average R-squared value of above 0.95. Beyond empirical significance, this research possesses practical significance as well. The use of WebSocket for real-time training feedback and Redis for message queuing creates a feasible real-time setting with options within the models dynamically updated in real time. Furthermore, the expected mean average error (MAE) was consistent across all three versions—ranging from 0.018803 to 0.019855—meaning the findings replicated a successful predictability of the rendered frameworks. Such findings not only... Thus, this not only contributes to the theoretical expansions of deep learning methods in financial time series prediction, but also provides the experimental basis for implementing algorithmic trading and evaluating real-world applications. Future testing setups may involve more hybridizations, different attentions, or even trading strategies based upon reinforcement learning for improved accuracy of predictions and adaptability of systems in fluctuating financial environments.

References

[1] J. Wu, X. Zhang, F. Huang, H. Zhou, and R. Chandra, *Review of deep learning models for crypto price prediction: Implementation and evaluation*, 2024. arXiv: 2405.11431 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.11431.

[2] X. Zhang, Z. Chen, and S. Wang, "A study of the impact of cryptocurrency price volatility on the stock and gold markets", *Finance Research Letters*, vol. 69, p. 106 114, 2024, ISSN: 1544-6123. DOI: https://doi.org/10.1016/j.frl.2024.106114. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1544612324011437.

[3] P. V, V. Devi, and K. M., "Accurate cryptocurrency price forecasting using deep learning techniques: A comparative analysis of daily and high-frequency predictions", *IJARCCE*, vol. 12, Jun. 2023. DOI: 10.17148/IJARCCE.2023.12682.

[4] Q. Guo, S. Lei, Q. Ye, and F. Zhiyang, "Mrc-lstm: A hybrid approach of multi-scale residual cnn and lstm to predict bitcoin price", Jul. 2021, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9534453.

[5] N. Gradojevic, D. Kukolj, R. Adcock, and V. Djakovic, "Forecasting bitcoin with technical analysis: A not-so-random forest?", *International Journal of Forecasting*, vol. 39, no. 1, pp. 1–17, 2023, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2021.08.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021001230.

[6] I. E. Livieris, N. Kiriakidou, S. Stavroyiannis, and P. Pintelas, "An advanced cnn-lstm model for cryptocurrency forecasting", *Electronics*, vol. 10, no. 3, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10030287. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/287.

[7] C. Tiwari, S. Pillai, A. Obaid, A. Saear, and A. Sabri, "Integration of artificial intelligence/machine learning in developing and defending web applications", Sep. 2023, p. 060 038. DOI: 10.1063/5.0171097.

[8] S. Chavhan, P. Raj, P. Raj, A. K. Dutta, and J. J. C. Rodrigues, "Deep learning approaches for stock price prediction: A comparative study of lstm, rnn, and gru models", in *2024 9th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2024, pp. 01–06. DOI: 10.23919/SpliTech61897.2024.10612666.

[9] G. L. Gil, P. Duhamel-Sebline, and A. McCarren, "An evaluation of deep learning models for stock market trend prediction", *[arXiv.org](http://arxiv.org/)*, vol. abs/2408.12408, Aug. 2024. DOI: 10.48550/arxiv.2408.12408.

[10] J. Zhang, W.-H. Chan, and Y. Lin, "Stock price prediction research based on cnn-lstm", *Highlights in business, economics and management*, vol. 41, pp. 667–676, Oct. 2024. DOI: 10.54097/cyqrxb84. [Online]. Available: https://drpress.org/ojs/index.php/HBEM/article/download/26304/25822.

[11] A. Ismailova, Z. Beldeubayeva, K. Kadirkulov, *et al.*, "Forecasting stock market prices using deep learning methods", *International Journal of Power Electronics and Drive*

Systems, vol. 14, no. 5, p. 5601, Oct. 2024. DOI: 10.11591/ijece.v14i5.pp5601-5611.

[12] J. Liu, "Stock market prediction model based on deep learning and enhancement of interpretabilit", *Academic journal of science and technology*, Nov. 2024. DOI: 10.54097/6r8hhv32.

[13] A. Tekriwal, "A novel deep learning model for stock prediction integrating historical prices, core volatility trends and financial news", *International journal of computer science and mobile computing*, vol. 13, no. 10, pp. 32–44, Oct. 2024. DOI: 10.47760/ijcsmc.2024.v13i10.004.

[14] Z. Wang, "Deep learning-based stock price prediction using lstm model", *Proceedings of business and economic studies*, Oct. 2024. DOI: 10.26689/pbes.v7i5.8611.

[15] O. M. Ahmed, L. M. Haji, A. M. Ahmed, and N. M. Salih, "Bitcoin price prediction using the hybrid convolutional recurrent model architecture", Oct. 2023. DOI: 10.48084/etasr.6223.

[16] H. Zheng, J. Wu, R. Song, L. Guo, and Z. Xu, "Predicting financial enterprise stocks and economic data trends using machine learning time series analysis", *Applied and Computational Engineering*, Jul. 2024. DOI: 10.54254/2755-2721/87/20241562.

[17] A. Liu, J. Li, and H. Ye, "A prediction model combining convolutional neural network and lstm neural network", pp. 318–321, Jul. 2023. DOI: 10.1109/aiars59518.2023.00071.

[18] F. Aksan, Y. Li, V. Suresh, and P. Janik, "Cnn-lstm vs. lstm-cnn to predict power flow direction: A case study of the high-voltage subnet of northeast germany", *Sensors*, vol. 23, no. 2, p. 901, Jan. 2023. DOI: 10.3390/s23020901.

[19] Z. Ti, "Stock prediction using deep learning: A comparison", *Transactions on computer science and intelligent systems research*, vol. 6, pp. 346–351, Oct. 2024. DOI: 10.62051/b4fefr32.

[20] X. Zhang, "Analyzing financial market trends in cryptocurrency and stock prices using cnn-lstm models", Jul. 2024. DOI: 10.20944/preprints202407.1119.v1.

[21] Z. He, H. Zhang, and L. Y. Por, "A comparative study on deep learning models for stock price prediction", *Advances in transdisciplinary engineering*, Jul. 2024. DOI: 10.3233/atde240502.

[22] C. Yinka-Banjo, M. Akinyemi, and B. Er-rabbany, "Stock market prediction using a hybrid of deep learning models", May 2023. DOI: 10.61549/ijfsem.v2i2.111.

[23] J. Wu, X. Zhang, F. Huang, H. Zhou, and R. V. Chandra, "Review of deep learning models for crypto price prediction: Implementation and evaluation", May 2024. DOI: 10.48550/arxiv.2405.11431. [Online]. Available: https://arxiv.org/pdf/2405.11431.

[24] A. Liu, F. Wang, and L. Zhao, "Cnn-bilstm for cryptocurrency trend prediction", *arXiv preprint*, 2023. arXiv: 2310.08945 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2310.08945.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines”, *ICML*, pp. 807–814, 2010.

[27] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

List of Symbols and Abbreviations

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska
WEIRD – ang. *Western, Educated, Industrialized, Rich and Democratic*
LSTM – Long Short-Term Memory
MLP – Multilayer Perceptron
CNN – Convolutional Neural Network
RNN – Recurrent Neural Network
DNN – Deep Neural Network
SSN – Sparkling Neural Network
RBFNN – Radial Basis Function Neural Network
SOTA – State of the Art
ARMA – Autoregressive Moving Average
ARIMA – Autoregressive Integrated Moving Average
SARIMA – Seasonal Autoregressive Integrated Moving Average
MIDAS – Multiple Instance Detection and Segmentation
CEEMDAN – Complete Ensemble Empirical Mode Decomposition with Adaptive Noise
CNN-BiLSTM-ECA – Convolutional Neural Network - Bidirectional LSTM and Efficient Channel Attention
SSA – Singular Spectrum Analysis
SVM – Support Vector Machine
GWO – Grey Wolf Optimizer
ECA – Exponential Component Attention
SGD – Stochastic Gradient Descent
WOA – Whale Optimization Algorithm
KPSS – Kwiatkowski–Phillips–Schmidt–Shin (Test)
ADF – Augmented Dickey–Fuller (Test)
EEMD – Ensemble Empirical Mode Decomposition
GRU – Gated Recurrent Unit
VMD – Variational Mode Decomposition
FIVMD – Interval Variational Modal Decomposition
RMSE – Root Mean Square Error
MSE – Mean Square Error
MAPE – Mean Average Percentage Error
MAE – Mean Average Error
CSI – China Security Index
SCI – Shanghai Composite Index

List of Figures

List of Tables

List of Appendices

Appendix 1. Solana optuna hyperparameter results

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
1	15s	32	0.15	130			6.38e-03			3.0	4.84e-02	3.20e-03	0.80	CNN
2	14s	16	0.12	100			2.20e-03			2.0	3.76e-02	1.99e-03	0.88	CNN
3	25s	128	0.13	200			4.31e-04			1.0	2.36e-02	9.35e-04	0.94	CNN
4	14s	64	0.37	160			1.80e-04			2.0	7.06e-02	6.04e-03	0.63	CNN
5	26s	32	0.34	160			3.85e-04			1.0	3.54e-02	1.87e-03	0.88	CNN
6	19s	128	0.35	200			1.16e-03			3.0	6.96e-02	5.95e-03	0.63	CNN
7	14s	32	0.21	110			5.89e-04			2.0	4.21e-02	2.51e-03	0.85	CNN
8	9s	64	0.17	100			1.27e-04			2.0	5.44e-02	3.74e-03	0.77	CNN
9	14s	64	0.44	170			4.78e-04			3.0	6.76e-02	5.77e-03	0.64	CNN
10	20s	32	0.15	190			7.57e-03			3.0	4.17e-02	2.28e-03	0.86	CNN
11	23s	16	0.25	180			8.69e-03			3.0	8.66e-02	8.95e-03	0.45	CNN
12	18s	16	0.10	130			2.52e-03			2.0	2.86e-02	1.30e-03	0.92	CNN
13	20s	16	0.25	130			3.51e-03			1.0	1.98e-02	7.18e-04	0.96	CNN
14	20s	16	0.27	130			2.86e-03			1.0	3.25e-02	1.62e-03	0.90	CNN
15	18s	16	0.46	130			2.97e-03			1.0	2.28e-01	5.57e-02	-2.43	CNN
16	25s	16	0.21	140			1.51e-03			1.0	2.51e-02	1.03e-03	0.94	CNN
17	17s	16	0.50	120			4.48e-03			2.0	1.02e-01	1.18e-02	0.27	CNN
18	20s	16	0.31	150			4.51e-03			2.0	1.29e-01	1.83e-02	-0.12	CNN
19	21s	16	0.20	120			1.75e-03			1.0	2.39e-02	1.00e-03	0.94	CNN
20	14s	128	0.11	110			1.70e-03			1.0	2.30e-02	8.72e-04	0.95	CNN
21	24s	16	0.20	150			8.49e-04			2.0	2.75e-02	1.25e-03	0.92	CNN
22	19s	16	0.25	120			4.20e-03			1.0	3.50e-02	1.97e-03	0.88	CNN
23	22s	16	0.19	140			2.40e-03			1.0	2.13e-02	8.12e-04	0.95	CNN
24	23s	16	0.18	140			8.99e-04			1.0	2.86e-02	1.34e-03	0.92	CNN
25	22s	16	0.10	140			2.01e-03			1.0	2.07e-02	7.58e-04	0.95	CNN
26	20s	16	0.11	140			2.44e-03			2.0	2.92e-02	1.42e-03	0.91	CNN
27	23s	16	0.11	160			1.22e-03			1.0	4.46e-02	2.95e-03	0.82	CNN
28	11s	128	0.16	140			5.54e-03			2.0	6.28e-02	4.84e-03	0.70	CNN
29	12s	128	0.14	150			5.53e-03			2.0	1.99e-02	6.78e-04	0.96	CNN
30	13s	128	0.16	170			6.22e-03			2.0	5.03e-02	3.44e-03	0.79	CNN
1	39s	16	0.25	140			2.19e-04				2.79e-02	1.31e-03	0.92	LSTM

Table Appendix 1.1 Continued from previous page

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
2	10s	32	0.10	180			2.42e-04				2.02e-02	7.23e-04	0.96	LSTM
3	25s	16	0.22	100			3.11e-04				2.31e-02	9.33e-04	0.94	LSTM
4	59s	64	0.29	120			2.40e-03				3.29e-02	1.61e-03	0.90	LSTM
5	35s	32	0.19	180			1.22e-03				2.30e-02	8.90e-04	0.95	LSTM
6	55s	32	0.31	170			8.98e-04				1.91e-02	6.46e-04	0.96	LSTM
7	1s	64	0.25	140			5.23e-03				2.22e-02	8.19e-04	0.95	LSTM
8	8s	32	0.12	200			9.84e-04				3.65e-02	1.96e-03	0.88	LSTM
9	36s	128	0.48	120			6.82e-03				2.10e-02	7.73e-04	0.95	LSTM
10	37s	16	0.16	140			1.63e-03				1.88e-02	6.30e-04	0.96	LSTM
11	41s	64	0.39	160			8.62e-03				2.59e-02	1.08e-03	0.93	LSTM
12	40s	64	0.39	160			6.72e-03				3.83e-02	2.03e-03	0.88	LSTM
13	41s	64	0.41	160			9.65e-03				2.16e-02	7.96e-04	0.95	LSTM
14	42s	64	0.38	160			3.85e-03				2.09e-02	7.55e-04	0.95	LSTM
15	51s	64	0.39	200			3.27e-03				1.88e-02	6.27e-04	0.96	LSTM
1	2s	32	0.13	140	73.0	4.0	1.11e-03	119.0	18.0	2.0	2.20e-02	8.36e-04	0.95	CNN-LSTM
2	30s	32	0.31	120	17.0	5.0	1.89e-03	49.0	17.0	3.0	1.95e-02	6.81e-04	0.96	CNN-LSTM
3	23s	64	0.17	130	60.0	3.0	3.18e-04	46.0	10.0	3.0	2.86e-02	1.39e-03	0.91	CNN-LSTM
4	45s	128	0.25	110	71.0	3.0	7.45e-03	76.0	58.0	1.0	2.30e-02	8.81e-04	0.95	CNN-LSTM
5	30s	32	0.42	110	113.0	4.0	1.01e-04	38.0	47.0	3.0	4.30e-02	3.08e-03	0.81	CNN-LSTM
6	58s	16	0.41	160	46.0	2.0	1.99e-03	42.0	64.0	3.0	2.15e-02	8.33e-04	0.95	CNN-LSTM
7	51s	32	0.14	160	37.0	2.0	1.78e-03	103.0	16.0	1.0	1.90e-02	6.55e-04	0.96	CNN-LSTM
8	48s	32	0.11	160	34.0	2.0	7.90e-03	66.0	44.0	1.0	3.10e-02	1.47e-03	0.91	CNN-LSTM
9	48s	16	0.19	190	87.0	5.0	9.66e-04	65.0	14.0	1.0	2.18e-02	8.18e-04	0.95	CNN-LSTM
10	19s	16	0.49	190	71.0	5.0	1.96e-04	69.0	21.0	3.0	2.27e-02	8.94e-04	0.94	CNN-LSTM
11	33s	16	0.25	200	110.0	5.0	4.70e-04	16.0	30.0	2.0	1.97e-02	7.05e-04	0.96	CNN-LSTM
12	53s	64	0.20	140	94.0	4.0	7.26e-04	126.0	29.0	2.0	2.17e-02	8.30e-04	0.95	CNN-LSTM
13	47s	128	0.10	180	91.0	4.0	1.07e-03	97.0	9.0	2.0	2.66e-02	1.09e-03	0.93	CNN-LSTM
14	46s	16	0.18	180	89.0	5.0	4.18e-03	128.0	25.0	2.0	2.47e-02	9.60e-04	0.94	CNN-LSTM
15	59s	32	0.33	150	127.0	4.0	8.63e-04	98.0	38.0	1.0	3.84e-02	2.04e-03	0.87	CNN-LSTM
1	32s	128	0.20	200	42.0	4.0	4.57e-04	120.0	13.0	2.0	5.54e-02	4.79e-03	0.70	LSTM-CNN
2	54s	16	0.43	120	37.0	5.0	5.73e-03	80.0	60.0	1.0	3.70e-02	2.13e-03	0.87	LSTM-CNN
3	39s	64	0.45	190	32.0	3.0	1.89e-03	94.0	48.0	3.0	1.14e-01	1.51e-02	0.07	LSTM-CNN
4	27s	64	0.19	200	75.0	4.0	4.09e-03	71.0	18.0	1.0	2.32e-02	8.41e-04	0.95	LSTM-CNN

Table Appendix 1.1 Continued from previous page

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
5	58s	128	0.39	140	32.0	3.0	3.76e-03	105.0	19.0	3.0	1.24e-01	1.76e-02	-0.09	LSTM-CNN
6	4s	16	0.43	130	22.0	5.0	1.12e-04	95.0	19.0	2.0	7.44e-02	7.94e-03	0.51	LSTM-CNN
7	39s	128	0.27	110	125.0	3.0	4.72e-04	84.0	10.0	3.0	1.27e-01	1.91e-02	-0.18	LSTM-CNN
8	7s	64	0.17	170	81.0	2.0	2.01e-03	20.0	55.0	1.0	2.20e-02	7.84e-04	0.95	LSTM-CNN
9	49s	32	0.32	190	122.0	4.0	1.80e-03	53.0	14.0	3.0	8.45e-02	8.26e-03	0.49	LSTM-CNN
10	20s	16	0.48	190	24.0	4.0	8.63e-04	46.0	21.0	2.0	2.55e-02	1.03e-03	0.94	LSTM-CNN
11	10s	64	0.12	160	76.0	2.0	5.41e-03	63.0	34.0	1.0	2.63e-02	1.12e-03	0.93	LSTM-CNN
12	9s	64	0.11	160	77.0	2.0	9.76e-03	59.0	33.0	1.0	2.58e-02	1.05e-03	0.94	LSTM-CNN
13	7s	64	0.10	170	96.0	2.0	9.80e-03	34.0	33.0	1.0	2.19e-02	8.29e-04	0.95	LSTM-CNN
14	4s	64	0.11	170	101.0	2.0	9.78e-03	27.0	33.0	1.0	3.50e-02	1.88e-03	0.88	LSTM-CNN
15	31s	32	0.25	150	56.0	2.0	6.45e-03	38.0	40.0	1.0	2.53e-02	1.12e-03	0.93	LSTM-CNN

Appendix 2. Cardano optuna hyperparameter results

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
1	19s	128	0.44	160			1.55e-04			1.0	2.55e-02	1.23e-03	0.19	CNN
2	21s	64	0.15	150			2.29e-04			3.0	2.08e-02	6.18e-04	0.59	CNN
3	28s	16	0.35	150			7.00e-04			3.0	1.93e-02	5.20e-04	0.66	CNN
4	28s	32	0.11	190			1.08e-04			3.0	1.84e-02	4.74e-04	0.69	CNN
5	21s	64	0.20	200			2.84e-03			3.0	1.63e-02	3.97e-04	0.74	CNN
6	20s	128	0.29	100			4.77e-03			2.0	2.74e-02	9.50e-04	0.37	CNN
7	23s	128	0.11	140			3.62e-03			1.0	2.61e-02	9.09e-04	0.40	CNN
8	23s	32	0.43	120			1.45e-04			2.0	3.19e-02	1.72e-03	-0.14	CNN
9	28s	32	0.50	110			2.13e-03			2.0	1.88e-02	4.79e-04	0.68	CNN
10	29s	16	0.19	130			3.52e-03			3.0	2.37e-02	7.60e-04	0.50	CNN
11	24s	128	0.38	190			4.37e-03			1.0	1.45e-02	3.68e-04	0.76	CNN
12	22s	64	0.43	130			4.59e-03			3.0	1.77e-02	6.07e-04	0.60	CNN
13	18s	16	0.22	190			2.37e-03			3.0	9.09e-03	1.36e-04	0.91	CNN
14	12s	16	0.19	200			1.61e-04			2.0	2.23e-02	6.79e-04	0.55	CNN

15	10s	32	0.25	200			1.46e-04			1.0	2.75e-02	1.14e-03	0.25	CNN
1	51s	32	0.46	190	94.0	2.0	2.53e-04	99.0	28.0	1.0	1.41e-02	3.09e-04	0.71	LSTM
2	44s	32	0.16	170	114.0	2.0	1.01e-03	27.0	13.0	2.0	9.54e-03	1.55e-04	0.85	LSTM
3	40s	128	0.13	160	92.0	2.0	2.69e-03	89.0	52.0	1.0	7.35e-03	1.05e-04	0.90	LSTM
4	32s	32	0.25	160	118.0	4.0	1.53e-03	46.0	41.0	1.0	7.09e-03	9.30e-05	0.91	LSTM
5	9s	16	0.31	180	126.0	3.0	1.02e-04	107.0	60.0	1.0	1.01e-02	1.57e-04	0.85	LSTM
6	58s	16	0.41	180	21.0	2.0	4.50e-04	27.0	33.0	3.0	1.46e-02	3.97e-04	0.63	LSTM
7	53s	64	0.37	150	104.0	4.0	5.33e-04	68.0	12.0	2.0	1.14e-02	2.36e-04	0.78	LSTM
8	35s	32	0.15	180	115.0	2.0	9.85e-03	108.0	41.0	1.0	6.22e-03	7.70e-05	0.93	LSTM
9	49s	64	0.40	100	49.0	4.0	9.76e-03	30.0	50.0	3.0	1.35e-02	2.89e-04	0.73	LSTM
10	56s	128	0.39	150	30.0	3.0	9.60e-04	60.0	21.0	3.0	1.36e-02	3.75e-04	0.65	LSTM
11	6s	32	0.22	120	71.0	5.0	9.71e-03	123.0	42.0	2.0	1.06e-02	1.78e-04	0.83	LSTM
12	6s	32	0.23	200	127.0	5.0	3.11e-03	51.0	41.0	1.0	1.78e-02	3.77e-04	0.65	LSTM
13	4s	32	0.10	200	79.0	5.0	3.89e-03	85.0	46.0	1.0	1.63e-02	3.13e-04	0.71	LSTM
14	58s	32	0.11	200	74.0	3.0	5.07e-03	84.0	53.0	1.0	6.87e-03	9.20e-05	0.91	LSTM
15	38s	32	0.10	200	70.0	3.0	3.96e-03	84.0	60.0	2.0	7.47e-03	1.08e-04	0.90	LSTM
1	10s	16	0.10	100			4.79e-03				1.08e-02	1.65e-04	0.85	CNN-LSTM
2	39s	128	0.39	110			1.25e-04				1.27e-02	3.11e-04	0.71	CNN-LSTM
3	20s	64	0.20	130			7.46e-04				9.25e-03	1.44e-04	0.87	CNN-LSTM
4	38s	32	0.44	170			8.38e-03				1.22e-02	2.00e-04	0.81	CNN-LSTM
5	58s	64	0.20	180			3.51e-04				9.58e-03	1.62e-04	0.85	CNN-LSTM
6	32s	16	0.17	140			1.48e-03				6.05e-03	7.40e-05	0.93	CNN-LSTM
7	26s	16	0.13	110			1.93e-04				9.61e-03	1.70e-04	0.84	CNN-LSTM
8	49s	64	0.10	200			1.15e-04				1.05e-02	2.00e-04	0.81	CNN-LSTM
9	25s	64	0.25	130			9.71e-03				5.94e-03	6.90e-05	0.94	CNN-LSTM
10	29s	128	0.42	110			4.67e-04				1.11e-02	2.19e-04	0.80	CNN-LSTM
11	55s	16	0.32	160			3.09e-03				9.82e-03	1.50e-04	0.86	CNN-LSTM
12	35s	16	0.16	140			2.19e-03				6.49e-03	7.50e-05	0.93	CNN-LSTM
13	35s	16	0.29	100			3.25e-03				6.35e-03	7.90e-05	0.93	CNN-LSTM
14	6s	16	0.50	150			3.00e-03				9.29e-03	1.39e-04	0.87	CNN-LSTM
15	37s	32	0.15	130			1.58e-03				6.09e-03	7.50e-05	0.93	CNN-LSTM
1	51s	64	0.34	140	75.0	2.0	7.11e-03	58.0	53.0	2.0	1.90e-02	6.25e-04	0.42	LSTM-CNN
2	48s	128	0.46	180	84.0	4.0	4.76e-03	128.0	61.0	3.0	2.22e-02	7.83e-04	0.27	LSTM-CNN
3	53s	32	0.41	190	65.0	5.0	9.08e-03	55.0	60.0	3.0	3.57e-02	1.53e-03	-0.43	LSTM-CNN
4	30s	64	0.43	130	115.0	2.0	1.01e-04	113.0	61.0	2.0	2.92e-02	1.12e-03	-0.05	LSTM-CNN
5	28s	64	0.28	120	17.0	2.0	6.62e-03	79.0	25.0	2.0	2.10e-02	8.19e-04	0.23	LSTM-CNN
6	35s	64	0.37	160	79.0	5.0	9.65e-04	54.0	55.0	1.0	1.15e-02	2.31e-04	0.78	LSTM-CNN
7	47s	64	0.39	150	92.0	5.0	2.19e-04	48.0	55.0	1.0	1.40e-02	3.08e-04	0.71	LSTM-CNN

8	50s	128	0.14	120	111.0	4.0	1.05e-03	52.0	57.0	2.0	1.09e-02	2.08e-04	0.81	LSTM-CNN
9	29s	128	0.40	130	83.0	4.0	3.42e-04	45.0	12.0	2.0	2.28e-02	7.49e-04	0.30	LSTM-CNN
10	46s	64	0.31	180	71.0	4.0	5.27e-04	47.0	30.0	1.0	1.88e-02	4.95e-04	0.54	LSTM-CNN
11	53s	16	0.23	100	40.0	3.0	1.54e-03	20.0	45.0	1.0	2.69e-02	1.03e-03	0.04	LSTM-CNN
12	5s	16	0.22	100	43.0	3.0	1.76e-03	16.0	44.0	1.0	1.40e-02	3.60e-04	0.66	LSTM-CNN
13	24s	16	0.21	160	45.0	3.0	2.39e-03	16.0	43.0	1.0	1.34e-02	3.02e-04	0.72	LSTM-CNN
14	26s	16	0.16	160	51.0	3.0	2.78e-03	86.0	41.0	1.0	3.81e-02	1.70e-03	-0.59	LSTM-CNN
15	16s	32	0.50	160	55.0	5.0	2.70e-03	88.0	35.0	1.0	1.45e-02	3.89e-04	0.64	LSTM-CNN

Appendix 3. Ethereum optuna hyperparameter results

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
1	41s	64	0.27	140			2.11e-04			1.0	2.16e-02	8.83e-04	0.95	CNN
2	29s	128	0.14	110			1.61e-04			1.0	2.57e-02	1.21e-03	0.93	CNN
3	32s	128	0.21	130			8.56e-03			1.0	2.83e-02	1.29e-03	0.92	CNN
4	31s	128	0.40	150			1.00e-03			3.0	1.38e-01	2.13e-02	-0.28	CNN
5	32s	128	0.38	150			6.22e-04			2.0	1.01e-01	1.15e-02	0.31	CNN
6	25s	64	0.41	110			1.23e-03			3.0	7.94e-02	7.43e-03	0.55	CNN
7	30s	128	0.47	180			1.74e-03			2.0	2.54e-01	6.93e-02	-3.16	CNN
8	46s	64	0.14	200			5.12e-03			2.0	5.66e-02	4.20e-03	0.75	CNN
9	26s	128	0.34	120			1.85e-03			2.0	1.15e-01	1.49e-02	0.11	CNN
10	31s	128	0.19	150			7.16e-03			3.0	7.19e-02	6.26e-03	0.62	CNN
11	4s	32	0.10	200			3.65e-03			2.0	4.44e-02	2.60e-03	0.84	CNN
12	49s	32	0.11	200			3.91e-03			2.0	4.57e-02	2.87e-03	0.83	CNN
13	47s	32	0.17	180			3.91e-03			2.0	5.51e-02	3.91e-03	0.77	CNN
14	4s	16	0.27	200			3.93e-03			2.0	1.24e-01	1.71e-02	-0.03	CNN
15	46s	32	0.11	180			5.96e-04			1.0	3.53e-02	1.86e-03	0.89	CNN
16	19s	64	0.23	180			5.11e-04			1.0	7.82e-02	7.16e-03	0.57	CNN
17	27s	16	0.15	170			3.15e-04			1.0	5.40e-02	3.75e-03	0.78	CNN
18	26s	32	0.24	170			5.40e-04			1.0	2.76e-02	1.27e-03	0.92	CNN
19	16s	64	0.16	190			1.15e-04			3.0	4.38e-02	2.58e-03	0.85	CNN
20	16s	64	0.32	160			2.17e-03			1.0	1.21e-01	1.64e-02	0.01	CNN
21	18s	32	0.10	190			3.16e-04			3.0	4.66e-02	2.86e-03	0.83	CNN

Table Appendix 3.1 Continued from previous page

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
22	20s	32	0.10	200			5.74e-03			2.0	4.46e-02	2.66e-03	0.84	CNN
23	20s	32	0.13	190			2.65e-03			2.0	6.53e-02	5.24e-03	0.69	CNN
24	20s	32	0.17	190			2.46e-03			2.0	6.10e-02	4.68e-03	0.72	CNN
25	21s	32	0.13	170			8.02e-04			2.0	2.22e-02	9.07e-04	0.95	CNN
26	29s	16	0.20	190			1.26e-03			1.0	2.99e-02	1.41e-03	0.92	CNN
27	29s	16	0.20	180			1.29e-03			1.0	8.67e-02	8.83e-03	0.47	CNN
28	23s	16	0.25	160			9.81e-04			1.0	2.91e-02	1.30e-03	0.92	CNN
29	32s	16	0.20	190			1.49e-03			1.0	8.69e-02	9.10e-03	0.45	CNN
30	24s	16	0.27	140			2.98e-04			1.0	3.55e-02	1.86e-03	0.89	CNN
1	59s	16	0.18	130			2.08e-03				1.88e-02	7.32e-04	0.96	LSTM
2	7s	128	0.39	200			2.90e-04				3.50e-02	2.11e-03	0.87	LSTM
3	44s	16	0.26	200			8.72e-04				2.34e-02	9.88e-04	0.94	LSTM
4	9s	128	0.38	200			1.04e-03				2.83e-02	1.42e-03	0.91	LSTM
5	25s	128	0.37	200			6.19e-04				2.44e-02	1.14e-03	0.93	LSTM
6	51s	64	0.47	180			9.63e-03				2.04e-02	8.06e-04	0.95	LSTM
7	59s	32	0.13	100			5.66e-04				2.54e-02	1.12e-03	0.93	LSTM
8	0s	32	0.25	160			1.60e-04				3.96e-02	2.65e-03	0.84	LSTM
9	45s	16	0.34	130			3.72e-04				2.06e-02	8.49e-04	0.95	LSTM
10	6s	16	0.26	180			1.29e-03				2.48e-02	1.05e-03	0.94	LSTM
11	50s	64	0.10	120			3.47e-03				2.07e-02	8.29e-04	0.95	LSTM
12	46s	16	0.19	150			2.14e-03				1.91e-02	7.46e-04	0.96	LSTM
13	36s	16	0.17	140			2.60e-03				2.11e-02	8.68e-04	0.95	LSTM
14	29s	16	0.19	160			2.94e-03				1.97e-02	7.63e-04	0.95	LSTM
15	4s	16	0.20	110			6.64e-03				1.92e-02	7.60e-04	0.95	LSTM
16	44s	16	0.14	140			1.88e-03				1.87e-02	7.31e-04	0.96	LSTM
17	38s	16	0.14	130			5.32e-03				1.81e-02	6.88e-04	0.96	LSTM
18	14s	32	0.23	140			1.86e-03				1.82e-02	7.03e-04	0.96	LSTM
19	48s	64	0.31	120			1.41e-03				2.25e-02	9.71e-04	0.94	LSTM
20	16s	16	0.14	100			4.97e-03				1.81e-02	6.93e-04	0.96	LSTM
21	22s	16	0.44	100			3.70e-03				1.99e-02	8.15e-04	0.95	LSTM
22	22s	16	0.15	110			4.08e-03				1.82e-02	6.92e-04	0.96	LSTM
23	46s	16	0.10	150			7.79e-03				2.19e-02	8.76e-04	0.95	LSTM
24	55s	16	0.11	160			9.29e-03				1.89e-02	7.16e-04	0.96	LSTM

Table Appendix 3.1 Continued from previous page

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
25	2s	16	0.12	170			8.80e-03				4.10e-02	2.39e-03	0.86	LSTM
26	2s	16	0.10	170			9.41e-03				2.41e-02	1.07e-03	0.94	LSTM
27	49s	128	0.22	180			7.26e-03				1.84e-02	7.06e-04	0.96	LSTM
28	25s	32	0.12	160			6.42e-03				1.83e-02	7.04e-04	0.96	LSTM
29	58s	64	0.16	170			9.66e-03				2.49e-02	1.04e-03	0.94	LSTM
30	38s	16	0.28	150			4.80e-03				3.74e-02	1.93e-03	0.88	LSTM
1	26s	128	0.45	150	47.0	5.0	7.90e-03	105.0	42.0	1.0	2.11e-02	8.97e-04	0.95	CNN-LSTM
2	32s	64	0.15	200	97.0	2.0	1.45e-03	35.0	17.0	2.0	1.98e-02	7.62e-04	0.95	CNN-LSTM
3	24s	64	0.34	140	34.0	4.0	1.97e-04	121.0	61.0	3.0	2.94e-02	1.55e-03	0.91	CNN-LSTM
4	36s	64	0.30	190	101.0	2.0	4.02e-03	100.0	27.0	3.0	1.96e-02	7.51e-04	0.95	CNN-LSTM
5	7s	128	0.47	190	96.0	3.0	3.82e-04	106.0	28.0	3.0	3.55e-02	2.11e-03	0.87	CNN-LSTM
6	6s	128	0.15	110	35.0	2.0	2.05e-04	88.0	20.0	1.0	3.41e-02	2.02e-03	0.88	CNN-LSTM
7	55s	16	0.21	150	82.0	3.0	7.58e-03	46.0	30.0	2.0	2.48e-02	1.12e-03	0.93	CNN-LSTM
8	30s	16	0.42	140	101.0	3.0	8.71e-04	55.0	56.0	1.0	1.97e-02	8.07e-04	0.95	CNN-LSTM
9	38s	32	0.12	190	16.0	5.0	7.23e-03	20.0	37.0	1.0	4.44e-02	2.69e-03	0.84	CNN-LSTM
10	49s	128	0.49	120	37.0	3.0	1.62e-03	101.0	22.0	2.0	3.16e-02	1.69e-03	0.90	CNN-LSTM
11	2s	64	0.22	170	128.0	2.0	1.61e-03	17.0	9.0	2.0	2.00e-02	8.28e-04	0.95	CNN-LSTM
12	56s	16	0.21	170	74.0	4.0	3.05e-03	48.0	9.0	2.0	2.51e-02	1.14e-03	0.93	CNN-LSTM
13	53s	16	0.21	170	71.0	4.0	2.75e-03	48.0	8.0	2.0	2.86e-02	1.26e-03	0.92	CNN-LSTM
14	34s	32	0.16	200	66.0	4.0	8.85e-04	34.0	16.0	2.0	1.89e-02	7.47e-04	0.96	CNN-LSTM
15	18s	16	0.27	170	122.0	4.0	3.11e-03	66.0	14.0	2.0	3.78e-02	2.14e-03	0.87	CNN-LSTM
16	52s	64	0.11	180	86.0	5.0	5.82e-04	72.0	44.0	2.0	1.92e-02	7.58e-04	0.95	CNN-LSTM
17	8s	16	0.27	200	61.0	2.0	1.55e-03	34.0	12.0	3.0	1.94e-02	7.71e-04	0.95	CNN-LSTM
18	37s	64	0.17	160	113.0	3.0	1.01e-04	33.0	21.0	2.0	3.89e-02	2.57e-03	0.85	CNN-LSTM
19	38s	32	0.37	130	80.0	4.0	4.58e-03	56.0	51.0	3.0	2.18e-02	8.80e-04	0.95	CNN-LSTM
20	46s	64	0.18	100	57.0	5.0	2.10e-03	72.0	34.0	1.0	2.09e-02	8.53e-04	0.95	CNN-LSTM
21	16s	16	0.26	180	112.0	2.0	6.24e-04	28.0	17.0	2.0	2.34e-02	9.87e-04	0.94	CNN-LSTM
22	17s	16	0.23	170	78.0	4.0	2.56e-03	47.0	8.0	2.0	3.02e-02	1.38e-03	0.92	CNN-LSTM
23	12s	16	0.14	160	88.0	4.0	4.68e-03	43.0	8.0	2.0	1.98e-02	7.67e-04	0.95	CNN-LSTM
24	17s	16	0.13	160	94.0	4.0	4.66e-03	60.0	13.0	2.0	2.70e-02	1.29e-03	0.92	CNN-LSTM
25	17s	16	0.13	160	89.0	3.0	5.34e-03	61.0	24.0	2.0	2.96e-02	1.46e-03	0.91	CNN-LSTM
26	18s	16	0.13	160	90.0	3.0	5.51e-03	63.0	24.0	2.0	1.97e-02	7.63e-04	0.95	CNN-LSTM
27	14s	16	0.10	140	109.0	4.0	5.96e-03	85.0	13.0	1.0	2.53e-02	1.13e-03	0.93	CNN-LSTM

Table Appendix 3.1 Continued from previous page

Trial	Duration	Batch Size	Dropout Rate	Epochs	Filters	Kernel Size	Learning Rate	LSTM Units1	LSTM Units2	Pool Size	MAE	MSE	R2 Score	Model
28	12s	16	0.11	140	112.0	3.0	8.74e-03	82.0	32.0	1.0	2.15e-02	8.45e-04	0.95	CNN-LSTM
29	4s	16	0.19	130	107.0	4.0	5.96e-03	83.0	26.0	1.0	2.18e-02	8.37e-04	0.95	CNN-LSTM
30	54s	16	0.19	120	106.0	5.0	8.55e-03	79.0	39.0	1.0	2.15e-02	8.75e-04	0.95	CNN-LSTM
1	13s	128	0.49	140	64.0	5.0	4.06e-03	98.0	35.0	3.0	8.41e-02	8.23e-03	0.51	LSTM-CNN
2	43s	32	0.44	120	127.0	3.0	9.14e-03	44.0	60.0	2.0	3.48e-02	2.05e-03	0.88	LSTM-CNN
3	48s	128	0.35	150	85.0	2.0	1.36e-03	51.0	62.0	3.0	1.37e-01	2.06e-02	-0.24	LSTM-CNN
4	50s	16	0.23	190	106.0	4.0	2.43e-04	69.0	10.0	3.0	1.21e-01	1.61e-02	0.03	LSTM-CNN
5	1s	32	0.23	100	106.0	3.0	4.03e-04	59.0	53.0	1.0	2.41e-02	1.12e-03	0.93	LSTM-CNN
6	21s	16	0.42	120	36.0	4.0	4.84e-03	117.0	61.0	1.0	1.96e-02	7.80e-04	0.95	LSTM-CNN
7	14s	16	0.15	160	102.0	4.0	8.80e-04	121.0	59.0	3.0	2.04e-02	8.14e-04	0.95	LSTM-CNN
8	49s	16	0.12	200	107.0	5.0	5.57e-04	18.0	9.0	1.0	2.31e-02	9.69e-04	0.94	LSTM-CNN
9	38s	32	0.23	170	122.0	4.0	3.64e-03	67.0	19.0	3.0	2.52e-02	1.14e-03	0.93	LSTM-CNN
10	53s	64	0.38	150	53.0	2.0	3.68e-03	111.0	60.0	2.0	5.60e-02	4.01e-03	0.76	LSTM-CNN
11	31s	32	0.28	180	17.0	5.0	1.26e-04	87.0	24.0	2.0	8.29e-02	8.97e-03	0.46	LSTM-CNN
12	48s	64	0.12	170	128.0	4.0	1.40e-03	83.0	44.0	3.0	3.32e-02	1.70e-03	0.90	LSTM-CNN
13	42s	64	0.20	170	128.0	4.0	1.86e-03	78.0	43.0	3.0	2.39e-02	1.02e-03	0.94	LSTM-CNN
14	14s	64	0.17	170	84.0	3.0	2.61e-03	32.0	25.0	2.0	4.11e-02	2.32e-03	0.86	LSTM-CNN
15	41s	32	0.11	190	120.0	4.0	9.69e-03	93.0	44.0	3.0	3.01e-02	1.37e-03	0.92	LSTM-CNN
16	14s	64	0.27	170	88.0	3.0	9.04e-04	70.0	21.0	2.0	7.63e-02	7.01e-03	0.58	LSTM-CNN
17	37s	32	0.31	130	117.0	5.0	1.90e-03	106.0	33.0	3.0	2.76e-02	1.24e-03	0.93	LSTM-CNN
18	20s	64	0.17	160	95.0	4.0	5.70e-03	81.0	46.0	3.0	1.92e-02	7.61e-04	0.95	LSTM-CNN
19	40s	128	0.18	140	69.0	3.0	6.23e-03	59.0	15.0	3.0	5.38e-02	3.93e-03	0.76	LSTM-CNN
20	7s	64	0.25	160	90.0	5.0	2.66e-03	44.0	50.0	2.0	2.16e-02	8.77e-04	0.95	LSTM-CNN
21	8s	32	0.32	200	97.0	4.0	6.41e-03	66.0	31.0	2.0	2.02e-02	8.25e-04	0.95	LSTM-CNN
22	10s	32	0.32	200	98.0	4.0	6.08e-03	66.0	30.0	2.0	7.58e-02	6.56e-03	0.61	LSTM-CNN
23	16s	32	0.35	190	79.0	4.0	7.00e-03	85.0	39.0	3.0	4.14e-02	2.26e-03	0.86	LSTM-CNN
24	17s	32	0.36	180	57.0	4.0	3.44e-03	98.0	39.0	3.0	4.35e-02	2.54e-03	0.85	LSTM-CNN
25	11s	32	0.16	180	80.0	3.0	9.83e-03	85.0	53.0	3.0	3.54e-02	1.93e-03	0.88	LSTM-CNN
26	48s	32	0.21	160	75.0	4.0	5.55e-03	77.0	39.0	3.0	2.60e-02	1.13e-03	0.93	LSTM-CNN
27	54s	64	0.26	190	116.0	5.0	2.78e-03	128.0	19.0	3.0	5.95e-02	4.36e-03	0.74	LSTM-CNN
28	18s	128	0.40	180	52.0	4.0	7.40e-03	92.0	49.0	3.0	4.21e-02	2.52e-03	0.85	LSTM-CNN
29	36s	32	0.35	160	43.0	3.0	4.93e-03	57.0	27.0	3.0	5.30e-02	3.92e-03	0.76	LSTM-CNN
30	3s	128	0.47	140	66.0	5.0	3.76e-03	104.0	37.0	3.0	4.26e-02	2.59e-03	0.84	LSTM-CNN

Appendix 4. Data processing pipeline

```
class DataProcessor:
    def __init__(self, config: ModelConfig):
        self.config = config
        self.mean: Optional[float] = None
        self.std: Optional[float] = None
        self.df: Optional[pd.DataFrame] = None
        self.logger = structlog.get_logger(__name__)

    def create_windows(self, data: np.ndarray, window_size: int)
    -> Tuple[np.ndarray, np.ndarray]:
        X, y = [], []
        for i in range(len(data) - window_size):
            X.append(data[i:i + window_size])
            y.append(data[i + window_size])
        return np.array(X), np.array(y)

    def fetch_and_prepare_data(self, ticker_symbol: str,
    start_date: str, end_date: str) -> Tuple[
        Tuple[np.ndarray, np.ndarray], Tuple[np.ndarray, np.ndarray],
        np.ndarray
    ]:
        self.logger.info("fetching_data", ticker=ticker_symbol,
        start_date=start_date, end_date=end_date)

        try:
            self.df = yf.Ticker(ticker_symbol).history(start=start_date,
            end=end_date)
            if self.df.empty:
                raise ValueError(f"No data found for
                ticker {ticker_symbol}")

            close_prices = self.df['Close'].values
            if len(close_prices) < self.config.lookback:
                raise ValueError(f"Insufficient data points")

            self.mean = close_prices.mean()
            self.std = close_prices.std()
            normalized_prices = (close_prices - self.mean) / self.std

            train_size = int(len(normalized_prices) *
            self.config.train_test_split)
            train_data = normalized_prices[:train_size]
            test_data = normalized_prices[train_size:]

            X_train, y_train=self.create_windows(train_data,
            self.config.lookback)
            X_test, y_test=self.create_windows(test_data,
```

```
            self.config.lookback)

            return (X_train, y_train), (X_test, y_test), close_prices
        except Exception as e:
            self.logger.error("data_preparation_failed", error=str(e))
            raise
```

Listing 12. DataProcessor

Appendix 5. Server Index App.py File

```
from flask import Flask, jsonify, request
from flask_socketio import SocketIO
import structlog
from datetime import datetime, timedelta
import redis
from flask_cors import CORS
import os

from validations import TrainingRequest, PredictionRequest
from ml_app.config import ModelConfig
from ml_app.predictor import CryptoPredictor
from ml_app.websocket.manager import WebSocketManager
from utils.helpers import fetch_historical_data, make_predictions
from utils.logger import setup_logging, get_logger,\
    RequestIdContext

app = Flask(__name__)
CORS(app, resources={
    r"/*": {
        "origins": os.getenv('CORS_ORIGINS', '*'),
        "methods": ["GET", "POST", "PUT", "DELETE"],
        "allow_headers": ["Content-Type", "Authorization"]
    }
})

redis_client = redis.Redis(host='redis', port=6379, db=0)

socketio = SocketIO(
    app,
    cors_allowed_origins=os.getenv('CORS_ORIGINS', '*'),
    message_queue='redis://redis:6379/0',
    async_mode='eventlet'
)

websocket_manager = WebSocketManager(socketio)

setup_logging(log_level=os.getenv("LOG_LEVEL", "INFO"))
```

```
logger = get_logger(__name__)

@app.route('/train', methods=['POST'])
> def train_model():...

@app.route('/predict', methods=['POST'])
>def predict_model():...

@app.route('/health', methods=['GET'])
>def health_check():...

@app.errorhandler(Exception)
def handle_error(error):
    logger.error("unexpected_error",
                 error=str(error),
                 error_type=type(error).__name__,
                 exc_info=True)
    return jsonify({
        'error': 'Internal server error',
        'message': str(error)
    }), 500

@app.after_request
def after_request(response):
    response.headers.add('Access-Control-Allow-Origin',
                         os.getenv('CORS_ORIGINS', '*'))
    response.headers.add('Access-Control-Allow-Headers',
                         'Content-Type,Authorization')
    response.headers.add('Access-Control-Allow-Methods',
                         'GET,PUT,POST,DELETE,OPTIONS')
    response.headers.add('Access-Control-Allow-Credentials',
                         'true')
    return response

if __name__ == '__main__':
    logger.info("starting_server")
    socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```

Listing 13. File: app.py

Appendix 6. Optuna Optimization Code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D,\
    Flatten, Dense, LSTM, Dropout, Reshape, LeakyReLU
from tensorflow.keras.optimizers import Adam

def build_cnn(params, input_shape):
```

```
7       """Build and compile a CNN model with given parameters."""
8       if hasattr(params, 'suggest_int'):
9           filters1 = params.suggest_int('filters1', 16, 128)
10          filters2 = params.suggest_int('filters2', 16, 128)
11          kernel_size1 = params.suggest_int('kernel_size1', 2, 5)
12          kernel_size2 = params.suggest_int('kernel_size2', 2, 5)
13          pool_size = params.suggest_int('pool_size', 1, 3)
14          dropout_rate = params.suggest_float('dropout_rate', 0.1, 0.5)
15          learning_rate = params.suggest_float('learning_rate', 1e-4,
16          1e-2, log=True)
17      else:
18          filters1 = int(params['filters1'])
19          filters2 = int(params['filters2'])
20          kernel_size1 = int(params['kernel_size1'])
21          kernel_size2 = int(params['kernel_size2'])
22          pool_size = int(params['pool_size'])
23          dropout_rate = float(params['dropout_rate'])
24          learning_rate = float(params['learning_rate'])
25
26      model = Sequential([
27          Conv1D(filters=filters1, kernel_size=kernel_size1,
28          activation='relu', input_shape=input_shape, padding='same'),
29          MaxPooling1D(pool_size=pool_size, padding='same'),
30          Dropout(dropout_rate),
31          Conv1D(filters=filters2, kernel_size=kernel_size2,
32          activation='relu', padding='same'),
33          MaxPooling1D(pool_size=pool_size, padding='same'),
34          Flatten(),
35          Dense(units=1),
36          LeakyReLU(alpha=0.1)
37      ])
38      model.compile(optimizer=Adam(learning_rate=learning_rate),
39      loss='mean_squared_error')
40      return model
41
42  def build_lstm(params, input_shape):
43      """Build and compile an LSTM model with given parameters."""
44      if hasattr(params, 'suggest_int'):
45          units1 = params.suggest_int('units1', 16, 128)
46          units2 = params.suggest_int('units2', 8, 64)
47          dropout_rate = params.suggest_float('dropout_rate',
48          0.1, 0.5)
49          learning_rate = params.suggest_float('learning_rate',
50          1e-4, 1e-2, log=True)
51      else:
52          units1 = int(params['units1'])
53          units2 = int(params['units2'])
54          dropout_rate = float(params['dropout_rate'])
55          learning_rate = float(params['learning_rate'])
56
```

```
57      model = Sequential([
58          LSTM(units=units1, return_sequences=True,
59          input_shape=input_shape),
60          Dropout(dropout_rate),
61          LSTM(units=units2),
62          Dropout(dropout_rate),
63          Dense(units=1),
64          LeakyReLU(alpha=0.1)
65      ])
66      model.compile(optimizer=Adam(learning_rate=learning_rate),
67      loss='mean_squared_error')
68      return model
69
70  def build_cnn_lstm(params, input_shape):
71      """Build and compile a CNN-LSTM model with given parameters."""
72      if hasattr(params, 'suggest_int'):
73          filters = params.suggest_int('filters', 16, 128)
74          kernel_size = params.suggest_int('kernel_size', 2, 5)
75          pool_size = params.suggest_int('pool_size', 1, 3)
76          lstm_units1 = params.suggest_int('lstm_units1', 16, 128)
77          lstm_units2 = params.suggest_int('lstm_units2', 8, 64)
78          dropout_rate = params.suggest_float('dropout_rate', 0.1,
79          0.5)
80          learning_rate = params.suggest_float('learning_rate', 1e-4,
81          1e-2, log=True)
82      else:
83          filters = int(params['filters'])
84          kernel_size = int(params['kernel_size'])
85          pool_size = int(params['pool_size'])
86          lstm_units1 = int(params['lstm_units1'])
87          lstm_units2 = int(params['lstm_units2'])
88          dropout_rate = float(params['dropout_rate'])
89          learning_rate = float(params['learning_rate'])
90
91      model = Sequential([
92          Conv1D(filters=filters, kernel_size=kernel_size,
93          activation='relu',
94          input_shape=input_shape, padding='same'),
95          MaxPooling1D(pool_size=pool_size, padding='same'),
96          Dropout(dropout_rate),
97          Reshape((-1, filters)),
98          LSTM(units=lstm_units1, return_sequences=True),
99          Dropout(dropout_rate),
100         LSTM(units=lstm_units2),
101         Dropout(dropout_rate),
102         Dense(units=1),
103         LeakyReLU(alpha=0.1)
104     ])
105     model.compile(optimizer=Adam(learning_rate=learning_rate),
106     loss='mean_squared_error')
```

```
    return model

def build_lstm_cnn(params, input_shape):
    """Build and compile an LSTM-CNN model with given parameters."""
    if hasattr(params, 'suggest_int'):
        lstm_units1 = params.suggest_int('lstm_units1', 16, 128)
        lstm_units2 = params.suggest_int('lstm_units2', 8, 64)
        filters = params.suggest_int('filters', 16, 128)
        kernel_size = params.suggest_int('kernel_size', 2, 5)
        pool_size = params.suggest_int('pool_size', 1, 3)
        dropout_rate = params.suggest_float('dropout_rate', 0.1, 0.5)
        learning_rate = params.suggest_float('learning_rate',
        1e-4, 1e-2, log=True)
    else:
        lstm_units1 = int(params['lstm_units1'])
        lstm_units2 = int(params['lstm_units2'])
        filters = int(params['filters'])
        kernel_size = int(params['kernel_size'])
        pool_size = int(params['pool_size'])
        dropout_rate = float(params['dropout_rate'])
        learning_rate = float(params['learning_rate'])

    model = Sequential([
        LSTM(units=lstm_units1, return_sequences=True,
        input_shape=input_shape),
        Dropout(dropout_rate),
        LSTM(units=lstm_units2, return_sequences=True),
        Dropout(dropout_rate),
        Reshape((-1, lstm_units2)),
        Conv1D(filters=filters, kernel_size=kernel_size,
        activation='relu', padding='same'),
        MaxPooling1D(pool_size=pool_size, padding='same'),
        Dropout(dropout_rate),
        Flatten(),
        Dense(units=1),
        LeakyReLU(alpha=0.1)
    ])
    model.compile(optimizer=Adam(learning_rate=learning_rate),
    loss='mean_squared_error')
    return model
```

Listing 14. Model Building Functions

Appendix 7. Client Side Custom Websocket Hook

```
 "use client";
import { useState, useCallback, useEffect } from "react";
import { io, Socket } from "socket.io-client";
```

```
4  import { ModelStages } from "@/constants/model-stages.constant";
5  import { TrainingUpdateResponse } from "@/types/training-update";
6  import { EvaluatingUpdateResponse } from "@/types/evaluating-update";
7  import { VisualizingUpdateResponse } from "@/types/visualizing-update";
8
9  const SOCKET_URL = process.env.NEXT_PUBLIC_WS_URL||"ws://localhost:5000";
10
11 export function useModelSocket() {
12   const [isConnected, setIsConnected] = useState<boolean>(false);
13   const [error, setError] = useState<Error | null>(null);
14   const [progressData, setProgressData] = useState<{
15     training?: TrainingUpdateResponse;
16     evaluating?: EvaluatingUpdateResponse;
17     visualizing?: VisualizingUpdateResponse;
18   }>({});
19
20   const [currentStage, setCurrentStage] = useState<ModelStages>(
21     ModelStages.PARAMETERS
22   );
23   const [currentProgress, setCurrentProgress] = useState<number>(0);
24
25   const handleTrainingUpdate = useCallback((data:
26   TrainingUpdateResponse)
27   => {
28     setProgressData((prev) => ({ ...prev, training: data }));
29     setCurrentStage(ModelStages.TRAINING);
30     setCurrentProgress(data.data.progress);
31   }, []);
32
33   const handleEvaluationUpdate = useCallback(
34     (data: EvaluatingUpdateResponse) => {
35       setProgressData((prev) => ({ ...prev, evaluating: data }));
36       setCurrentStage(ModelStages.EVALUATING);
37       setCurrentProgress(data.data.progress);
38     },
39     []
40   );
41
42   const handleVisualizationUpdate = useCallback(
43     (data: VisualizingUpdateResponse) => {
44       setProgressData((prev) => ({ ...prev, visualizing: data }));
45       setCurrentStage(ModelStages.VISUALIZING);
46       setCurrentProgress(data.data.progress);
47       if (data.data.progress === 100)
48       setCurrentStage(ModelStages.PREDICTING);
49     },
50     []
51   );
52
53   useEffect(() => {
```

```
54      const socket: Socket = io(SOCKET_URL, {
55        transports: ["websocket"],
56      });
57
58      socket.on("connect", () => {
59        setIsConnected(true);
60        console.log("Connected to WebSocket server");
61      });
62      socket.on("connect_error", (err) => {
63        setError(new Error(`Connection failed: ${err.message}`));
64        console.error(`Connection failed: ${err.message}`);
65      });
66      socket.on("disconnect", () => setIsConnected(false));
67      socket.on("training_update", handleTrainingUpdate);
68      socket.on("evaluating_update", handleEvaluationUpdate);
69      socket.on("visualizing_update", handleVisualizationUpdate);
70
71      return () => {
72        socket.disconnect();
73        socket.off("training_update", handleTrainingUpdate);
74        socket.off("evaluating_update", handleEvaluationUpdate);
75        socket.off("visualizing_update", handleVisualizationUpdate);
76      };
77    }, [handleTrainingUpdate, handleEvaluationUpdate,
78    handleVisualizationUpdate]);
79
80    const resetState = useCallback(() => {
81      setProgressData({});
82      setError(null);
83      setCurrentStage(ModelStages.PARAMETERS);
84      setCurrentProgress(0);
85    }, []);
86
87    const isStageComplete = useCallback(
88      (stage: ModelStages) => {
89        const stageOrder = Object.values(ModelStages);
90        const currentIndex = stageOrder.indexOf(currentStage);
91        const stageIndex = stageOrder.indexOf(stage);
92        return currentIndex > stageIndex;
93      },
94      [currentStage]
95    );
96
97    return {
98      isConnected,
99      error,
100     progressData,
101     currentStage,
102     currentProgress,
103     setCurrentStage,
```

```
104       resetState,
105       isStageComplete,
106     };
107 }
```

Listing 15. Custom React Hook for Websocket