Comparing Traditional Time Series Models and Deep Learning Approaches in Stock Price Forecasting

Nyuhai Liu

Dept of Mathematics, University College London, London, United Kindom

Abstract. This study conducts a comparative analysis between traditional statistical models and state-of-the-art deep learning approaches in the context of stock price forecasting. Using a decade of daily trading data from Apple Inc. (AAPL), spanning 2014 to 2024, this paper evaluates models on two forecasting tasks: predicting the closing price and predicting the next-day log-returns. The experiments demonstrate that the ARIMA model remains a strong baseline, particularly in controlling absolute prediction errors and maintaining stable performance across different market regimes. In contrast, recurrent neural networks such as GRU exhibit superior performance across almost all evaluation metrics, suggesting that they are better able to capture temporal dependencies in financial time series. However, deeper and more complex hybrid architectures, including stacked and convolutional variants, show a clear tendency to overfit the training data and fail to generalize to unseen periods, especially during volatile market phases. These findings underline the inherent difficulties of applying deep learning in financial forecasting, where challenges such as high noise levels, structural breaks, and low signal-to-noise ratios persist. Future work should therefore emphasize stronger regularization strategies, models explicitly designed to handle non-stationarity, and advanced methods of feature engineering and signal extraction to achieve more reliable predictive performance.

1 Introduction

Financial markets are characterised by volatility and uncertainty, particularly in stock prices. Numerous latent factors contribute to this phenomenon, such as changes in immigration or trade policies, natural disasters, and public health crises (e.g., COVID-19). Efforts to predict stock prices have never ceased, driven by the objectives of risk management and enhancing returns. However, the complexity of markets has posed successive bottlenecks and challenges to these endeavours. One of the most classic attempts was the introduction of the ARIMA model, which has been widely adopted for decades due to its interpretability and statistical validity. However, it still has numerous limitations. Stock markets are highly-volatile, non-linear, and non-stationary, while traditional ARIMA models rely on several assumptions, limiting their ability to capture non-linear relationships and long-term time correlations. In recent years, with the advancement of deep learning and the rapid increase in computational power, an increasing number of deep learning models have been applied across various fields.

Among these, Recurrent Neural Networks (RNNs) are frequently employed for processing time-series structured data. The unique architecture of RNNs enables the description of dynamic temporal behaviour, leading to impressive performance in time-series forecasting, speech recognition, natural language processing and so forth.

Deep learning is a branch of machine learning that employs artificial neural networks (ANNs) as its structural foundation.[1] Typically employing multi-layered architectures, the term “deep” refers to the network's depth. Artificial neural networks are fundamentally non-linear statistical data modelling tools (probabilistic models) designed to mimic the structure and function of the animal central nervous system, approximating or estimating complex functions. In stock price forecasting, neural network models take multiple features (such as open, high, low, volume, and various technical indicators) as inputs. These models approximate the complex real-world functions influencing stock price movements, outputting predicted share prices.

ANN comprises input, hidden, and output layers. Each layer contains multiple neurons, with the strength of connections between neurons determined by distinct weights. Neuron is a computational processing unit within the network that collects all components transmitted from the upper layer. Through a specified activation function, it transmits the activated output value to different neurons in the next layer with varying weights.[2] Activation functions are typically non-linear and vary according to the network architecture and task type. The most commonly employed activation functions include sigmoid, tanh, ReLU etc. The learning method employed by neural networks also varies, while backpropagation is frequently used to adjust the weights across layers, thereby approximating the target function.

In 1974, Werbos proposed the backpropagation algorithm, which at the time failed to generate much excitement.[3] It was not until the 1980s that backpropagation became widely applied to deep neural networks.[4] Simply put, backpropagation comprises two phases: forward and backward propagation, followed by weight updates. Each iteration of the propagation process involves forward propagation: feeding training data into the neural network to obtain predicted outputs; and backward propagation: calculating the difference between these predictions and the actual training targets, known as the model's loss, based on a defined loss function.

The most prevalent networks in deep learning include FFNNs, RNNs, and CNNs, each suited to distinct tasks. CNNs are frequently employed for image processing, while RNNs, unlike FFNNs, incorporate time-dependent hidden states within their neurons. Unlike feed-forward neural networks, which accept inputs of relatively fixed structure, RNNs can describe dynamic temporal behaviour. This enables their widespread application in time-dependent tasks, accepting broader sequence-structured data inputs. However, during backpropagation to update weight parameters, RNN models are prone to gradient vanishing or explosion [5], limiting the model's ability to retain only short-term dependencies and hindering the capture of long-term event correlations. Integrating LSTM can partially resolve this issue.

Hochreiter and Schmidhuber proposed the Long Short-Term Memory (LSTM) network in 1997.[6] The core of the LSTM lies in the introduction of a memory cell (cell state) and a gating mechanism. Unlike traditional RNNs which directly overlay information using nonlinear functions, LSTMs primarily employ addition for updates during propagation. This allows information from distant past steps to survive gradient propagation, thereby preventing gradient vanishing.

In 2014, Cho et al. proposed the GRU [7], a simplified variant of LSTM with fewer parameters and faster computation, achieving comparable or occasionally superior performance. The drawback of GRUs lies in their lack of an explicit ‘long-term memory cell’ (cell state), potentially rendering them less stable than LSTMs when handling extremely long sequence dependencies.

This paper will focus on discussing the performance of LSTM, GRU, and several ensemble learning models in forecasting AAPL stock prices (close), while also attempting to use these models for log-return forecasting. Data was downloaded from Yahoo Finance, covering the period 1 January 2014 to 31 December 2024, encompassing close, open, high, low, and volume. **Fig 1** shows the close prices of AAPL over this period.

Fig. 1. AAPL close price from 2014 -2024

Feature enhancement, window sliding and data standarisation were introduced to process raw data, transforming it into an acceptable form for the models. Multiple complementary evaluation metrics were employed to comprehensively assess predictive performance across different models, including RMSE, MAE, and R. By combining these metrics, it is possible to comprehensively evaluate the strengths and weaknesses of various models from multiple perspectives.

This study systematically compares the performance of traditional time series models against deep learning models in stock forecasting tasks. Using Apple Inc. (AAPL) daily data as an example, prediction experiments were conducted on both closing prices (Close) and logarithmic returns (Return). Results indicate that traditional ARIMA models demonstrate robust performance in capturing short-term linear dependencies and trends. Conversely, deep learning models (including single-layer and multi-layer LSTM, GRU, CNN-LSTM, GRU-LSTM, etc.) exhibit advantages in modelling non-linear features when predicting Close, yet remain constrained by high data noise, weak signal strength, and limited sample size when forecasting Return. This paper further discusses core challenges facing current deep learning approaches in financial forecasting, including susceptibility to overfitting, difficulty in handling structural breaks, and extremely weak signals when predicting returns. The study highlights that future research urgently requires breakthroughs in model regularisation, non-stationary modelling, and enhancing signal extraction capabilities.

2 Related Work

Classical statistical models such as ARIMA remain standard baselines for financial time-series forecasting due to their transparency and sound inference procedures [8]. Yet decades of empirical evidence show that asset returns exhibit heavy tails, volatility clustering, and other stylized facts inconsistent with linear-Gaussian stationarity, which limits ARIMA's expressive power in turbulent markets [9-11]. With the resurgence of neural networks, backpropagation-trained deep models [3], [4] and, in particular, sequence models like LSTM [6], and GRU [7], have been widely adopted to capture nonlinear dynamics and long-range temporal dependencies. In equity prediction specifically, LSTM-based approaches have demonstrated superior out-of-sample directional accuracy and risk-adjusted returns versus traditional machine-learning baselines on large universes (e.g., S\&P 500 constituents) [12], [13], and have been further enhanced by exogenous information such as news/sentiment [14]. Beyond pure recurrent architectures, hybrids and variants—ranging from classical ARIMA–ANN combinations [15] to attention-augmented recurrent ensembles and GRU/XGBoost pipelines—have been proposed to balance interpretability, nonlinearity, and robustness [16]. Comparative studies and surveys corroborate these trends: deep learning (LSTM/GRU and their hybrids) generally outperforms linear models on volatile, nonlinear financial series, while performance differences across recurrent families (LSTM vs. GRU) can be dataset- and horizon-dependent.[17-18] Collectively, this literature motivates the study design: retaining ARIMA as an interpretable baseline and benchmark a suite of deep recurrent models—including GRU—under consistent data splits and evaluation metrics to assess predictive efficacy in stock price forecasting.

3 Modelling Methodology

3.1 ARIMA

The autoregressive integrated moving average (ARIMA) model is one of the most widely used classical approaches for time-series forecasting. An ARIMA model combines three components: an autoregressive (AR) part that regresses the current observation on its past values, a moving average (MA) part that models the dependency between the current observation and past forecast errors, and an integration (I) component.

Formally, an ARIMA(p,d,q) process can be written as:

$$y_t = c + \sum_{i=1}^{p} \phi_i \, y_{t-i} + \sum_{j=1}^{q} \theta_j \, \varepsilon_{t-j} + \varepsilon_t \qquad (1)$$

where p is autoregressive order, d is differencing order, q is moving average order, and ε_t is white noise error term.

In practice, ARIMA models are often estimated in a walk-forward manner, where parameters are re-fitted periodically to adapt to structural changes in the data. Walk-forward manner enables the model to be consistent with real-world situations.

3.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is designed to overcome the vanishing and exploding gradient problems of traditional RNNs when modeling long sequences. LSTMs introduce a gating mechanism that allows the model to selectively retain or discard information over long temporal spans, making them particularly well-suited for sequential data such as text, speech, and financial time series. At each time step, the LSTM maintains two types of states: the cell state C_t and the hidden state h_t , corresponding to long-term memory and short-term memory respectively. The information flow is controlled by three gates: the input gate i_t, the forget gate f_t, and the output gate o_t. The gating components employ sigmoid functions to constrain outputs between 0 and 1, thereby regulating the flow of information. The memory cell stores long-term information across time steps, enabling near-unmodified forward transmission of data.

Formally, the LSTM updates at time step t can be expressed as:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \qquad (2)$$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \qquad (3)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \qquad (4)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c), \qquad (5)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \qquad (6)$$

$$h_t = o_t \odot \tanh(c_t) \tag{7}$$

Where x_t is the input vector at time t, σ denotes the sigmoid activation, tanh is the hyperbolic tangent activation, and $\odot$ is the element-wise multiplication.

3.3 Gated Recurrent Units (GRU)

Compared to LSTM, GRU features simpler architecture, faster training, and particular suitability for small datasets or low-computational-power devices. The GRU primarily employs two gates: the Update gate and the Reset gate. See **Fig 2** for the structure of GRU.

Given an input sequence $\{x_t\}_{t=1}^{T}$ with $x_t \in \mathbb{R}^d$ and a hidden state $h_t \in \mathbb{R}^h$, the GRU controls information flow with two gates and maintains a single recurrent state h_t:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{8}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{9}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h), \tag{10}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{11}$$

Here, $\sigma(\cdot)$ is the logistic sigmoid, $\tanh(\cdot)$ is the hyperbolic tangent, and $\odot$ denotes elementwise multiplication. The learnable parameters are $W_\star \in \mathbb{R}^{h\times d}$, $U_\star \in \mathbb{R}^{h\times h}$, and $b_\star \in \mathbb{R}^h$ for $\star \in \{z, r, h\}$.

The update gate z_t interpolates between the previous state and the candidate state, controlling how much past information is retained. The reset gate r_t regulates how much of h_{t-1} is used when forming the candidate $\tilde{h}_t$, enabling the unit to drop irrelevant history. A single GRU layer uses three affine blocks, with parameter count typically fewer than LSTM (which uses four blocks), yielding faster training/inference at similar capacity.

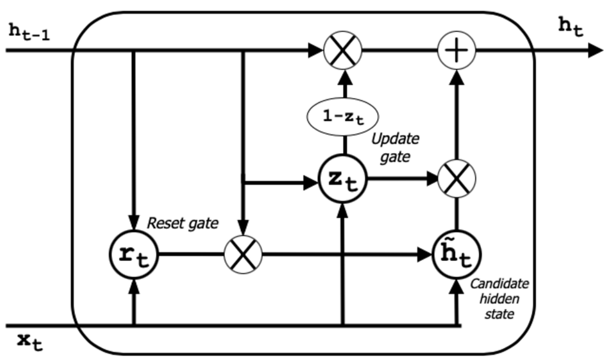

Fig. 2. GRU Structure

3.4 Ensemble Learning Models

To capture both local feature patterns and sequential dependencies, constructing hybrid models by combining two distinct feature extractors $f^{(1)}(\cdot)$ and $f^{(2)}(\cdot)$. Let the input sequence be X, the two representations are obtained as:

$$h^{(1)} = f^{(1)}(X),\ h^{(2)} = f^{(2)}(X) \tag{12}$$

Two integration strategies are considered for combining these representations: one is stacking (sequential composition): the output of the first extractor serves as the input of the second; the other one is concatenation (parallel composition): both extractors operate on the same input in parallel, and their outputs are concatenated along the feature dimension.

$$h = f^{(2)}(f^{(1)}(X)) \tag{13}$$

$$h = \text{concat}\left(h^{(1)}, h^{(2)}\right) \tag{14}$$

In practice, different hybrid models instantiate $f^{(1)}$ and $f^{(2)}$ with specific neural modules. For example, in the CNN-GRU model, $f^{(1)}$corresponds to the CNN block and $f^{(2)}$to the GRU encoder, combined via concatenation as defined above. After comparing the performance between the two methods under same condition, the one with better results will be presented. Three hybrid models were tried, including CNN-LSTM (stack), CNN-GRU and LSTM-GRU (concatenation).

3.5 Evaluation Metrics

To comprehensively evaluate model performance, multiple regression and classification metrics are chosen, which are commonly used in financial forecasting: ***Root Mean Squared Error (RMSE):*** RMSE penalizes large deviations more heavily and reflects the standard deviation of prediction errors. ***Mean Absolute Error (MAE):*** MAE measures the average magnitude of prediction errors without considering their direction. ***Pearson Correlation Coefficient (R):*** This metric captures the linear correlation between predictions and true values.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2} \tag{15}$$

$$\mathrm{MAE} = \frac{1}{\mathrm{N}}\sum_{\mathrm{t}=1}^{\mathrm{N}}|\mathrm{y}_{\mathrm{t}} - \hat{\mathrm{y}}_{\mathrm{t}}| \tag{16}$$

$$R = \frac{\sum_{t=1}^{N}(y_t - \bar{y})(\hat{y}_t - \bar{\hat{y}})}{\sqrt{\sum_{t=1}^{N}(y_t - \bar{y})^2}\sqrt{\sum_{t=1}^{N}(\hat{y}_t - \bar{\hat{y}})^2}} \tag{17}$$

3.6 Multi-task LSTM (Return prediction)

To enhance generalization on return prediction, a multi-task learning (MTL) framework was adopted, which jointly optimizes regression and classification tasks. The regression head and classifiation head share one input from stack LSTM. LSTM is responsible for extracting long-term dependencies in the data, the regression head uses linear as the activation function, MSE as the loss to predict the magnitude of rise and fall of return. While the classification head uses sigmoid as the activation function and binary cross entropy as the loss to predict the direction of changes in return.

The total loss is defined as:

$$\mathcal{L} = \mathcal{L}_{\mathrm{reg}} + \lambda_{\mathrm{cls}}\,\mathcal{L}_{\mathrm{cls}} \tag{18}$$

- Regression loss (MSE):

$$\mathcal{L}_{\mathrm{reg}} = \frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2 \tag{19}$$

- Classification loss (binary cross-entropy):

$$\mathcal{L}_{\mathrm{cls}} = -\frac{1}{N}\sum_{t=1}^{N}[\, z_t \log(\hat{z}_t) + (1 - z_t)\log(1 - \hat{z}_t)] \tag{20}$$

Here, $\lambda_{\mathrm{cls}} = 0.1$controls the weight of the classification task relative to regression. Meaning that when training the model, more attention is paid on regression task instead of classification. This framework encourages the model to learn representations that are informative for both numerical prediction and trend classification.

4 Experiment

4.1 Experimental Environment and Implementation

All experiments are implemented in **Python** on **Google Colab.** This study uses the standard scientific Python stack for data handling and visualization (*NumPy, Pandas, SciPy, scikit-learn, Matplotlib*). Classical time-series models (ARIMA/SARIMA) are built with **statsmodels** (*sm.tsa.SARIMAX*), while deep learning models (LSTM/GRU and hybrids) are implemented in **TensorFlow 2** (Keras API). To ensure reproducibility, random seeds were fixed (*NumPy/TensorFlow*), keep all preprocessing strictly time-ordered, and fit normalisers only on the training split. See **Table 1.**

Table 1. Software stack (packages and roles).

Package	Role
NumPy / Pandas	Array \& tabular data processing
SciPy	Numerical routines (stats, optimization)
scikit-learn	Scaling, train/validation split, basic metrics
statsmodels	ARIMA/SARIMA via *sm.tsa.SARIMAX*
TensorFlow 2 (Keras)	LSTM/GRU/CNN model building and training
Matplotlib	Plots (learning curves, forecasts)

4.2 Data Preprocessing

Using daily trading data of Apple Inc. (AAPL) spanning from January 1, 2014 to December 31, 2024. The dataset includes the adjusted closing price, open, high, low, and trading volume, from which additional features are derived.

4.2.1 Target Variable Definition

Both closing prices and log-returns are considered as forecasting targets. Closing price prediction is intuitive and interpretable, reflecting the absolute market level. In contrast, log-returns are closer to stationary after differencing and are widely used in risk modeling and financial applications, but they suffer from a low signal-to-noise ratio, making them more challenging to predict. By examining both targets, this study highlights the relative strengths and limitations of traditional models and deep learning approaches under different forecasting tasks. This paper will pay main attention on log-close prediction, while also attempting to use these models for log-return forecasting.

The primary regression target is the next-day log-transformed closing price, as shown by equation (21). When examining models using metrics, the log-close will be transformed back into closing prices, then calculating RMSE, MAE, R on real price scale instead of log-scale. Then follows the next-day log-return, see equation (22). For the multi-task learning setting, a binary classification label was constructed, indicating the next-day direction of price movement (upward or downward), defined as in equation (23).

$$y_{t+1} = log(Close_{t+1}) \tag{21}$$

$$r_{t+1} = log(\frac{Close_{t+1}}{Close_t}) \tag{22}$$

$$z_{t+1} = \begin{cases} 1, & if\ Close_{t+1} > Close_t \\ 0, & Otherwise \end{cases} \tag{23}$$

4.2.2 Feature Engineering

More than 30 technical indicators were computed, including: Moving Average (MA), Exponential Moving Average (EMA), Relative Strength Index (RSI), Stochastic Oscillator, Bollinger Bands, Average True Range (ATR), Moving Average Convergence Divergence (MACD), On-Balance Volume (OBV). These features are widely used in quantitative finance and provide complementary signals beyond raw prices.

4.2.3 Sliding Window Construction

To transform the dataset into supervised learning samples suitable for model training, sliding windows were essential. **Fig 3** demonstrates this process.

Sliding window serves three purposes: firstly, it transforms variable-length sequences into uniformly shaped training data, facilitating direct learning by models such as CNN1D/LSTM/GRU; secondly, it preserves recent trends, volatility, and local patterns within each sample, enabling models to capture short-to-medium-term dependencies; thirdly, overlapping windows expand the effective sample size. For each window, the feature matrix consists of the previous 40 days of inputs, while the target corresponds to the next-day value.

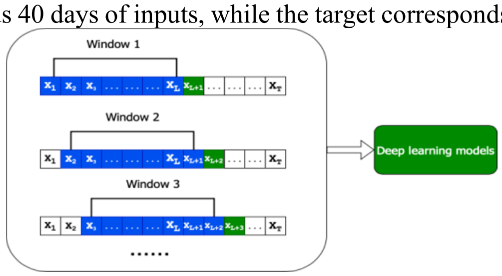

Fig. 3. Sliding window demonstration

4.2.4 Train–Validation–Test Split

This study chronologically split the dataset into three segments to mimic real-world forecasting: Training set: up to December 31, 2021; Validation set: January 1, 2022 – December 31, 2023; Test set: January 1, 2024 – December 31, 2024.

This setup allows the models to be trained on earlier market conditions, validated on recent data for hyperparameter tuning, and tested on the most recent period for out-of-sample evaluation.

4.2.5 Data Normalisation

To prevent data leakage, the data was splited into training, validation, and test sets in chronological order, standardising solely based on training-period statistics. To reduce scale differences across variables, standardizing all features using statistics computed from the training set only. The same normalization parameters are applied to the validation and test sets, ensuring that no future information leaks into training:

$$x' = \frac{x-\mu_{\text{train}}}{\sigma_{\text{train}}} \tag{24}$$

It should be noted that the present study involves a methodological asymmetry. Specifically, the ARIMA model was re-estimated dynamically via walk-forward validation, allowing it to gradually adapt to distributional changes during the test period, while the deep learning models were trained once on a fixed training set and kept static during validation and testing. Although the sliding-window mechanism provides dynamic samples during preprocessing, it does not equate to rolling re-training or fine-tuning in the prediction phase. Consequently, the adaptiveness of ARIMA and deep learning models is not strictly comparable. Due to computational constraints, the current experiment environment was unable to implement dynamic updating for the neural models, which may underestimate their performance under structural breaks. Future work could investigate rolling retraining or incremental fine-tuning for deep learning models to ensure a fairer comparison.

4.3 ARIMA

Autoregressive Integrated Moving Average (ARIMA) is used as the statistical baseline. *Walk-forward:* Real-world series are nonstationary and exhibit concept drift, so parameters fitted once quickly become stale. Walk-forward re-estimates ARIMA on an expanding or sliding window at each forecast origin, using only information available at that time—both mirroring deployment and eliminating look-ahead bias.

Model setting: **Order:** (1,1,1); **Refitting frequency:** every 10 days; **Error distribution:** assumed Gaussian white noise.

4.4 LSTMs and GRU

The first deep learning model is a *single-layer Long Short-Term Memory* (LSTM) network. This model is designed to capture long-term temporal dependencies without introducing excessive capacity, reducing the risk of overfitting. After establishing an LSTM baseline, a *GRU* is additionally evaluated to test whether comparable or superior accuracy can be achieved with fewer parameters and faster convergence. The outputs of both model are projected through a fully connected dense layer with 'linear' activation to predict the next-day log closing price.

Architectures of them are quite similar: **1 *LSTM /GRU layer* with [128] hidden units**; **Dropout**: [0.2]; **Regularization**: L2 penalty of [0.0001]; **Batch size**: [32]; **Optimizer**: Adam(0.001); **Loss**: MSE, Metrics: MAE; **Epochs**: trained up to 200, Early stopping (es) with patience=10, learning rate reduction (rlr) with patience=10, min_lr=1e-5.

To investigate whether deeper architectures improve performance, experiment with a *two-layer stacked LSTM*. While additional layers increase modeling capacity, they may also lead to overfitting when the signal-to-noise ratio is low.

Architecture: **2 *stacked LSTM layers* with [128], [64] hidden units**; **Dropout**: [0.2] between layers; **Batch size**: [32]; **Optimizer**: Adam(0.001); **Loss**: MSE, Metrics: MAE; **Epochs**: trained up to 100 with rlr patience=10, min_lr=1e-5; **Regularization**: L2 penalty of [0.0001].

4.5 Hybrid Models

4.5.1 CNN-LSTM (Stack)

The *CNN-LSTM* model first applies temporal convolution to extract local patterns (e.g. short-term momentum signals) before feeding the resulting feature maps into an LSTM to capture long-term dependencies.

Architecture: **CNN block:** Convolutional layers with [64] filters, kernel size [5], padding='causal'; Activation: ReLU; **LSTM block:** LSTM layer with [128] hidden units; Dropout [0.2]; **Regularization:** Both L2 penalty of [0.0001]; **Optimizer:** Adam(0.001);

 https://doi.org/10.1051/itmconf/20258001021

Loss: MSE, Metrics: MAE; **Training setup:** Batch size [64], Epochs [100], learning rate reduction (rlr) with patience=10, min_lr=1e-5.

4.5.2 LSTM-GRU and CNN-GRU (Concatenation)

LSTM–GRU employs feature-level concatenation to fuse the inductive biases of both gated units: LSTM retains long-range dependencies and slow-evolving trends, whilst GRU exhibits greater sensitivity to short-to-medium-term dynamics with fewer parameters and faster convergence. Introducing *CNN-GRU* at the same time to compare with CNN-LSTM. The CNN–GRU model combines local patterns with temporal memory: the former employs causal 1-D convolutions to extract short-range patterns and burst/shape features, while the latter uses gated memory to aggregate these local cues over longer time spans.

(a) LSTM-GRU: **LSTM layer:** *return_sequences=False*, with [128] hidden units; Dropout [0.2]; Regularization: L2 penalty of [0.0001]. **GRU layer:** *return_sequences=False*, with [64] hidden units. **Optimizer:** Adam(0.001); **Loss:** MSE, Metrics: MAE; **Training setup:** Batch size [64], Epochs [100], learning rate reduction (rlr) with patience=10, min_lr=1e-5.

(b) CNN-GRU: **CNN block:** Convolutional layers with [64] filters, kernel size [5], padding='causal'; Activation: ReLU; **GRU layer:** *return_sequences=False*, with [128] hidden units; **Optimizer:** Adam(0.001); **Loss:** MSE, Metrics: MAE; **Training setup:** Batch size [64], Epochs [100], learning rate reduction (rlr) with patience=10, min_lr=1e-5.

5 Results

Report results for all models on the **test set (2023--2024).** Performance is compared using RMSE, MAE, Pearson correlation.

5.1 Simple models

Table 2 summarizes the performance of the ARIMA(1,1,1) model under the walk-forward scheme, single-layer LSTM and GRU.

Table 2. Predictions of simple model on close price (Test).

Model	RMSE	MAE	R
ARIMA	6.9589	5.12	0.9697
LSTM	12.8887	10.83	0.9779
Stack LSTM	18.9053	15.73	0.9606
GRU	**5.3762**	**4.13**	**0.9790**

Overall, the ARIMA model provides a valid result when predicting the next day's log-close, while GRU outperforms ARIMA and LSTM on all evaluation metrics.

The correlation of single-layer LSTM is 0.9779, meaning that this model is able to capture the trend of close price. However, the value of RMSE is higher than ARIMA and GRU, which indicates that LSTM fails to reliably forecast closing prices during periods of extreme volatility, such as financial crises or pandemics. Also, LSTM models both show a higher MAE value, meaning that the overall robustness of LSTM models were inferior to that of the GRU on this task.

See **Fig 4** for predicted values of ARIMA model. See **Fig 5** for visualisation of Pred vs True closing price among LSTMs and GRU.

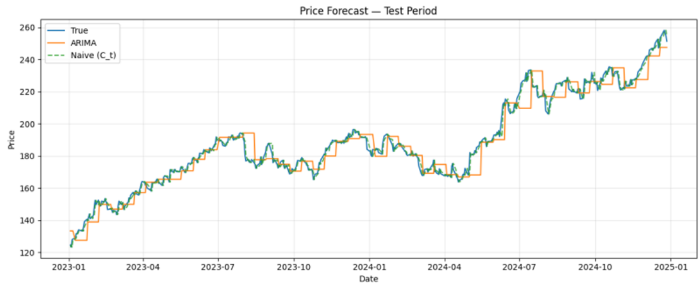

Fig. 4. Predicted vs. actual closing prices (ARIMA).

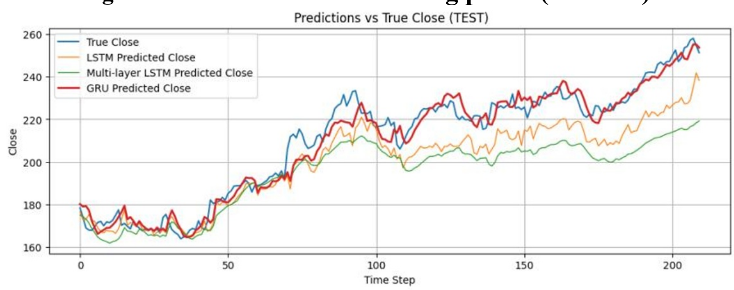

Fig. 5. Predicted vs. True closing prices (Simple models).

5.2 Hybrid models

Table 3 reports the results for the hybrid models.

Table 3. Hybrid model results (Test).

Model	RMSE	MAE	R
CNN-LSTM	22.2936	19.27	0.9529
LSTM-GRU	18.2607	15.55	0.9301
CNN-GRU	12.7640	10.94	0.9465

The performance of hybrid and multi-layer models is significantly worse than simple GRU models. See **Fig 6**. This figure also shows that all hybrid models exhibited varying degrees of bias during prediction, particularly towards the latter end of the test set.

5.3 Comparative Summary

Table 4 consolidates all models for direct comparison.

Table 4. Hybrid model results (Test).

Model	RMSE	MAE	R
ARIMA	6.9589	5.12	0.9697
LSTM	12.8887	10.83	0.9779
Stack LSTM	18.9053	15.73	0.9606
GRU	**5.3762**	**4.13**	**0.9790**
CNN-LSTM	22.2936	19.27	0.9529
LSTM-GRU	18.2607	15.55	0.9301
CNN-GRU	12.7640	10.94	0.9465

Findings: The traditional ARIMA model exhibits robustness across absolute error metrics such as RMSE and MAE, indicating its advantage in mitigating forecasting biases during days of substantial volatility. Among deep learning models, the single-layer GRU outperforms ARIMA across all metrics, suggesting superior relative forecasting efficacy on most routine trading days and during extreme volatility episodes, whilst better capturing price trend dynamics. However, more complex models not only failed to enhance performance but also underperformed the single-layer model on both validation and test sets. This finding highlights the peculiarities of financial time series: in environments dominated by noise and limited effective samples, simpler models often achieve more robust predictions, while complex architectures do not necessarily yield substantial benefits.

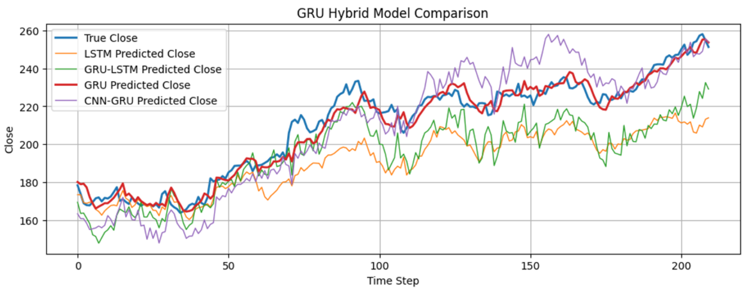

Fig. 6. Comparison of hybrid models and GRU.

6 Discussion

6.1 Why Simple Models Perform Better

The GRU model consistently achieved the best performance across most metrics. This suggests that the model capacity is well-matched to the information content of the dataset.

The failure of more complex models may be driven by several possible reasons:

(1) *Data characteristics---Weak signals and noise-dominated:* The daily price sequence for AAPL exhibits extremely limited genuinely predictable patterns. A simple single-layer GRU model possesses moderate capacity, capable of capturing primary autocorrelations and trends. Complex hybrid models (CNN-GRU, LSTM-GRU) instead tend to 'overfit' the noise in the training set, leading to degraded generalisation performance during validation/testing.

(2) *Insufficient Sample Size:* 2014–2024 daily frequency data contains about 3,000-4,000 points; valid sample size after sliding windowing is about 2–3 thousand. For models like CNN-GRU with larger parameter counts, this data volume is insufficient to support complex architectures, making overfitting likely.

(3) *Task characteristics---Short-term dependencies dominated:* Stock price forecasting relies primarily on short-term linear/weakly non-linear correlations. The 'long memory' capability of GRU/LSTM does not truly demonstrate advantages in such high-noise tasks. While CNN layers prove highly effective in image tasks, they may not capture more valuable local patterns than GRUs when applied to one-dimensional financial time series.

6.2 Close VS Return

This study is primarily focused on predicting closing prices, conducting only exploratory analysis of log returns in Discussion section. In fact, the study also tried the same models in close forecasting on log-return prediction, while also trained a multi-task learning model to attempt for better outcomes. Hit ratio was introduced as a metric to evaluate the models, which denotes the proportion of samples where the predicted return sign matches the actual return sign, reflecting directional accuracy (>50% indicates superiority to random performance).

Results on return forecasting: Experimental results corroborate this: both ARIMA and deep learning models performed near random benchmarks in yield forecasting, with correlation metrics (R) approaching zero and hit ratios stabilising around 50–56%, failing to demonstrate substantive predictive capability. The best outcomes were generated by multi-task LSTM model, with classification hit ratio 56.29% and RMSE, MAE values that are close to naive baseline. See **Fig 7** for predicted results of multi-task LSTM. This indicates that within the current framework, both ARIMA and deep learning models have been unable to effectively extract predictable patterns from returns, further underscoring the noise-dominated nature of financial markets. Consequently, return forecasting is regarded as a limitation of current models and a direction for future research, rather than the core contribution of this paper.

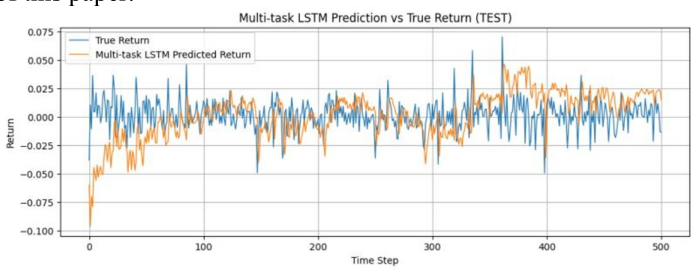

Fig. 7. Multi-task LSTM Predicted vs. True Return (TEST).

6.3 Consistency with Prior Literature

Our findings are consistent with previous studies reporting that ***simpler models often outperform complex ones*** in financial forecasting tasks (e.g., Sezer et al., 2020). This indicates that increasing model complexity does not guarantee better results in domains where the signal-to-noise ratio is inherently low. Future research should therefore carefully consider the trade-off between model expressiveness and robustness, especially under limited data conditions.

7 Conclusion and Future Work

This paper systematically compared classical statistical methods (ARIMA) with several deep learning architectures (single-layer LSTM, multi-layer LSTM, GRU, CNN-LSTM, LSTM-GRU, CNN-GRU) for stock price forecasting. Using ten years of daily AAPL data, the study evaluated models on regression accuracy and directional prediction with metrics including RMSE, MAE, Pearson correlation.

7.1 Key findings

(1) The GRU consistently outperformed both ARIMA and more complex neural architectures, achieving the most stable performance across all metrics. (Even without introducing walk-forward validation, GRU outperforms ARIMA.); (2) Due to the exponential increase in the number of parameters and limitations in training data, LSTM, multi-layer LSTM and hybrid models tended to perform poorly in close price forecasting task, providing little to no improvement over simpler baselines; (3) Multi-task learning improved classification accuracy on log-return prediction, but has yet to yield a stable and satisfactory outcome, highlighting the unpredictability of financial markets and the limitations of current models.

7.2 Limitations and Future work

This study has several limitations:

(1) Focus on a single asset (AAPL), limiting generalizability to broader markets. (2) Only daily-frequency data were used, which may omit valuable dynamics. (3) The target was next-day log price; return prediction remains an open challenge. (4) The issue of distribution bias between training and test data arising from real-world market fluctuations remains unresolved.

Future research could extend this work in several directions:

(1) Applying models to **multiple assets** to assess cross-sectional generalization. Incorporating **higher-frequency data** (e.g., intraday) to better capture short-term dynamics. (2) Exploring **advanced architectures** such as Transformers and attention mechanisms for capturing long-range dependencies. Developing **multi-task or hybrid frameworks** that jointly predict returns, volatility, and trends in a more balanced manner. (3) Where computational resources permit, introducing walk-forward training methodology to deep learning models to ensure fairness.

Overall, our findings emphasize the importance of aligning model complexity with the intrinsic information content of financial data, and they highlight promising avenues for future exploration.

References

1. LeCun, Y et al., Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539
2. M. T. Hagan et al., Neural Network Design. Boston, MA, USA: PWS, 1996.
3. Werbos, P. J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences (Ph.D. thesis). Harvard University.
4. Rumelhart et al., Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0
5. S. Hochreiter et al., 'Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,' In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.
6. S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735
7. Cho, K. et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014. https://doi.org/10.48550/arXiv.1406.1078
8. G. E. P. Box et al., Time Series Analysis: Forecasting and Control, 5th ed. Hoboken, NJ, USA: Wiley, 2015.
9. R. Cont, "Empirical properties of asset returns: Stylized facts and statistical issues," Quantitative Finance, vol. 1, no. 2, pp. 223–236, 2001. https://doi.org/10.1080/713665670
10. R. F. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation," Econometrica, vol. 50, no. 4, pp. 987–1007, 1982. https://doi.org/10.2307/1912773
11. T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," Journal of Econometrics, vol. 31, no. 3, pp. 307–327, 1986. https://link.springer.com/article/10.1007/BF02614309?
12. D. M. Q. Nelson et al.,, "Stock market's price movement prediction with LSTM neural networks," in Proc. IJCNN, 2017, pp. 1419–1426. http://dx.doi.org/10.1109/IJCNN.2017.7966019

13. T. Fischer and C. Krauss, European Journal of Operational Research, vol. 270, no. 2, pp. 654–669, 2018. doi: 10.1016/j.ejor.2017.11.054
14. S.-H. Ko, G. Lee, and J.-W. Lee, “Investing with the news: Sentiment-augmented LSTM for stock price prediction,” PLOS ONE, vol. 16, no. 4, e0247893, 2021.
15. G. P. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model,” Neurocomputing, vol. 50, pp. 159–175, Jan. 2003. https://doi.org/10.1016/S0925-2312(01)00702-0
16. X. Jiang et al., “An attention GRU-XGBoost model for stock price prediction,” in Proc. ICCET, 2022, pp. 61–67. https://doi.org/10.1145/3573834.3573837
17. T. B. Shahi et al., “Stock price forecasting with deep learning: A comparative study,” Mathematics, vol. 8, no. 9, p. 1441, 2020. https://doi.org/10.3390/math8091441
18. O. B. Sezer et al., “Financial time series forecasting with deep learning: A systematic literature review: 2005–2019,” Applied Soft Computing, vol. 90, 2020, 106181. https://doi.org/10.1016/j.asoc.2020.106181