Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра математики

Бакалаврська робота

освітній ступінь – бакалавр

на тему:

“Developing a Hybrid AI model for Financial Market Prediction”

Виконав: студент 4-го року навчання
Освітньо-наукової програми
“Прикладна математика”, 113
Войтішин Микита Євгенович
Керівник Кузьменко Д. О.
старший викладач

Рецензент_____

Бакалаврська робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 2025 р.

Київ – 2025

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Abstract

Over the recent years, financial time series modeling has presented a significant challenge due to market stochasticity and volatility. The stock market is influenced not only by market data such as price and volume but also by a wide range of additional external factors, including macroeconomic indicators, seasonality, fundamentals, and market sentiment.

The increasing availability of diverse financial data, combined with the rapid advances in artificial intelligence (AI), has opened up new possibilities for analyzing and understanding how stock markets behave. These technologies have the potential to capture more complex nonlinear patterns that traditional statistical and machine learning models often fail to detect.

This research examines how combining various model architectures and feature sets with domain - specific knowledge from the financial sector can enhance uncertainty quantification, a crucial aspect of making informed decisions and investments in financial markets.

1.2 Background and Motivation

While stock prices appear to evolve with an element of randomness, they are not purely random. Financial markets follow stochastic processes, meaning that although there is inherent uncertainty, patterns, and trends are also embedded within the noise.

Traditional statistical models have provided valuable insights into market behavior but often fall short in capturing nonlinear interactions, regime changes, and latent dependencies. On the other hand, modern machine learning and deep learning models - especially when applied to multimodal data sources - have shown promising results in extracting hidden signals from large, noisy datasets.

Of particular interest is the integration of textual sentiment analysis, which reflects the behavioral dimension of market participants. Investor sentiment, as expressed through news articles, earnings calls, analyst commentary, or social media, can have a substantial impact on stock prices, often in ways that are not directly observable in numerical data.

1.3 Research Objectives

This research aims to deepen the understanding of stock market behavior by conducting an ablation study across a range of modeling techniques and input feature types. The objective is to systematically assess the contributions of different data features and model architectures, including traditional statistical methods, machine learning algorithms, neural networks, and probabilistic frameworks.

A central component of this work is the integration of diverse inputs - price data, technical indicators, macroeconomic variables, and sentiment indicators - into a unified modeling pipeline. The task is formulated as a regression problem, where the target variable is the daily close price return, computed as the percentage change in the closing price from the previous trading day. The primary metric for evaluating model performance is the Mean Squared Error (MSE).

Two key hypotheses guide this study:

1. **Quantifying uncertainty is essential in financial modeling.**
   Estimating the unknown enhances model robustness and facilitates more informed decision-making under uncertainty. Bayesian methods provide a framework to generate probabilistic forecasts that reflect varying levels of model confidence.

2. **Sentiment analysis captures behavioral signals that are often hidden in numerical data.**
   To explore this, sentiment scores will be retrospectively extracted and

quantified from archived textual sources using natural language processing techniques, capturing the overall mood reflected in the content.

2. LITERATURE REVIEW

Green, Gudmundsson, and Belcher (2022)[1] introduced Borch, a deep probabilistic programming language built on PyTorch, designed to integrate Bayesian reasoning with deep neural networks. Unlike traditional models that rely on point estimates, Borch treats model parameters as probability distributions, allowing it to explicitly model uncertainty in both data and model structure. Reflecting on the paper, one of the key ideas in modeling any system is recognizing what we don't know (Kläs and Vollmer, 2018). While this might seem difficult, it's actually straightforward - we don't need to understand the reason behind our uncertainty, only to measure how uncertain we are. To do this effectively, it helps to categorize different types of uncertainty so we can better identify and understand the gaps in our knowledge.

Hauzenberger, Huber (2022) [2] presented a flexible Bayesian Neural Network (BNN) framework designed to model complex nonlinearities and time variation in macroeconomic and financial datasets. Their architecture incorporates stochastic volatility and shrinkage priors to handle high-dimensional inputs, enabling improved estimation and forecasting. Through experiments, the authors demonstrate that the inclusion of macroeconomic variables and modeling components like activation functions and volatility priors significantly enhances predictive accuracy. This research underscores the value of macro-level features in time series forecasting and supports their integration in advanced neural architectures. In line with these findings, our ablation study also incorporates macroeconomic indicators to evaluate their influence on financial forecasting

performance across different model classes, highlighting their relevance for robust and interpretable predictions.

Recent research by Kirtac and Germano (2024)[3] highlighted the effectiveness of large language models (LLMs) in financial sentiment analysis and stock return prediction. In their study, models such as OPT, BERT, and FinBERT were applied to over 965,000 financial news articles, with the OPT model achieving the highest predictive accuracy (74.4%) and the strongest trading performance (Sharpe ratio of 3.05). While their FinBERT model was pre-trained on formal financial texts, our research builds on this by employing a FinBERT variant fine-tuned on Twitter data. This allows us to capture more immediate and sentiment-rich signals from social media, which can reflect market reactions in real time. By integrating this tuned FinBERT model in sentiment extractor pipeline, our study explores whether social media driven approaches in predicting stock returns are able to improve forecasting accuracy.

3. PRELIMINARIES

A solid understanding of financial analysis types, data processing pipelines, and the mathematical foundations of predictive models is essential for building robust forecasting systems.

3.1 Financial Market Analyses

3.1.1 Technical Analysis

Technical analysis involves the study of historical market data, including price and volume, to identify patterns and trends that may signal future price movements. In this work, a range of technical indicators, including SMA, EMA, RSI, ROC, and Bollinger Bands, were utilized to capture momentum and volatility.

3.1.2 Sentiment Analysis

Sentiment analysis leverages natural language processing (NLP) techniques to quantify the emotional tone or opinion expressed in textual data related to financial markets. In this study, sentiment scores were derived from Twitter financial text sources and incorporated as predictive features. These scores provide insights into market psychology, capturing reactions to news, earnings, or macroeconomic developments, and offer a complementary perspective to purely numerical indicators.

3.1.3 Anomaly Detection Analysis

Anomaly detection analysis in financial time series aims to uncover rare or extreme events that significantly diverge from typical market behavior. This analysis helps identify periods of irregular volatility, unexpected price shifts, or external movements that may not be captured by technical analysis.

3.1.4 Macroeconomic Analysis

Macroeconomic analysis integrates broad economic indicators to contextualize asset behavior within the larger economy. These indicators capture the underlying economic environment and help inform the model about structural forces that influence long-term price trends and market cycles.

3.2 Feature Engineering

To enhance model performance and capture complex market dynamics, a diverse set of engineered features was developed. These include return - based target transformations, rolling statistical indicators, technical analysis metrics, anomaly scores, macroeconomic variables, and sentiment-derived signals.

3.2.1 Output Target Transformations

The main prediction target is the Close Price Return, defined as:

$$Close\ Price\ Return_t = \frac{Close\ Price_t - Close\ Price_{t-1}}{Close\ Price_t}$$

3.2.2 Rolling Features

Moving averages and rolling statistics help smooth out noise and highlight trends over various time windows. In our work, we implemented the majority of rolling features using window sizes of 5, 10, 20, 50, 100, and 200 days.

- Simple Moving Average

$$SMA_t(n) = \frac{1}{n}\sum_{i=0}^{n-1} P_{t-i},\ where\ n - window\ size$$

- Exponential Moving Average

$$EMA_t(n) = P_t * \alpha + EMA_{t-1}(n) * (1 - \alpha),\ where\ \alpha = \frac{2}{n+1}$$

Fig 1: Demo Streamlit EMA and SMA Visualisation

3.2.3 Technical Features

Commonly used technical indicators provide insight into price momentum, volatility, and overbought/oversold conditions:

- Bollinger Bands for $window\ size\ n$

$$Mean\ Bollinger\ Band_t = SMA_t(n)$$
$$Upper\ Bollinger\ Band_t = SMA_t(n) + 2 * \sigma_t(n)$$
$$Lower\ Bollinger\ Band_t = SMA_t(n) - 2 * \sigma_t(n)$$

Fig 2: Demo Streamlit Bollinger Bands Visualisation

- Relative strength Index (RSI)

$$RS_t(n) = \frac{Average\ Gain\ over\ n\ days}{Average\ Loss\ over\ n\ days}$$

$$RSI_t(n) = 100 - \frac{100}{1 + RS_t(n)}$$

3.2.4 Volatility and Momentum Features

- Intraday Volatility

$$Intraday\ Volatility_t = \frac{High_t - Low_t}{Open_t}$$

- Rate of Change (ROC)

$$ROC_t(n) = \frac{Close_t - Close_{t-n}}{Close_{t-n}} \times 100$$

- Daily Volume Change

$$Volume\ Change_t = \frac{Volume_t - Volume_{t-1}}{Volume_{t-1}}$$

3.2.5 Anomaly Detection Scores

Continuous anomaly scores were computed using the Isolation Forest algorithm, which isolates outliers by randomly partitioning data:

$$A_t = IsolationForestScore(Feature\ vector_t) \in [-1, 1]$$

Discrete anomaly scores detect unusual market behavior using interquartile range (IQR) filtering:

$$Extreme_t = \{1\ if\ |Close\ Return_t - Median| > 1.5 \times IQR,\ 0\ otherwise\}$$

3.3 Probabilistic Financial Modeling

3.3.1 Random Walk Simulation

Random walk simulation offers a baseline for understanding stock price movements under uncertainty and noise. It assumes that price changes are independent and identically distributed, reflecting the efficient market hypothesis where future movements cannot be predicted from past data. While overly simplistic, it serves as a valuable benchmark to compare the performance of more

sophisticated forecasting models. Models that fail to outperform a random walk are generally considered ineffective for practical financial prediction tasks.

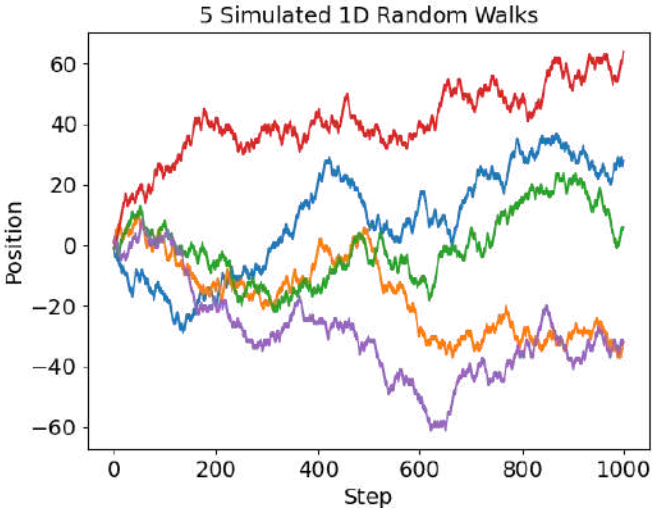

Fig 3: 5 Simulated Random Walks

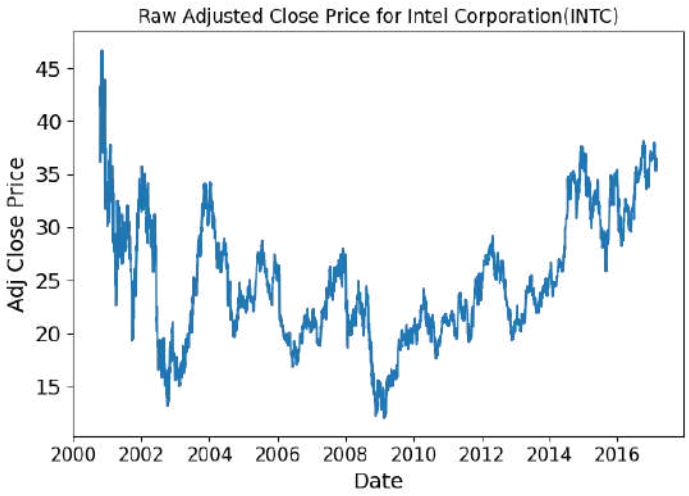

Fig 4: Raw Close Price for INTC

There are noticeable similarities between the random walk simulation and the distribution of actual close price returns. However, when we examine the distribution of stock returns, we observe that it has heavier tails compared to a normal distribution - indicating a higher probability of extreme values, which is expected due to regular unusual stock market movements. This behavior is better captured by a Student's t-distribution, which justifies its use as the target distribution in our Bayesian Neural Network (BNN) models. In contrast, the random walk model assumes normally distributed changes, which underestimates the likelihood of large price movements and fails to represent real-world market volatility completely.

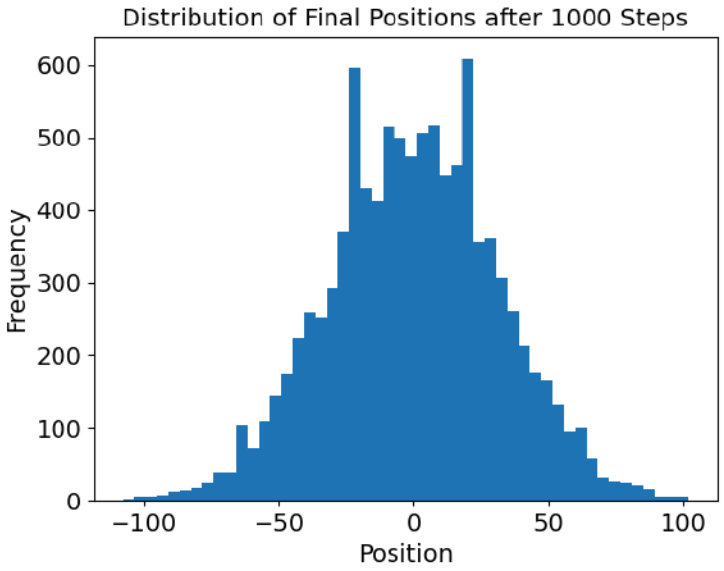

Fig 5: Distribution of Random Walk

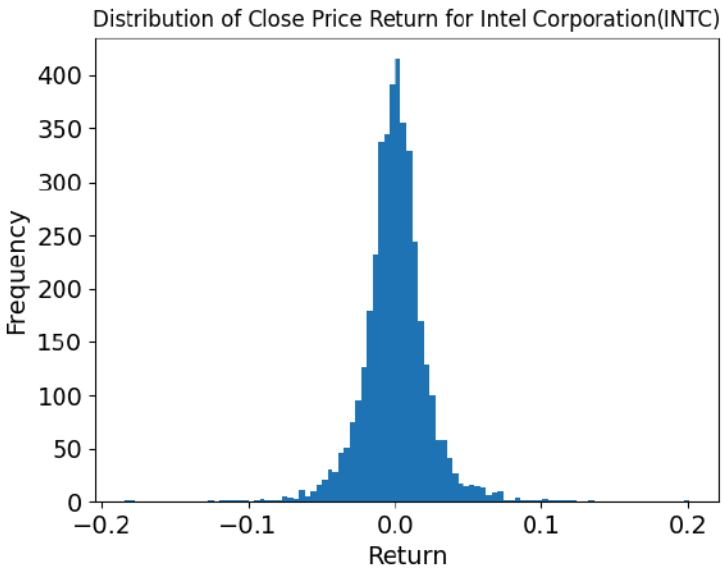

Fig 6: Distribution of Price Return

3.3.2 Volatility Simulation

Given a stock's current price $P_t = \$200$ and an estimated market volatility σ we aim to estimate the probability that the stock's price will reach or exceed $210 on the next trading day.

To do this, we model the $Close\ Price\ Return\ R$ as a normally distributed random variable:

$$R \sim N(0, \sigma^2)$$

The corresponding price at the next time step is:

$$P_{t+1} = P_t(1 + \mathrm{R})$$

We are interested in computing:

$$P(P \geq 210) = P(R \geq \frac{210-200}{200}) = P(R \geq 0.05)$$

Using the standard normal distribution:

$$\sigma = 0.01(1\%): z = 5, P(P \geq 210) \sim 0$$

$$\sigma = 0.05(5\%) : z = 1, P(P \geq 210) \sim 0.16$$

$$\sigma = 0.1(10\%): z = 0.5, P(P \geq 210) \sim 0.31$$

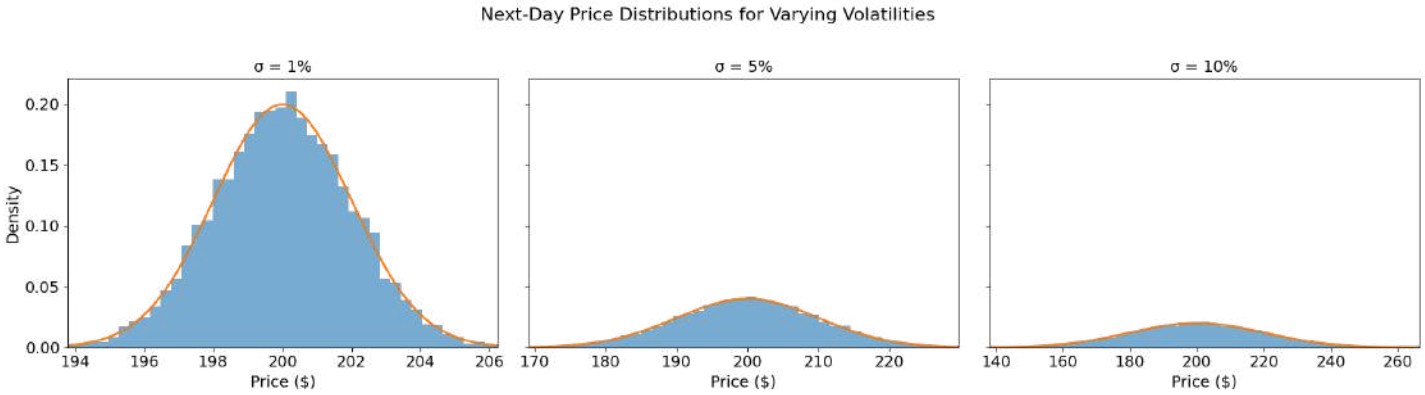

Fig 7: Next-Day Price Distributions for Varying Volatilities

As shown in the chart, higher volatility significantly increases the probability of large price movements. For example, with a 10% daily volatility, there is a 31% chance of the stock closing at or above $210, while at 1% volatility, that probability is near zero.

3.3.2 Bayesian Inference

Bayesian methods enable models to adapt as new information becomes available, making them well-suited for environments where conditions frequently change and patterns are difficult to predict.

The core of Bayesian inference is expressed as:

$$Posterior = Likelihood \times Prior \div Evidence$$

where:

- $prior$ - our belief about the parameters before observing data
- $likelihood$ - how likely the data is given the parameters
- $evidence$ - the probability of observing the data under all possible parameter values
- $posterior$ - updated belief about the parameters after observing data

$We\ define\ let\ \theta \in \Theta\ be\ the\ vector\ of\ model\ parameters$

$and\ let\ D = \left\{(x_i, y_i\right\}_{i=1}^{n}\ be\ the\ observed\ dataset$

Hence we can define the Bayes' Theorem:

$$P(\theta|\, D) = \frac{P(D|\theta) \times P(\theta)}{P(D)}$$

where

- $P(\theta|\, D) = P(\theta|\, y,\, x) - posterior$
- $P(D|\theta) = P(y|\theta, x) - likelihood$
- $P(\theta) = p(\theta|x) - prior\ knowledge$
- $P(D) = \int p(y,\, \theta \mid x) d\theta - evidence\ /\ marginal\ likelihood$

The complexity of an evidence integral usually is quite computationally expensive due to the high dimensionality of the feature datasets, hence for most likelihood problems this integral cannot be solved.

3.3.3 Probabilistic Programming Language Borch

Borch[4], the python probabilistic deep learning library, handles the complexity of computing the evidence using variational inference. Borch is built on four main components: distributions for sampling and log-probability calculations, Random Variables (RVs) that combine tensors with distributions, posteriors that manage inference sampling, and modules that track RVs and inference processes. One of the advantages of using Borch is its ability to approximate the model evidence, which is often a big challenge in Bayesian inference. It computes the integral by approximating true posterior $P(\theta \mid D)$ with a simple distribution $q(\theta)$.
It optimizes the Evidence Lower Bound (ELBO):

$$log\ P\ (D)\ \geq\ E_{q(\theta)}\ [log\ P(D \mid \theta)]\ -\ KL(q(\theta)\ ||\ P(\theta)]$$

Appendix [1] provides an example of the model class used in the ablation study, showing how the BNN integrates into the experimental pipeline.

4. METHODS

4.1 Data Collection

The dataset used in this research includes financial market information focused specifically on five major semiconductor stocks: Qualcomm (QCOM), Advanced Micro Devices (AMD), Intel Corporation (INTC), Nvidia Corporation (NVDA), and Taiwan Semiconductor Manufacturing (TSM).

The reason behind selecting these particular stocks is their consistently high trading volume and significant market volatility, which ensures the AI models have

sufficient diversity of scenarios for robust learning and accurate prediction capabilities.

We have trained and validated models using data from the period 2000-2018, making the analysis retrospective.

4.1.1 Market Data

Historical market data, including Open, High, Low, Close and Volume (OHLCV) features, has been provided by AI Alpha Lab[5], a global AI fund specializing in financial data analytics and market insights.

4.1.2 Macroeconomic Indicators

The macroeconomic indicators integrated into the dataset were sourced from the publicly available API[6]. These indicators were selected to augment the financial market data by encapsulating broader economic conditions and dynamics, thus enhancing the robustness and contextual understanding of the financial models. The specific indicators included are:

- DCOILWTICO: Oil prices, reflecting energy market conditions.
- CPIAUCSL: Consumer Price Index representing inflationary trends.
- FEDFUNDS: Federal Funds Rate, indicating monetary policy stance.
- GDP: Gross Domestic Product, capturing overall economic output.
- PSAVERT: Personal Saving Rate, reflecting consumer saving behavior and economic sentiment.
- UMCSEN: Consumer Sentiment Index, indicating public economic outlook.
- UNRATE: Unemployment Rate, insights into labor market conditions.

Some of these indicators are reported on monthly or quarterly frequencies, which differ from the daily granularity of financial time series. To align them with the rest

of the dataset, we applied linear extrapolation between available data points. This ensured a continuous and synchronized macroeconomic input stream for model training and evaluation.

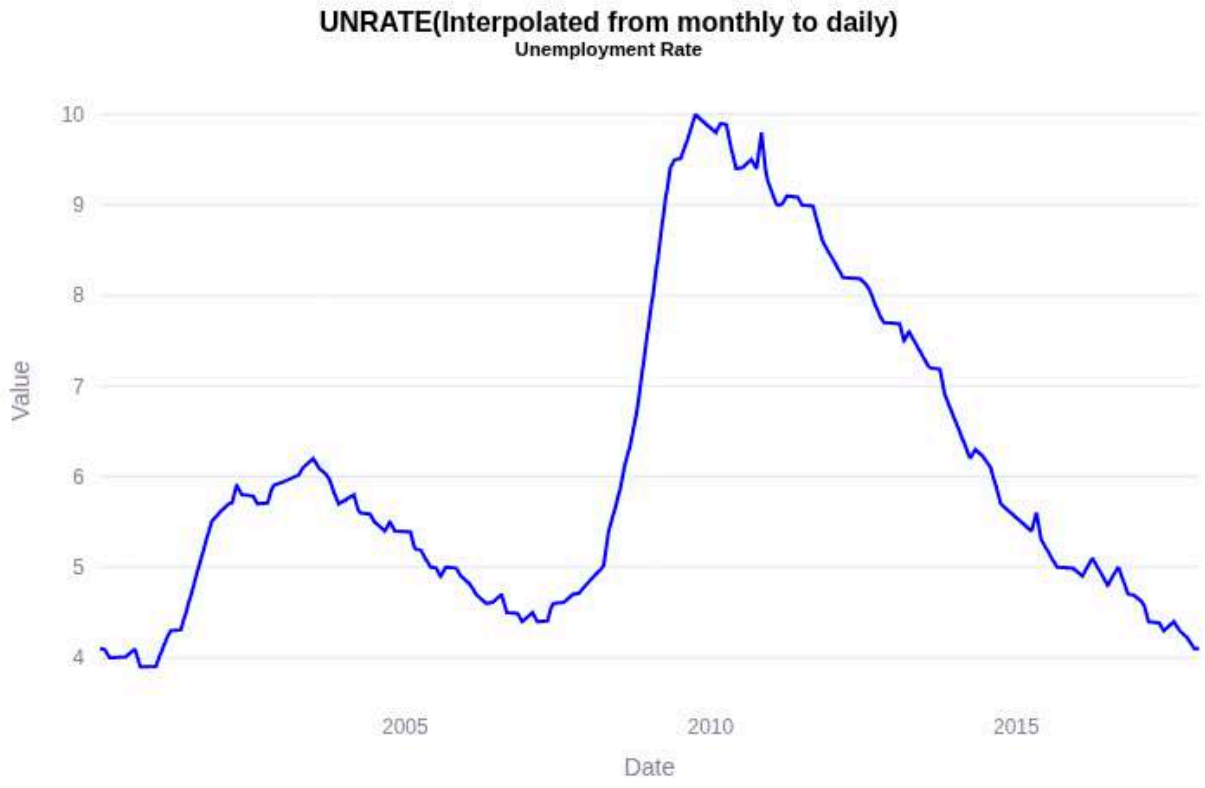

Fig 8: Demo Streamlit UNRATE Visualisation

4.1.3 Sentiment Data

Sentiment data was derived from textual summaries of Twitter content, processed using a specially configured assistant built with OpenAI GPT-4o model [7]. To convert textual summaries into quantifiable sentiment scores, a fine-tuned transformer-based model Twitter- roBERa-base[8] was implemented. This model assigns sentiment scores ranging from -1 - highly negative to +1 - highly positive, reflecting the market sentiment mood towards specific days of trading specific stocks.

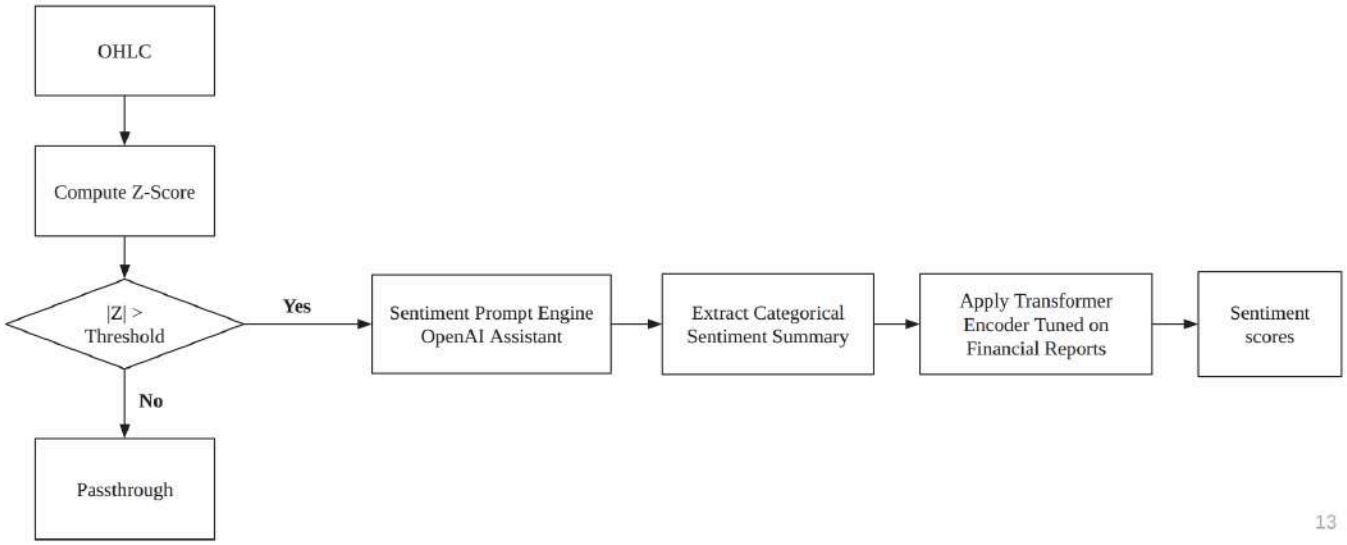

Fig 9: Sentiment Extraction Flowchart

4.1.4 Sentiment Signal Processing

Based on the sentiment extraction flow chart above, the following example, generated using an OpenAI Assistant[7], illustrates how social media sentiment was quantified and integrated into the forecasting pipeline:

"On November 5, 2015, Nvidia (ticker: NVDA) experienced a notable price jump, which coincided with a surge in Twitter activity. The sentiment on social media was mixed, with many users expressing excitement over Nvidia's strong earnings report and its advancements in gaming and AI technologies. This generated a wave of optimism among investors. At the same time, some users voiced concerns about market volatility and potential overvaluation, reflecting a degree of skepticism."

To quantify this sentiment, the tweet content was processed by a specialized transformer model named FitTwitterRoberta[8]. The model outputs the following sentiment probabilities: positive: 0.9345, neutral: 0.0608 and negative: 0.0047.

This resulted in a strong positive sentiment signal for this data point, which served as a valuable input for the main forecasting model.

4.2 Data Processing

A comprehensive automated data processing pipeline was implemented in Python, involving several preprocessing steps. These include data cleaning, feature engineering, normalization, anomaly detection, and combining multiple data sources.

4.3 Model Architectures

This section defines the core forecasting model architectures explored in this study. Each architecture, ranging from classical statistical models to modern deep learning and probabilistic approaches, was implemented to analyze its suitability for financial time series prediction. These models form the foundation of our ablation study, allowing us to systematically evaluate the contribution of different components and modeling strategies to predictive performance.

4.3.1 Statistical Models

As a starting point of the forecasting pipeline, classical statistical models such as ARIMA and SARIMA were applied to establish baseline performance. These models serve primarily as exploratory tools, and high predictive accuracy was not expected due to the complexity and non-linearity of financial time series.

$$ARIMA(p,\ d,\ q)$$

$where\ \ p\ -\ number\ of\ autoregressive\ terms,$
$d\ -\ degree\ of\ differencing,$
$q\ -\ number\ of\ moving\ averages\ terms$

$$SARIMA(p,\ d,\ q)\ \times\ (P,\ D,\ Q)_s$$

$where\ (p,\ d,\ q)\ -\ non\ seasonal\ ARIMA\ orders,$
$(P,\ D,\ Q)\ -\ seasonal\ SARIMA\ orders,$
$s\ -\ length\ of\ seasonal\ cycle$

4.3.2 Machine Learning Models

The Multiple Linear Regression (MLR) model establishes a linear relationship between a dependent variable $y \in R$ and a set of independent input features $x = [x_1, x_2,..., x_d]^T \in R^d$. The model estimates the output using a weighted sum of the input features plus a bias term.

The mathematical formulation is:

$$\widehat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_d x_d = x^T \beta$$

The model is trained by minimazing the sum of squared differences between the predicted values $\widehat{y}$ and the actual target values y, using the Ordinary Least Squares (OLS) method.

4.2.3 Simple Neural Networks

We have built a simple feedforward neural network consisting of one hidden layer.

Let the input vector be denoted by $x \in R^d$, where d is the number of input features.

The network first applies a linear transformation using weight matrix $W_1 \in R^{h \times d}$ and bias vector $b_1 \in R^h$, followed by a Leaky ReLU activation function with a negative slope of 0.01. This can be written as:

$$h = LeakyReLU(W_1 x + b_1)$$

Next, the hidden representation $h \in R^h$ is passed through a second linear layer defined by $W_2 \in R^{k \times h} and\ b_2 \in R^k$, producing the final output:

$$\hat{y} = W_2 h + b_2$$

Here, $\hat{y} \in R^k$ is the predicted output vector, where h is the number of hidden units and k is the output dimensionality.

4.2.4 Bayesian Neural Networks

Unlike static volatility estimates, Bayesian Neural Networks allow us to infer σ dynamically. The standard deviation σ is not fixed but is estimated by defining a prior distribution and updating it using observed data, allowing us to model both uncertainty and variability across time.

$$Let\ X \in R^{Nx\ D},\ where\ D = n_{features} and\ y \in R^{NxK},\ where\ K = n_{targets}$$

$$h = tanh(W_1 \bullet X + b_1),$$

$$where\ W_1 \sim N(0,\ \sigma^2) - weight\ prior\ and\ b_1 \sim N(0,\ \sigma^2) - bias\ prior$$

$$The\ model\ estimates\ both\ the\ location(\mu)\ and\ scale\ (\sigma)\ of\ the\ output:$$

$$\mu = W_\mu \bullet h + b_\mu,\ where\ b_\mu \sim N(0, 1),$$

$$where\ \mu - estimated\ price$$

$$log\ \sigma = W_\sigma \bullet h + b_\sigma,\ where\ b_\sigma \sim N(0, 2),$$

$$where\ \sigma - predicted\ uncertainty\ /\ variation$$

$$\sigma = softplus(log\ \sigma) + \epsilon,\ where\ \epsilon = 10^{-4}$$

$$Output\ distribution:\ y \sim Student's\ T(v = 4,\ \mu,\ \sigma)$$

$$where\ \nu - degrees\ of\ freedom,\ \mu - location\ and\ \sigma - scale$$

5. EXPERIMENTS

Various types of modeling approaches have been explored throughout the experimentation process. These experiments helped identify the most relevant and promising model architectures, along with effective feature sets. Based on these insights, a structured ablation study was designed to evaluate the contribution of each component to the model's performance.

5.1 Ablation Study Design

The following architectures were evaluated across multiple prediction windows (1, 3, and 5 days):

1. ARIMA and SARIMA as statistical baselines
2. Multiple Linear Regression (MLR) for scalable parametric prediction
3. Simple Neural Networks (SNN) for capturing non - linear patterns
4. Bayesian Neural Networks (BNN) for modeling uncertainty

For the features, we have decided to group them into three main groups: baseline data, macro data and extra data.

Extra data includes rolling, technical, datetime, anomalies and sentiment features.

An example of a Python function, which is designed to generate results for one of the architectures is available in Appendix [1].

5.2 Ablation Study Results

In this section, we summarize the outcomes of the ablation experiments, highlighting how different model configurations, feature sets, and transformations affect forecasting performance. By comparing variations within and across model

types, we identify the components that contribute most to prediction accuracy. Notably, statistical models were trained on the raw closing prices, consistent with their differencing-based architecture, while all machine learning and deep learning architectures were trained on price returns.

5.2.1 Autoregressive Integrated Moving Average (ARIMA)

Model	Order	Test MSE
ARIMA	(5, 1, 2)	87.7524
ARIMA	(5, 1, 3)	87.2625
ARIMA	(5, 1, 4)	87.5063
ARIMA	(5, 2, 2)	132.0894
ARIMA	(5, 2, 3)	131.8767
ARIMA	(5, 2, 4)	141.5588
ARIMA	(10, 1, 2)	86.7561
ARIMA	(10, 1, 3)	86.7650
ARIMA	(10, 1, 4)	92.1445
ARIMA	(10, 2, 2)	131.4923
ARIMA	(10, 2, 3)	138.9908
ARIMA	(10, 2, 4)	131.9085
ARIMA	(20, 1, 2)	87.2072
ARIMA	(20, 1, 3)	87.8019
ARIMA	(20, 1, 4)	87.6478
ARIMA	(20, 2, 2)	130.1978
ARIMA	(20, 2, 3)	130.3049
ARIMA	(20, 2, 4)	150.2526

Table 1: ARIMA Ablation Study Results

The ARIMA models across varying configurations delivered moderate performance, with the best Test MSE of 86.7561 observed for the (10, 1, 2) configuration. Most first-difference models (d=1) resulted in significantly lower

test MSEs compared to second-difference models (d=2), indicating that additional differencing may degrade performance, often exceeding 130 in Test MSE. These results suggest that first-order differencing captured sufficient stationarity, while additional differencing introduced noise and worsened generalization.

5.2.2 Seasonal Autoregressive Integrated Moving Average (SARIMA)

Seasonal ARIMA models slightly improved over ARIMA by capturing seasonality, especially in lower seasonal orders. The best Test MSE was 86.6839 for (10, 1, 3)(1, 0, 1, 40), outperforming the best ARIMA model.

Model	Order	Seasonal Order	Test MSE
SARIMA	(10, 1, 1)	(0, 0, 0, 5)	86.793
SARIMA	(10, 1, 1)	(0, 0, 0, 40)	86.793
SARIMA	(10, 1, 1)	(0, 0, 1, 5)	86.920
SARIMA	(10, 1, 1)	(0, 0, 1, 40)	86.779
SARIMA	(10, 1, 1)	(1, 0, 0, 40)	86.684
SARIMA	(10, 1, 1)	(1, 0, 1, 40)	87.350
SARIMA	(10, 1, 3)	(0, 0, 0, 5)	86.894
SARIMA	(10, 1, 3)	(0, 0, 0, 40)	86.894
SARIMA	(10, 1, 3)	(0, 0, 1, 5)	86.921
SARIMA	(10, 1, 3)	(0, 0, 1, 40)	86.936
SARIMA	(10, 1, 3)	(1, 0, 0, 40)	86.706
SARIMA	(10, 1, 3)	(1, 0, 1, 40)	86.803

Table 2: SARIMA Ablation Study Results

5.3.2 Multiple Linear Regression (MLR)

MLR models demonstrated excellent performance at short-term predictions, particularly for the prediction window 1, where the best Test MSE reached as low as 2.46e-05. Test errors increased with longer horizons (3 and 5), and the inclusion

of non-linear transformations or complete feature sets did not always lead to improvement. Notably, adding all features for non-linear MLR significantly worsened test performance, indicating overfitting (Test MSE > 2,000), which highlights the model's sensitivity to input complexity.

Model	Transformations	Features	Test MSE
MLR_INTC_1	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: True	2.46e-05
MLR_INTC_3	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: True	9.94e-05
MLR_INTC_1	Scaled: True, Non-linear: False	Baseline: True, Macro: False, Extra: False	1.145e-04
MLR_INTC_1	Scaled: True, Non-linear: True	Baseline: True, Macro: False, Extra: False	1.236e-04
MLR_INTC_1	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: False	1.306e-04
MLR_INTC_5	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: True	1.354e-04
MLR_INTC_3	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: False	1.487e-04
MLR_INTC_3	Scaled: True, Non-linear: False	Baseline: True, Macro: False, Extra: False	1.498e-04
MLR_INTC_3	Scaled: True, Non-linear: True	Baseline: True, Macro: False, Extra: False	1.504e-04
MLR_INTC_5	Scaled: True, Non-linear: False	Baseline: True, Macro: True, Extra: False	1.643e-04
MLR_INTC_5	Scaled: True, Non-linear: False	Baseline: True, Macro: False, Extra: False	1.686e-04
MLR_INTC_5	Scaled: True, Non-linear: True	Baseline: True, Macro: False, Extra: False	1.709e-04
MLR_INTC_1	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: False	5.462e-04
MLR_INTC_3	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: False	1.2061e-03
MLR_INTC_5	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: False	2.0059e-03
MLR_INTC_1	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: True	2.9981e+03
MLR_INTC_3	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: True	5.7779e+03
MLR_INTC_5	Scaled: True, Non-linear: True	Baseline: True, Macro: True, Extra: True	4.1486e+04

Table 3: Multiple linear regression Ablation Study Results

5.2.4 Simple Neural Network (SNN)

Neural network models provided stable and competitive Test MSEs in the 1.6e-04 to 2.0e-04 range, with the best result at 1.671e-04 for prediction window 3 using a minimal feature set. Performance degraded slightly as additional features were included. These models outperformed many MLR models on mid-term horizons but were less effective than MLR on next day prediction.

Model	Transformations	Features	Test MSE
NN_INTC_3	Scaled: True	Baseline: True, Macro: False, Extra: False	1.671e-04
NN_INTC_3	Scaled: True	Baseline: True, Macro: True, Extra: True	1.680e-04
NN_INTC_5	Scaled: True	Baseline: True, Macro: True, Extra: True	1.690e-04
NN_INTC_1	Scaled: True	Baseline: True, Macro: True, Extra: True	1.794e-04
NN_INTC_5	Scaled: True	Baseline: True, Macro: True, Extra: False	1.979e-04
NN_INTC_5	Scaled: True	Baseline: True, Macro: False, Extra: False	1.983e-04
NN_INTC_1	Scaled: True	Baseline: True, Macro: False, Extra: False	2.007e-04
NN_INTC_3	Scaled: True	Baseline: True, Macro: True, Extra: False	2.076e-04
NN_INTC_1	Scaled: True	Baseline: True, Macro: True, Extra: False	5.373e-04

Table 4: Simple Neural Network Ablation Study Results

5.2.5 Bayesian Neural Network (BNN)

Bayesian neural networks (BNNs) achieved strong performance, with a best test MSE of 2.105e-04 for a prediction window of 5 days. Unlike SNNs or machine learning regressions (MLRs), Bayesian neural networks (BNNs) handle increasing complexity more gracefully. Although performance was slightly worse than SNNs at shorter horizons, BNNs provided consistent and reliable estimates. However, complex models still led to higher errors - 1.745e-03 - suggesting that even probabilistic models require careful feature selection.

Model	Transformations	Features	Test MSE
BNN_INTC_5	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: False	2.105e-04
BNN_INTC_5	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: True	2.130e-04
BNN_INTC_3	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: True	2.147e-04
BNN_INTC_3	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: False	2.281e-04
BNN_INTC_1	Scaled: True, Student-T	Baseline: True, Macro: False, Extra: False	2.310e-04
BNN_INTC_5	Scaled: True, Student-T	Baseline: True, Macro: False, Extra: False	2.372e-04
BNN_INTC_1	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: False	2.386e-04
BNN_INTC_3	Scaled: True, Student-T	Baseline: True, Macro: False, Extra: False	4.653e-04
BNN_INTC_1	Scaled: True, Student-T	Baseline: True, Macro: True, Extra: True	1.745e-03

Table 5: Bayesian Neural Network Ablation Study Results

6. RESULTS

After conducting an ablation study and evaluating a wide range of model architectures, feature permutations, and transformation techniques, we identified the best-performing combinations for each forecast horizon. The final comparison table below summarizes the test Mean Squared Error (MSE) across the selected models, reflecting the most effective setups in terms of predictive performance.

Linear Regression models demonstrated the best performance for short-term (1-day) predictions, benefiting from a simple structure and effective scaling. Simple Neural Networks (SNNs) achieved the lowest error rates on 3-day and 5-day horizons, making them the most suitable architecture for middle to long-term forecasts due to their capacity to capture nonlinear dependencies.

Bayesian Neural Networks (BNNs), although slightly behind in absolute Mean Squared Error (MSE), excelled in modeling predictive uncertainty and capturing market volatility. Importantly, their performance remained stable across all horizons, and they performed exceptionally well in long-term settings.

7. CONCLUSIONS

The results of our study highlight the importance of selecting the appropriate modeling framework based on the prediction window and the desired balance between accuracy and uncertainty estimation. While linear regression remains a strong baseline for short-term forecasts, more complex architectures, such as simple neural networks and Bayesian approaches, prove essential as the prediction window extends. In particular, Bayesian Neural Networks not only maintain competitive accuracy but also provide robust uncertainty estimates, which are crucial for informed financial decision-making in volatile markets. Their ability to

model predictive distributions and quantify confidence makes them a compelling choice for real-world financial forecasting systems.

8. ACKNOWLEDGMENTS

We would like to acknowledge that the formatting of this work was carried out with the assistance of ChatGPT, a large language model developed by OpenAI. We confirm that its use was strictly limited to formatting and layout tasks; all ideas, and written content presented here were created solely by the authors.

9. DEMO

9.1 Streamlit

The Demo App is an interactive visualization tool built using Streamlit[9]**.** It provides an end-to-end workflow for financial time series modeling - starting from data loading and preprocessing, through exploratory analysis and model training, to prediction, logging to tracking server, and performance evaluation.

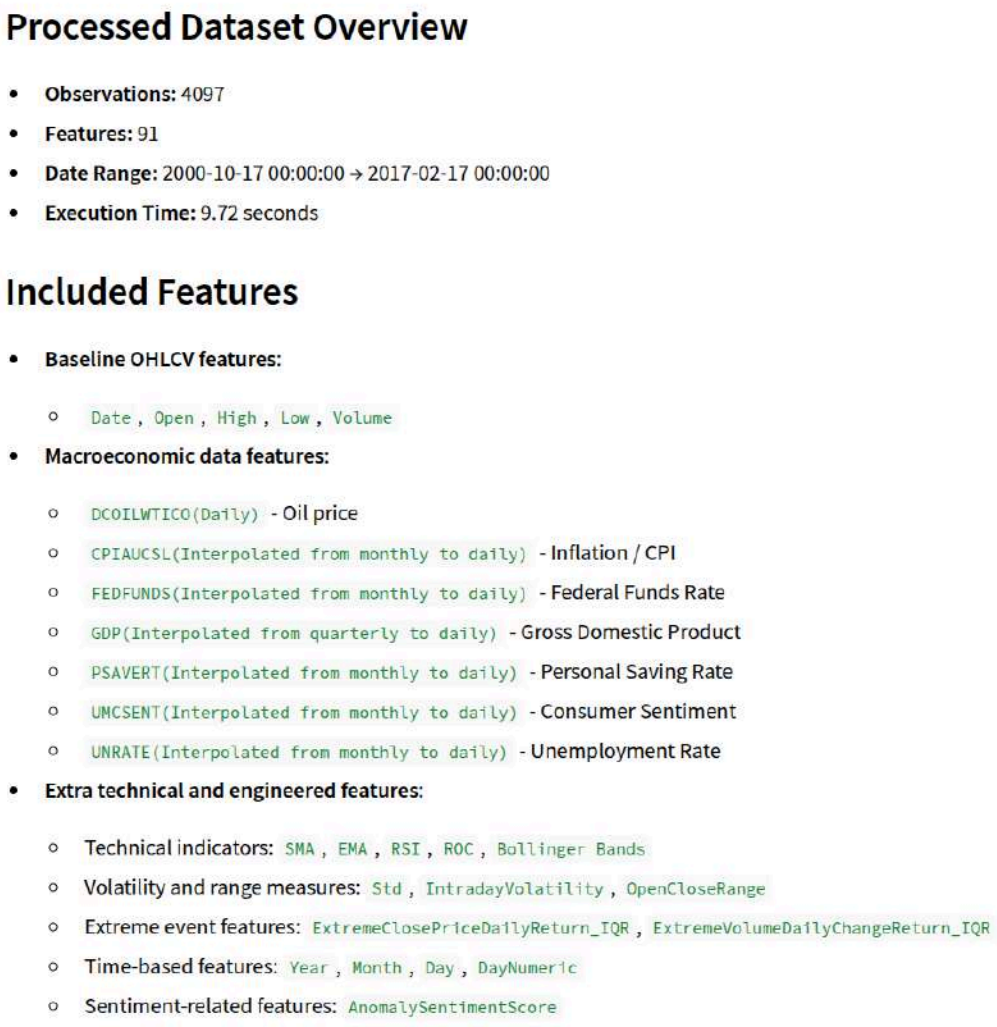

Fig 10: Streamlit Processed Dataset and Feature Engineering Breakdown

9.2 MLflow

To ensure reproducibility and efficient experiment management, we have implemented an MLflow Tracking Server[10] to log and monitor all modeling experiments. The setup captures key details, including model configurations, training and validation losses, evaluation metrics, and other relevant parameters, throughout the experimentation process.

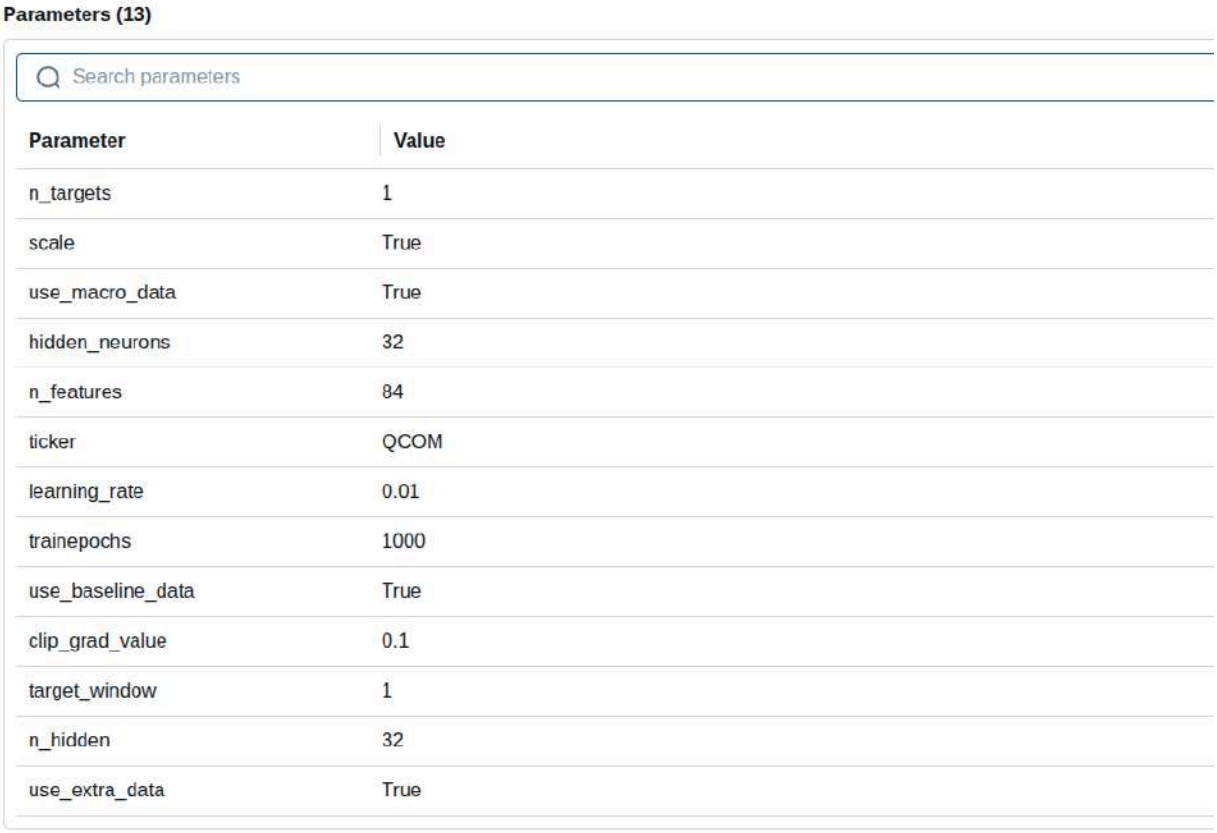

Fig 11: MLflow Model Configuration Overview

The following example from MLflow visualizes model predictions along with uncertainty intervals, demonstrating performance over time.

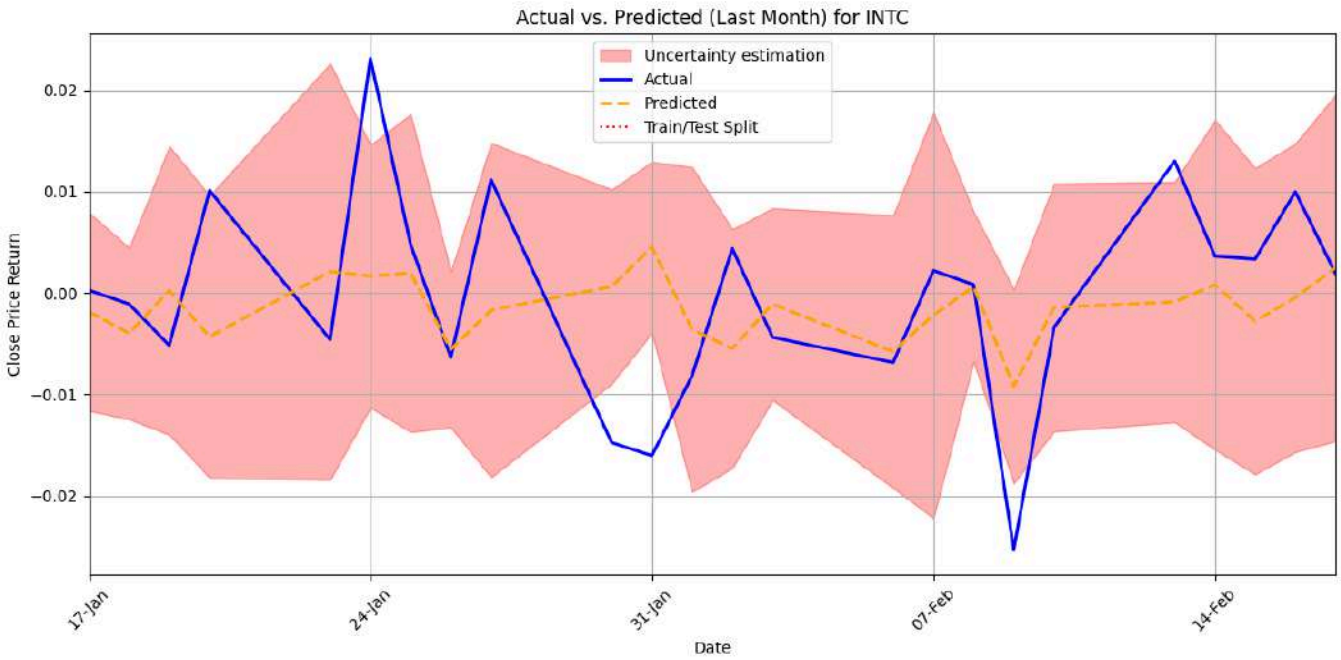

Fig 12: MLflow Actual vs Predicted Evaluation Plot

10. REFERENCES

[1] “Borch: A Deep Universal Probabilistic Programming Language”, arXiv:2209.06168, Sep 2022

[2] “Enhanced Bayesian Neural Networks for Macroeconomics and Finance”, arXiv:2211.04752v2, Nov 2022

[3] “Sentiment trading with large language models”,arXiv:2412.19245, Dec 2024

[4] Deep Universal Probabilistic Programming Language

[5] AI Alpha Lab ApS

[6] Federal Reserve Bank of St. Louis

[7] Open AI Assistant API Documentation

[8] Twitter-roBERTa-base Sentiment Transformer

[9] Python Streamlight Framework

[10] Mlflow Tracking and Logging Setup

APPENDIX 1

```python
class BayesianNeuralNet_v5(borch.Module):
    def __init__(
        self,
        ticker,
        n_features,
        n_targets,
        n_hidden,
        X_train,
        y_train,
        X_test,
        y_test,
    ):
        super().__init__(posterior=borch.posterior.Automatic())

        self.linear1 = borch.nn.Linear(
            n_features, n_hidden,


        )
        self.activation = nn.Tanh()

        self.mean_head = borch.nn.Linear(
            n_hidden, n_targets,
            bias=dist.Normal(0, 1),
        )

        self.logstd_head = borch.nn.Linear(
            n_hidden, n_targets,
            bias=dist.Normal(0, 2),
        )

        # Store inputs
        self.ticker = ticker
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test

    def forward(self, X):
        x = self.linear1(X)
        x_activated = self.activation(x)

        mean = self.mean_head(x_activated)
        log_std = self.logstd_head(x_activated)
        std = F.softplus(log_std) + 1e-4

        self.target = borch.distributions.StudentT(df=4, loc=mean, scale=std)
        return self.target
```

Fig 13: Main Bayesian Neural Network class with prior settings and output StudentT distribution

Role:
You are a financial sentiment analyst trained to extract the sentiment from Twitter surrounding a stock on a specific date, based on either price or volume anomaly, computed either by Z-score formula or anomaly detection ML algorithm - Isolation Forest model. A z-score can theoretically take any real value from negative to positive infinity, since it measures standard deviations from the mean. In practice, observations with |z| > 2 (unusual) or especially |z| > 3 (highly unusual) are typically flagged as anomalies.

Background:
The task is related to predicting the future stock price, based on different types of data. I build some anomaly detection algorithms, based on price and volume change, so I have received the dates of the anomaly. And what I am missing currently is the actual textual/sentiment explanation of those anomalies dates. So I need you to apply your knowledge and extract those twitter summaries for me.

Task:
Your goal is to capture the related to the certain stock emotional tone in the Twitter environment, extract and summarize it.

Focus on typical trading emotional tones: euphoria, fear, disbelief, greed, boredom and overall direction of sentiment: bullish, bearish, or mixed. Also you need to extract the concrete news/event that influenced the change in price. Also you should be sure if it's huge bullish or bearish movement. Do not use repeated words and emotions. But use your twitter knowledge to answer. Provide a summary at least of 100 words.

Write in a human, twit style.

Input format:
json
```
{
  "ticker": "TICKER_SYMBOL",
  "date": "YYYY-MM-DD"
}
```

Output format:
json
```
{
  "ticker": "TICKER_SYMBOL",
  "date": "YYYY-MM-DD",
  "summary_tweet": "SUMMARY"
}
```

Fig 14: Instructions for OpenAI API Twitter Sentiment Extractor Assistant

```python
def run_bayesian_single_model(
    ticker: str,
    trainepochs: int = 1000,
    learning_rate: float = 0.01,
    clip_grad_value: float = 0.1,
    hidden_neurons: int = 1,
    scale: bool = True,
    predsamples: int = 15,
    use_baseline_data_bool: bool = True,
    use_macro_data_bool: bool = True,
    use_extra_data_bool: bool = True,
    target_window: int = 5,
    cachedir: str = "cachedir"
):
    output = train_and_eval_bnn_model(
        ticker,
        trainepochs=trainepochs,
        learning_rate=learning_rate,
        clip_grad_value=clip_grad_value,
        hidden_neurons=hidden_neurons,
        scale=scale,
        predsamples=predsamples,
        use_baseline_data_bool=use_baseline_data_bool,
        use_macro_data_bool=use_macro_data_bool,
        use_extra_data_bool=use_extra_data_bool,
        target_window=target_window,
        cachedir=cachedir
    )

    if output is None:
        return None

    model, actual, predicted, results = output
    results_df = pd.DataFrame([results])
    results_df.to_csv("results/bayesian_neural_networks_results.csv", index=False)

    return model, actual, predicted, results_df
```

Fig 15: This function configures, trains, and evaluates a Bayesian Neural Network with flexible data and model settings, enabling automated experimentation for ablation studies.

```python
def run_bayesian_all_neural_networks(ticker: str):
    results_list = []
    windows = [1, 3, 5]

    for scale_bool in [True]:
        for window in windows:
            for use_macro_data, use_extra_data in [
                (False, False),
                (True, False),
                (True, True),
            ]:
                model, actual, predicted, results = train_and_eval_bnn_model(
                    ticker,
                    trainepochs=1000,
                    learning_rate=0.01,
                    clip_grad_value=0.1,
                    hidden_neurons=1,
                    scale=scale_bool,
                    predsamples=15,
                    use_baseline_data_bool=True,
                    use_macro_data_bool=use_macro_data,
                    use_extra_data_bool=use_extra_data,
                    target_window=window,
                    cachedir="cachedir"

                )

                results_list.append(results)

    results_df = pd.DataFrame(results_list)
    results_df = results_df.sort_values(by=["test MSE","target"], ascending=True)
    results_df.to_csv("results/bayesian_neural_networks_results_NEW.csv", index=False)

    return results_df
```

Fig 16: This function executes multiple Bayesian Neural Network configurations by varying feature sets and target windows, collecting and exporting sorted results for evaluation.