Journal of Mathematics and Modeling in Finance (JMMF)
Vol. 5, No. 1, Winter & Spring 2025
Research paper

Comparing the performance of different deep learning architectures for time series forecasting

Reza Taleblou[1]

[1] Faculty of Economics, Allameh Tabatabai University, Tehran, Iran
talebloo.r@atu.ac.ir

Abstract:
In this paper, we evaluate the performance of two machine learning architectures—Recurrent Neural Networks (RNN) and Transformer-based models—on four commodity-based company indices from the Tehran Stock Exchange. The Transformer-based models used in this study include AutoFormer, FEDformer, Informer, and PatchTST, while the RNN-based models consist of GRU and LSTM. The dataset comprises daily observations collected from April 20, 2020, to November 20, 2024. To enhance the generalization power of the models and prevent overfitting, we employ two techniques: splitting the training and test samples, and applying regularization methods such as dropout. Hyperparameters for all models were selected using a visual method. Our results indicate that the PatchTST model outperforms other methods in terms of Root Mean Squared Error (RMSE) for both 1-day and 5-day (1-week) forecasting horizons. The FEDformer model also demonstrates promising performance, particularly for forecasting the MetalOre time series. In contrast, the AutoFormer model performs relatively poorly for longer forecasting horizons, while the GRU and LSTM models yield mixed results. These findings underscore the significant impact of model selection and forecasting horizon on the accuracy of time series forecasts, emphasizing the importance of careful model choice and hyperparameter tuning for achieving optimal performance.

Keywords: Financial Time Series Forecasting, Deep Learning, Recurrent Neural Network(RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit(GRU), Transformer Architecture
Classification: C22, C45,C53, G17

1 Introduction

Time series forecasting is a critical task across diverse fields such as finance, economics, and industry, as accurate predictions of future values can profoundly impact decision-making, risk management, and resource allocation. While traditional statistical and econometric methods have long been foundational for this purpose, their reliance on assumptions like linearity and stationarity—often violated in real-world scenarios—limits their practical effectiveness. In recent years, machine learning techniques, particularly deep learning approaches, have gained traction for time series forecasting. These methods excel at capturing complex nonlinear patterns

[1]Corresponding author
Received: 14/12/2024 Accepted: 09/03/2025
https://doi.org/10.22054/JMMF.2025.83410.1157

and dynamic relationships in data, overcoming many constraints of conventional models and delivering superior predictive performance.

The objective of this paper is to utilize novel deep learning methods to estimate the expected return of multiple assets in financial markets. The models presented in this paper enable the simultaneous prediction of several variables using a single method. In this regard, two important points are raised:

Data Selection and Complexity: What data should be used to demonstrate this capability? In this paper, solely to avoid cluttered diagrams, four indices have been selected from over 32 significant stock market indices of the Tehran Stock Exchange. This data possesses several characteristics. First, it pertains to commodity-based companies, and a wide range of global variables influence the sales of these companies and, consequently, their stock prices. Therefore, the indices used for prediction in this paper are effectively exposed to shocks that commodity-based companies face anywhere else in the world. Second, exchange rate shocks and currency jumps directly impact these companies, increasing the complexity of predicting these variables due to a wide array of factors (including Iranian sanctions and geopolitical risks, etc.). Third, unforeseen shocks, such as changes in regulations regarding royalty payments for these companies or export and import regulations, further complicate the price behavior of these companies and their related indices examined in this paper. Consequently, the indices used for prediction in this paper represent some of the most complex data for forecasting.

Scalable Deep Learning Architectures for Multivariate Forecasting: Are there deep learning models for time series prediction that can be used simultaneously for multiple variables on a large scale? This question can be elaborated as follows: Can a model be introduced that can be trained on a wide range of time series data and be used individually for each of them? Here, two important architectures from deep learning models for time series prediction are introduced. While models in the RNN family have interesting capabilities for this task, they have a disadvantage: they perform prediction using a single equation, meaning they do not utilize information from other variables when predicting one specific variable. On the other hand, some models with transformer architectures have the capability to consider predictions as simultaneous equations and utilize information from other variables when predicting each variable.

In this study, we evaluate the performance of different machine learning architectures, transformer and RNN, on four different commodity-based company indices in Tehran stock exchange. The transformer architecture models used in this study include AutoFormer, FEDformer, Informer, and PatchTST, while the RNN architecture models include GRU and LSTM. Our goal is to investigate the performance of these models on time series forecasting tasks, and to identify the most efficient model for each forecasting horizon.

The data used in this study was collected daily from April 20, 2020, to Nov 20, 2024, and includes four different commodity-based companies index in Tehran stock

exchange. 80 percent of data was used for training (from April 20, 2020 to Feb 1, 2024) and 20 percent of data was used for testing (from Feb 1, 2024 to 20 Nov 2024). Finally, the performance of each model across different forecasting horizons was evaluated using the root mean squared error (RMSE) criterion.

The remainder of this paper is organized as follows. In Section 2, we review popular techniques used for time series forecasting, including traditional statistical and econometrics methods and deep learning approaches with emphasis on RNN and transformer architectures. In Section 3, we describe the data used in this study, and discuss the experimental setup, the results, and implications of our findings. Finally, in Section 4, we conclude our study and suggest directions for future research.

2 Time Series Forecasting methods

In the framework of classical portfolio theory, two sets of inputs are required to construct a portfolio of financial assets. The first is the variance-covariance matrix, which indicates the risk of each asset and their correlation. The second is the mean equation, which represents the expected return of each asset over different horizons. The first, the variance-covariance matrix, is typically modeled using two methods: the MGARCH family of models [34] and [35], or the Multivariate Stochastic Volatility (MSV) model [37] and [36]. These models are suitable for both high and low-frequency data. However, the second, constructing the mean equation of returns that can accurately model the expected return, is typically modeled using econometric time series tools.

Traditional econometric models for time series forecasting include Autoregressive integrated moving average (ARIMA), exponential smoothing (ES), and seasonal decomposition models. The simplicity and interpretability of these models have led to wide use for time series forecasting tasks [40] and [41]. ARIMA and ES models are a popular choice for time series forecasting, as they can capture both short-term and long-term dependencies in the data. However, ARIMA models have limitations, such as the assumption of linearity and stationarity, which may not hold in many real-world applications. Also ES models face limitations, such as the assumption of a constant trend and seasonality, which similarly may not hold in many real-world applications. Seasonal decomposition models are used to decompose a time series into its trend, seasonal, and residual components. These models can be used to identify and remove seasonality from the data, which can improve the accuracy of time series forecasting. This method is mainly suitable for low-frequency data but may not be effective for high-frequency data or when the number of variables is large. Therefore, as an alternative, machine learning and neural network methods are used.

Neural network methods like Deep learning approaches have become increasingly popular for time series forecasting tasks, due to their ability to learn complex

patterns and relationships in data [39]. Recurrent neural networks (RNNs) and transformers are two popular deep learning architectures used for time series forecasting.

RNNs are a type of neural network that can learn long-term dependencies in data. RNNs have been widely applied for time series forecasting tasks, due to their ability to capture both short-term and long-term dependencies in the data.

Transformers are a popular type of neural network that can learn complex in data. Transformers have been widely used for natural language processing tasks, but have also been used to time series forecasting tasks. In this regard, in this paper, several methods from two different neural network architectures have been compared.

2.1 RNN Architecture

Recurrent Neural Networks (RNNs) are a category of artificial neural networks designed to process sequential data by utilizing internal memory states. This capability allows RNNs to remember previous information and utilize it to make future forecasting, making them ideal for tasks such as natural language processing, speech recognition, and time series predictions [21] [23]. The concept of RNNs dates back to the 1980s, when researchers Rumelhart, Hinton, and Williams introduced networks with internal memory states in 1986, laying the foundation for modern RNN architectures [21].

A significant advancement in RNNs came with the description of Long Short-Term Memory (LSTM) networks by Hochreiter et al. [23]. LSTMs addressed the vanishing gradient problem that plagued early RNNs, enabling them to learn long-range dependencies in sequential data [23]. Gated Recurrent Units (GRU) were proposed by Cho et al. as a simpler alternative to LSTMs [22] . GRUs also mitigate the vanishing gradient problem but with fewer parameters than LSTMs, making them computationally more efficient [22] .

At the core of RNNs, including their LSTM and GRU variants, is their distinctive architecture featuring internal loops. These loops facilitate the continuous flow of information as the network processes data, allowing RNNs to maintain a hidden state that acts as a memory bank. This mechanism effectively integrates current input with accumulated knowledge from previous inputs [22]. Thanks to this design, RNNs, along with their advanced forms like LSTMs and GRUs, have become indispensable in various applications. They excel in natural language processing, speech recognition, machine translation, and predictive text applications, demonstrating their proficiency in handling sequential data.

LSTM Architecture

The LSTM uses a multilayer LSTM encoder and an MLP decoder [23]. It builds upon the LSTM-cell that improves the exploding and vanishing gradients problems

of classic RNNs. This network has been extensively used in sequential forecasting tasks like language modeling, and time series predictions. The forecasts are obtained by transforming the hidden states into contexts $\mathbf{c}_{[t+1:t+H]}$, that are decoded and adapted into $\hat{\mathbf{y}}_{[t+1:t+H],[q]}$ through MLPs.

$$\mathbf{h}_t = \text{LSTM}([\mathbf{y}_t, \mathbf{x}_t^{(h)}, \mathbf{x}^{(s)}], \mathbf{h}_{t-1}) \tag{1}$$

$$\mathbf{c}_{[t+1:t+H]} = \text{Linear}([\mathbf{h}_t, \mathbf{x}_{[:t+H]}^{(f)}]) \tag{2}$$

$$\hat{y}_{\tau,[q]} = \text{MLP}([\mathbf{c}_\tau, \mathbf{x}_\tau^{(f)}]) \tag{3}$$

where $\mathbf{h}_t$, is the hidden state for time t, $\mathbf{y}_t$ is the input at time t and $\mathbf{h}_{t-1}$ is the hidden state of the previous layer at $t-1$, $\mathbf{x}^{(s)}$ are all static exogenous inputs, $\mathbf{x}_t^{(h)}$ historic exogenous, $\mathbf{x}_{[:t+H]}^{(f)}$ are future exogenous available at the forecasting time.

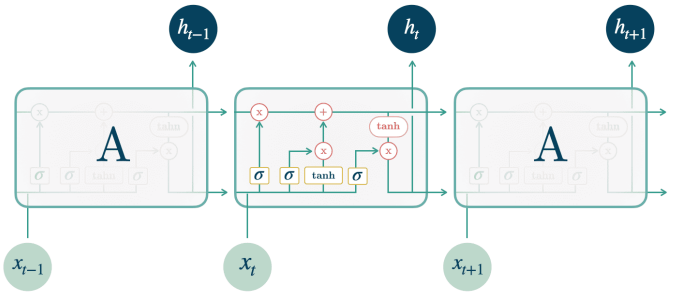


Figure 1: LSTM model architecture [23].

GRU

the Gated Recurrent Unit (GRU) proposed to improve on LSTM and RNN. The forecasts at each time are given by a MLP decoder. This architecture follows closely the original Multi Layer RNN with the main difference being its use of the GRU cells [22] . The predictions are obtained by transforming the hidden states into contexts $\mathbf{c}_{[t+1:t+H]}$, that are decoded and adapted into $\hat{\mathbf{y}}_{[t+1:t+H],[q]}$ through MLPs.

$$\mathbf{h}_t = \text{GRU}([\mathbf{y}_t, \mathbf{x}_t^{(h)}, \mathbf{x}^{(s)}], \mathbf{h}_{t-1}) \tag{4}$$

$$\mathbf{c}_{[t+1:t+H]} = \text{Linear}([\mathbf{h}_t, \mathbf{x}_{[:t+H]}^{(f)}]) \tag{5}$$

$$\hat{y}_{\tau,[q]} = \text{MLP}([\mathbf{c}_\tau, \mathbf{x}_\tau^{(f)}]) \tag{6}$$

where $\mathbf{h}_t$, is the hidden state for time t, $\mathbf{y}_t$ is the input at time t and $\mathbf{h}_{t-1}$ is the hidden state of the previous layer at $t-1$, $\mathbf{x}^{(s)}$ are all static exogenous inputs, $\mathbf{x}_t^{(h)}$ historic exogenous, $\mathbf{x}_{[:t+H]}^{(f)}$ are future exogenous available at the forecasting time.

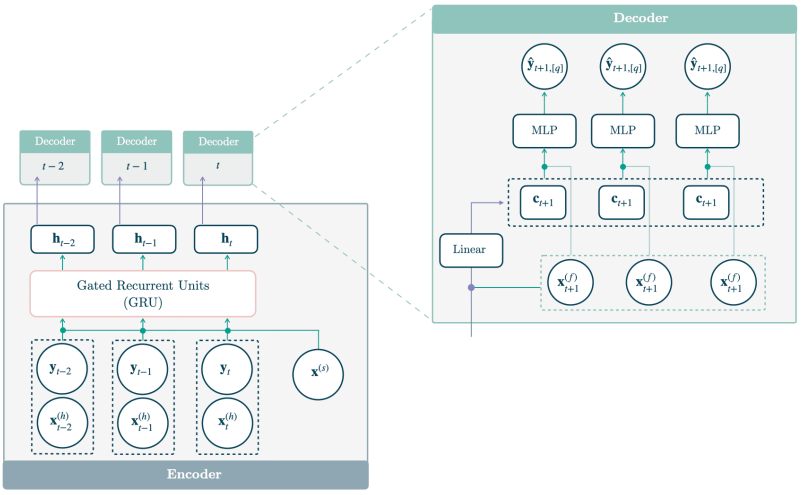


Figure 2: GRU architecture

2.2 Transformer Architecture

The Transformer innovation in deep learning [2] has recently gained significant attention due to its excellent performance in natural language processing (NLP) [3], computer vision (CV) [4], and speech processing [5]. The Transformer architecture structure based on [2] is showed in figure 3.

In the past few years, numerous Transformers have been introduced to advance the state-of-the-art in various tasks. Transformers have demonstrated excellent modeling capabilities for long-term dependencies, making them attractive for time series forecasting. Various types of Transformers have been proposed to address specific challenges in time series modeling and have been successfully used to diverse time series tasks, such as forecasting [6], [16]), anomaly detection [13], [9], and classification [15], [14] . Specifically, seasonality or periodicity is one of the important characteristics of time series [10] . How to efficiently model both long-term and short-term temporal dependencies simultaneously remains a challenge [10], [11] . Several comprehensive studies on deep learning for time series, including forecasting, have been conducted [6],

The Transformer architecture is a type of neural network that is widely applied in NLP and other machine learning tasks. In the context of time series forecasting, the Transformer architecture is used to forecast the future values of a time series based on its past values.

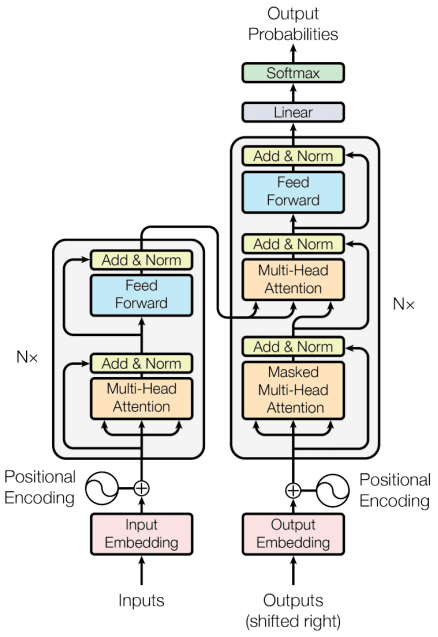


Figure 3: The Transformer architecture based on [2]

The encoder is the first component of the Transformer architecture. It takes in the input time series data and converts it into a sequence of vectors, where each vector represents a single element of the time series.
The self-attention mechanism is a key component of the Transformer architecture. It allows the model to weigh the importance of each element in the input sequence and focus on the most relevant elements when making predictions.
The decoder is the second component of the Transformer architecture. It takes in the output of the encoder and generates a sequence of vectors that represent the predicted future values of a time series. The output of a Transformer architecture is the predicted future values of the time series.
The Transformer architecture is trained using supervised learning, where the model is trained on a dataset of input-output pairs. The model learns to predict the output based on the input by minimizing the error between the predicted output and the actual output.
The Transformer architecture has several advantages that make it a popular choice for time series forecasting tasks. The Transformer architecture is more interpretable than other neural network architectures, making it easier to understand how the model predicts. It is also flexible and can be used for a wide range of tasks, including time series predictions, language translation, and image recognition. The Transformer architecture can achieve high accuracy in time series forecasting tasks,

especially when compared to other neural network architectures.
The conventional self-attention framework is based on the [2] and is defined as follows:

Query: The query is a vector that represents the input element.

Key: The key is a vector that represents the input element.

Value: The value is a vector that represents the input element.

Weighted Sum: The weighted sum is calculated by multiplying the query and key vectors and summing the result.

The matrix formulation of the Transformer architecture can be written as:

$$H = \sigma(WhX + bh) \tag{7}$$

where H is the output of a Transformer architecture, σ is the activation function, Wh is the weight matrix, X is the input element, and bh is the bias vector.
The attention weights can be calculated using the following formula:

$$A = softmax(QK^T/\sqrt(d)) \tag{8}$$

where A is the attention weight matrix, Q is the query vector, K is the key vector, T is the transpose operator, and d is the dimensionality of the input element.
The output of the Transformer architecture can be calculated using the following formula:

$$Y = AV \tag{9}$$

where Y is the output of the Transformer architecture, A is the attention weight matrix, and V is the value vector.
Transformer architectures have become increasingly popular for time series forecasting tasks, due to their ability to learn complex patterns and relationships in data. AutoFormer, FEDformer, Informer, and PatchTST are four popular transformer architectures applied for time series forecasting that use a self-attention mechanism to learn long-term dependencies in data. They have been shown to outperform traditional econometrics methods and other deep learning approaches on time series forecasting tasks.

Autoformer

The Autoformer model tackles the challenge of finding reliable dependencies on intricate temporal patterns of long-horizon forecasting [20] . The architecture has the following distinctive features:

- In-built progressive decomposition in trend and seasonal components based on a moving average filter.

- Auto-Correlation mechanism that discovers the period-based dependencies by calculating the auto-correlation and aggregating similar sub-series based on the periodicity.
- Classic encoder-decoder proposed by [2] with a multi-head attention mechanism.
- The Autoformer model utilizes a three-component approach to define its embedding:

  - It employs encoded autoregressive features obtained from a convolution network.

  - Absolute positional embeddings obtained from calendar features are utilized.

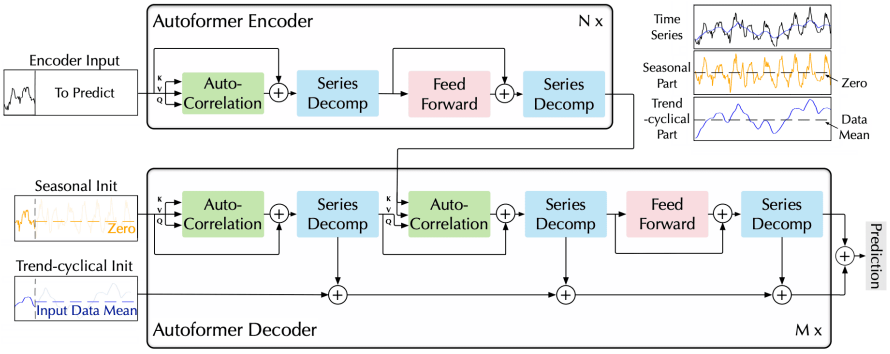


Figure 4: Autoformer architecture based on [20]

FEDformer

The FEDformer model tackles the challenge of finding reliable dependencies on intricate temporal patterns of long-horizon forecasting [19]. The architecture has the following distinctive features:

- In-built progressive decomposition in trend and seasonal components based on a moving average filter.
- Frequency Enhanced Block and Frequency Enhanced Attention to perform attention in the sparse representation on basis such as Fourier transform.
- Classic encoder-decoder proposed by [2] with a multi-head attention mechanism.

The FEDformer model utilizes a three-component approach to define its embedding:
- It employs encoded autoregressive features obtained from a convolution network.
- Absolute positional embeddings obtained from calendar features are utilized.

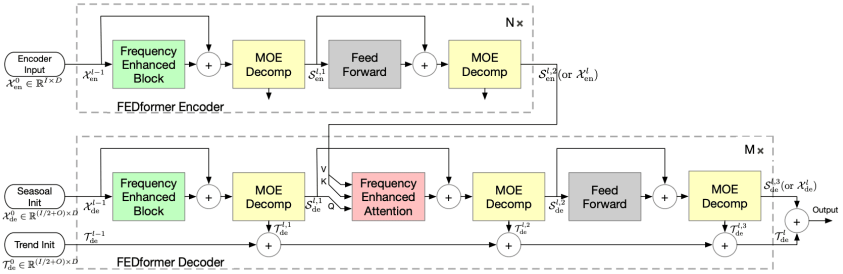


Figure 5: FEDformer architecture based on [19]

Informer

The Informer model tackles the vanilla Transformer computational complexity challenges for long-horizon forecasting [17].

The architecture has three distinctive features: - A Prob-Sparse self-attention mechanism with an O time and memory complexity Llog(L). - A self-attention distilling process that prioritizes attention and efficiently handles long input sequences. - An MLP multi-step decoder that predicts long time-series sequences in a single forward operation rather than step-by-step. The Informer model utilizes a three-

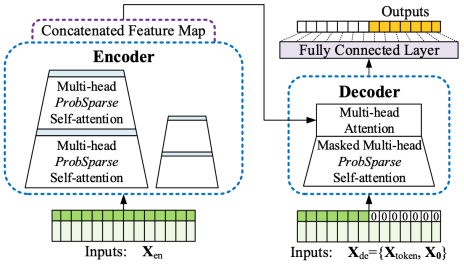


Figure 6: Informer architecture based on [17]

component approach to define its embedding: - It employs encoded autoregressive features obtained from a convolution network. - It uses window-relative positional embeddings derived from harmonic functions. - Absolute positional embeddings obtained from calendar features are utilized.

PatchTST

The PatchTST (Patch Time Series Transformer) model is a recent innovation in the field of time series forecasting, proposed by [18]. The PatchTST model leverages the Transformer architecture for efficient time series forecasting. in general, Transformer Architecture is designed for NLP and LLM [2]. this models in align with prior paper based on Transformers (like Informer, FEDformer, Autoformer)

want to develop a model for capturing long dependency for multivariate Time Series forecasting. this goal is achieved with two key components:Patching and Channel-Independence. Patching: The input time series is segmented into subseries-level patches, which serve as input tokens to the Transformer. This patching design retains local semantic information, reduces computational and memory usage of attention maps, and allows the model to attend to longer historical data

Channel-Independence: Each channel contains a single univariate time series that shares the same embedding and Transformer weights across all series. This approach simplifies the model and enhances its generalization ability.

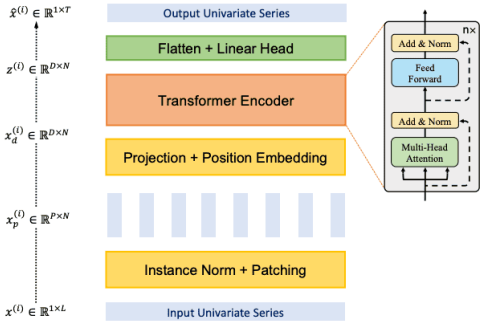


Figure 7: PatchTST backbone for supervised learning based on [18]

The PatchTST model has demonstrated state-of-the-art performance in long-term forecasting tasks. Its patching design and channel-independence allow for efficient computation and retention of local semantic information, making it particularly effective for multivariate time series forecasting. The model also supports self-supervised pre-training and fine-tuning, which enhances its performance on various datasets [26] .

The PatchTST model represents a significant advancement in time series forecasting by leveraging the strengths of Transformer architecture. Its innovative patching design and channel-independence makes it a robust and efficient choice for handling complex time series data.

Patching: To enable a transformer model to handle long-term time series forecasting, this model proposes a patching mechanism to divide the input sequence into smaller segments, called patches. Each patch is a contiguous sub-sequence of the input sequence. The patches are then fed into the transformer encoder.

The patching mechanism allows the transformer model to capture both short-term and long-term dependencies in the time series data. The patch size p controls the trade-off between the two. A smaller patch size captures more short-term dependencies, while a larger patch size allow us to captures more long-term dependencies.

Given a set of multivariate time series samples, each with a time window of length L: $(x_1, ..., x_L)$, where each x_t at time step t is a vector of dimension M. This

sample resembles an L-length lagged data sequence as found in econometrics and statistics, $(x_1, x_2, ..., x_{L-2}, x_{L-1}, x_L)$. for feeding the model first we use a set of data for each single variable i like $x_{1:L}^{(i)} = (x_1^{(i)}, ..., x_L^{(i)})$, where $i = 1, ..., M$ is the number of series. then for $horizon = T$ with this sample we want predict:

$$\hat{x}_{L:L+h}^{(i)} = (\hat{x}_{L+1}^{(i)}, ..., \hat{x}_{L+T}^{(i)}) \in \mathbb{R}^{1\times T}$$

Transformer Encoder: we define $W_p \in \mathbb{R}^{P\times P}$ as trainable linear projection matrix and $W_{pos} \in \mathbb{R}^{D\times N}$ as a learnable additive position encoding matrix for maping the patches to transformer latent space of dimension D. now we monitor the temporal order of patches : $x_d^{(i)} = W_p x_p^{(i)} + W_{pos}$. where $x_d^{(i)} \in \mathbb{R}^{D\times N}$ denote the input that with be fed into transformer encoder.

For each head: $h = 1, \ldots, H$, in multi-head attention this $x_d^{(i)}$ will be transformed into the query, key, and value matrices.

$$\mathbf{Q}_h^{(i)} = (\mathbf{x}_d^{(i)})^T \mathbf{W}_h^Q$$

$$\mathbf{K}_h^{(i)} = (\mathbf{x}_d^{(i)})^T \mathbf{W}_h^K$$

$$\mathbf{V}_h^{(i)} = (\mathbf{x}_d^{(i)})^T \mathbf{W}_h^V$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices, respectively, and d is the dimensionality of the input sequence $\mathbf{W}_h^Q, \mathbf{W}_h^K \in \mathbb{R}^{D\times d_K}$ and $\mathbf{W}_h^V \in \mathbb{R}^{D\times D}$

In next stage, scaled production is used for producing attention output $\mathbf{Q}_h^{(i)} \in \mathbb{R}^{D\times N}$:

$$(\mathbf{O}_h^{(i)})^T = \text{Attention}(\mathbf{Q}_h^{(i)}, \mathbf{K}_h^{(i)}, \mathbf{V}_h^{(i)}) = \text{softmax}\left(\frac{\mathbf{Q}_h^{(i)} \mathbf{K}_h^{(i)T}}{\sqrt{d}}\right) \mathbf{V}_h^{(i)} \tag{10}$$

in last step, a flatten layer with linear head is used and the prediction results $\hat{x}_{L:L+h}^{(i)} = (\hat{x}_{L+1}^{(i)}, ..., \hat{x}_{L+T}^{(i)}) \in \mathbb{R}^{1\times T}$ will be obtained.

loss function for this optimization in base paper is MSE:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \mathbf{E}_x \frac{1}{M} \sum_{i=1}^{M} \|\hat{x}_{L+1:L+T}^{(i)} - x_{L+1:L+T}^{(i)}\|_2^2 \tag{11}$$

We can use the root mean squared error (RMSE) as the loss function to train the model.

2.3 Root Mean Squared Error - RMSE

In this paper, RMSE is used to evaluate the deviation between the predicted results and the observed values. Small values for this criterion indicate higher accuracy for the predication model.

$$\text{RMSE}(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau) = \sqrt{\frac{1}{H}\sum_{\tau=t+1}^{t+H}(y_\tau - \hat{y}_\tau)^2} \tag{12}$$

Parameters: y = Actual values, $\hat{y}$ = Predicted values

2.4 Hyper-parameter Selection

Deep learning models are the most advanced models for time series forecasting. They have outperformed statistical and tree-based methods in recent large-scale competitions, such as the M series, and are increasingly being used in industry. However, their performance is heavily affected by the selection of hyper-parameters. Selecting the optimal configuration, a process known as hyper-parameter tuning, is vital to achieve the best performance. hyper-parameters like the dropout probability (p) or learning rate (η) are common to several models.
There are several Techniques for Common Hyper-parameter Optimization including: Grid Search, Random Search, Bayesian Optimization, Population-Based Methods and Gradient-Based Optimization. [29] [30] [31] [32]. [33].

Hyper-parameter tuning consists of four steps: First, defining the training and test sets. Second, determining the search space. Third, sampling configurations with a search algorithm, training models, and evaluating them on the test set. Fourth, select and save the best model.

2.5 Dropout

Dropout is a regularization technique used to prevent over-fitting in neural networks. It works by randomly setting a fraction of the model's weights to zero during training, encouraging the model to learn stronger and more independent features. This allows the network to generalize better to new and unseen data. During inference, dropout is disabled. This means using a collection of trained sub-networks, which leads to better model performance [38].

2.6 Cross-Validation

It is a method for evaluating how well a model would have performed on historical data. Running cross-validation on historical data involves determining multiple hyper-parameters. This method is performed by determining a sliding window on past observations and forecasting the next period. This method differs from standard cross-validation because it preserves the temporal order of the data instead of splitting it randomly. This method allows us to get a better estimate of our model's predictive capabilities by considering multiple periods. When only one window is used, it is similar to the standard train-test split, where the test data is the last set of observations and the training set includes the previous data.

2.7 Time Series Cross-Validation

Time series cross-validation is a method for evaluating how well a model would have performed on historical data. It is similar to standard cross-validation, but it preserves the temporal order of the data instead of splitting it randomly.In time series cross-validation, we define a window of past observations and predict the next period. We then move the window forward in time and repeat the process. This allows us to evaluate our model's performance on multiple periods and get a better estimate of its predictive capabilities. Time series cross-validation is a more realistic evaluation of our model's performance because it takes into account the temporal dependencies in the data [27], [28]. cross-validation process consists of 4 steps:

Step 1: Determine the prediction horizon: we should determine the prediction horizon, which is the time period that we want to predict. This is defined based on the frequency of the data. In this case, the data is daily, so we can choose a prediction horizon of 5 days (h=5) for a week .

Step 2: Determine the window size: the window size is the length of the data that we will use to make predictions. We can choose any multiple of the prediction horizon as the window size. For example, if we want to predict 5 days ahead.

Step 3: Determine the step size: The step size is the length of time that we will move the window forward in each iteration. In this paper, the step size is set to 5 days.

Step 4: Determine the number of windows: The final step is to define the number of windows that we will use for prediction. In this article, the number of windows is set to 1 to 5.

3 Forecasting 4 commodities based Stock Index

In this study, 4 series is used that was collected daily from April 20, 2020, to Nov 20, 2024, and includes four different commodity-based companies indices in Tehran stock exchange. In order to examine a more accurate comparison of the predictive efficiency of these models, four time series with severe fluctuations in the Tehran Stock Exchange have been used. all series is related to commodities based industries: petrochemical, metal and metalOre, Othermine. this data is plotted in figure 8.

In this paper, two techniques were used for generalization and to prevent overfitting. First, 80 percent of the data was used for training (from April 20, 2020, to Feb 1, 2024), and 20 percent of the data was used for testing (from Feb 1, 2024, to Nov 20, 2024). Using 80 percent of the data, the hyperparameters were tuned, and the remaining 20 percent of the out-of-sample data was used to measure the model's performance.

Furthermore, the dropout technique was used in this paper to control the overfitting of the models. Dropout is a regularization technique where, during training, randomly selected neurons are "dropped out" (i.e., temporarily removed) from the network with a certain probability. The dropout probability is usually denoted as (p): The probability of dropping out a neuron. For example, if (p = 0.3), each neuron has a 30% chance of being dropped out during training. therefore, using the dropout parameter, we remove a percentage of the weights to prevent overfitting. Choosing the optimal dropout rate is also a hyperparameter, but based on numerous studies, including [38], this parameter has a significant impact on controlling overfitting." In this paper, the dropout rate in the search space was set to the range [0.05, 0.50]. As mentioned, the performance Deep learning models is considerably

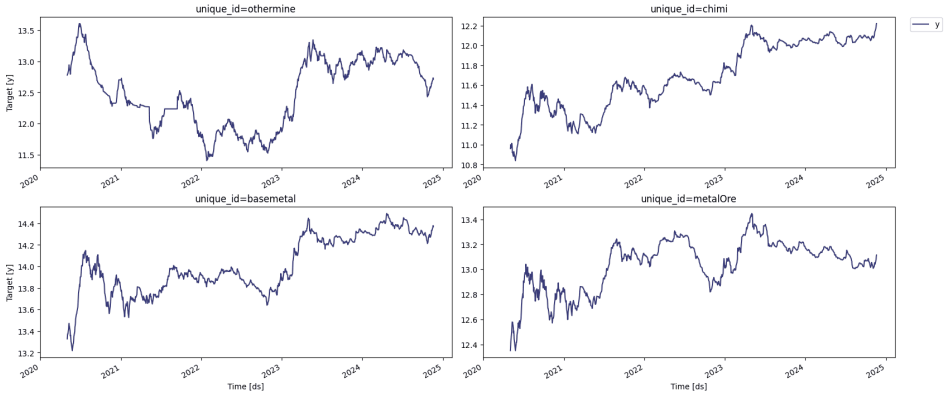


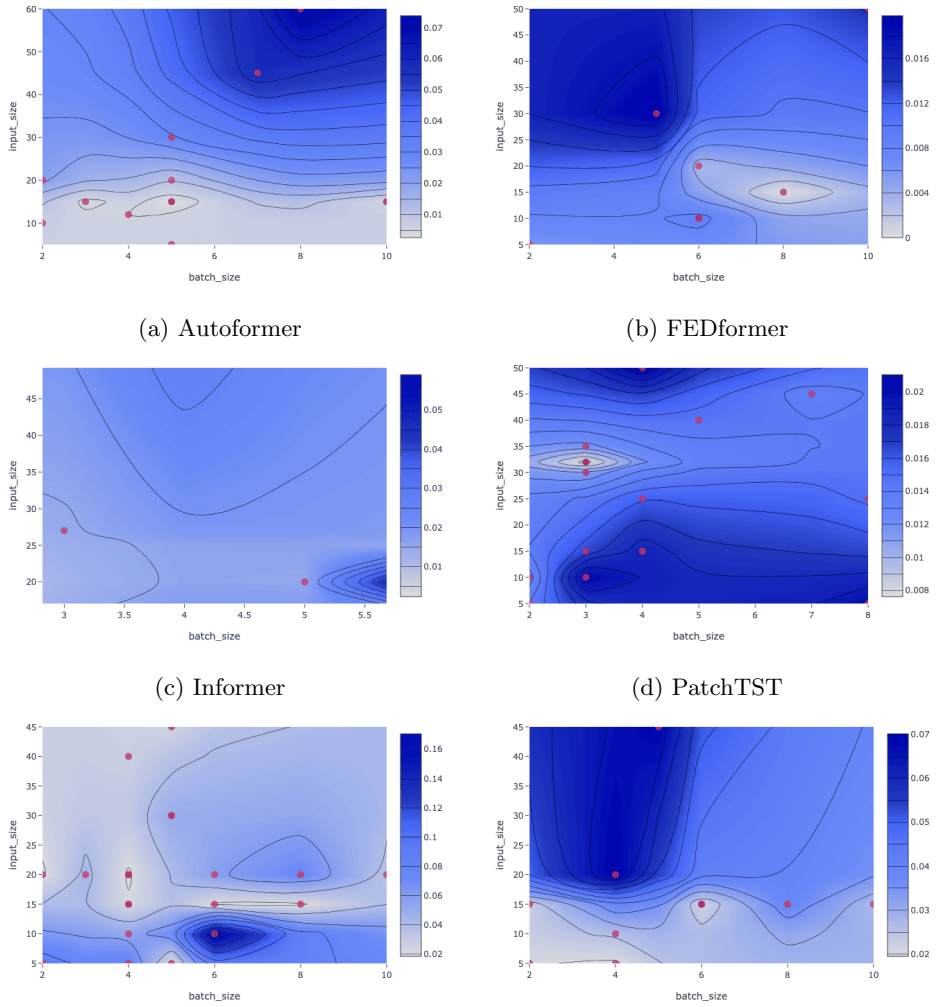
Figure 8: The major industries in the Tehran Stock Exchange are Petrochemicals, Metal and MetalOre and Othermine. Four time series, corresponding to these sectors, were collected daily from April 2020, to November 2024, and represent four commodity-based companies. These industries account for more than 50% of the total corporate market value.

affected by choosing hyper-parameters, which means that Selecting the optimal configuration, a process known as hyper-parameter tuning, is very important to achieve the best performance. about first 80% of data is selected for training the models and selecting related hyper-parameter in each model. in this section we report 6 hyper-parameter in each model only for illustration. The search space for some hyperparameters is summarized in the table 1. Based on the defined search

Table 1: The search space for some hyperparameters

hyperparameters	input	batch	dropout(p)	learning rate (η)	decoder size	head
search space	[2, 50]	[2, 25]	[0.05 , 0.50]	[0.001, 0.05]	[12, 128]	[2,8]

space, 100 random sets of hyperparameters are selected. The results are then plotted using RMSE as the evaluation metric, and additional samples are selected from



(a) Autoformer (b) FEDformer

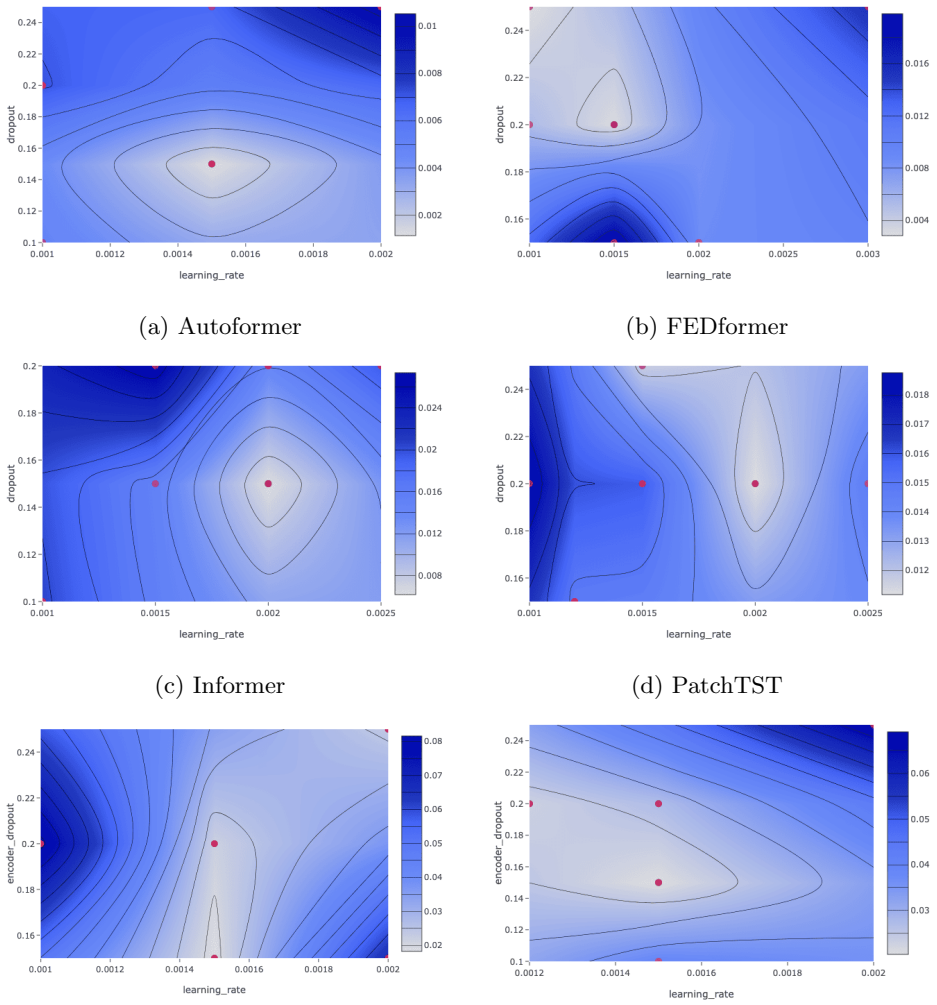
(c) Informer (d) PatchTST

(e) LSTM (f) GRU

Figure 9: hyper-parameters: tune in models for best parameters of batch size and input size

the optimal region for further fine-tuning.

in figure 9, batch size and input size result are plotted for a range of values and optimal values are selected. for example, for Autoformer model, batch size = 5 and input size = 15 is selected. for PatchTST model, batch size = 3 and input size = 32 is selected. in figure 10,dropout and learning rate are plotted for a range of values and optimal values are selected. for example, for Autoformer model, dropout $p = 0.15$ and learning rate $\eta = 0.0015$ is selected. for PatchTST model,



(a) Autoformer

(b) FEDformer

(c) Informer

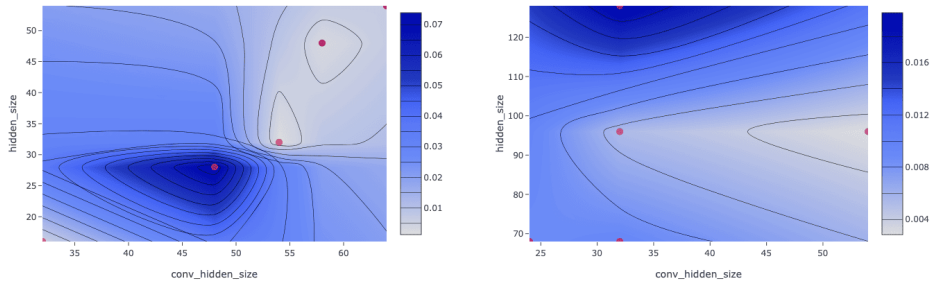
(d) PatchTST

(e) LSTM

(f) GRU

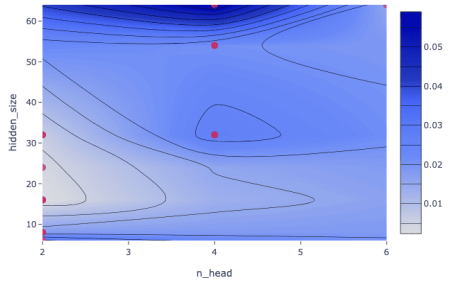
Figure 10: hyper-parameters: tune in models for best parameters of learning rate and drop out

dropout $p = 0.2$ and $\eta = 0.002$ is selected. in figure 11, parameters related to hidden size are plotted. The final hyperparameter tuning is described in the table 2. For reasons related to their computational complexity and architectural structure, models based on the Transformer architecture are used much more extensively than those based on RNNs. Consequently, the training duration for Transformer-based models is significantly longer, and for similar parameters, it is approximately five times longer compared to RNN-based models. Therefore, in order to expedite the

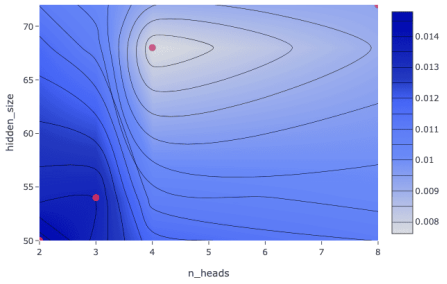


(a) Autoformer

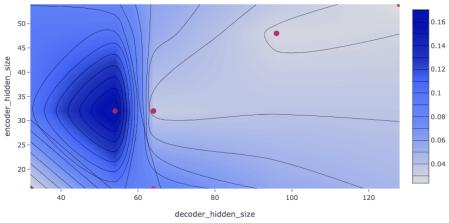
(b) FEDformer



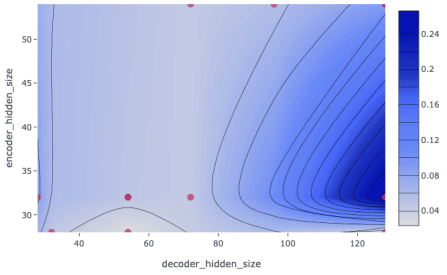
(c) Informer



(d) PatchTST



(e) LSTM



(f) GRU

Figure 11: hyper-parameters: tune in models for best parameter of learning rate and drop out

process of parameter tuning, a smaller number of max steps has been assigned to the Transformer-based models.

Table 2: final tuning hyperparameters for models with forecasting horizontal of 1 day

id	former	FEDformer	Informer	PatchTST	GRU	LSTM
input size	15	15	25	32	15	15
batch size	5	8	5	3	6	6
learning rate	0.0015	0.0015	0.002	0.002	0.0015	0.0015
dropout	0.15	0.2	0.15	0.2	0.2	0.15
decoder hidden size	-	-	-	-	128	48
encoder hidden size	-	-	-	-	54	28
layers	-	-	-	2	2	2
hidden size	54	54	54	50	-	-
max step	300	300	300	300	500	500

The performance of each model across different forecasting horizons was evaluated using the RMSE. Table 3 appears to show the RMSE values for different models on a forecasting task with a horizon of 1 day. The RMSE values are generally low, indicating that the models are doing a good job of forecasting the time series.

The PatchTST model has the lowest RMSE values for 3 out of the 4 time series (basemetal, chimi, and metalOre), indicating that it may be the most efficient model for forecasting these time series. The FEDformer model has the lowest RMSE value for the oilproduct time series, indicating that it may be a good choice for forecasting this particular time series.

The former model has relatively low RMSE values across all time series, indicating that it may be a robust and reliable choice for forecasting. The GRU and LSTM models have higher RMSE values than the other models, indicating that they may not be as effective for forecasting these time series.

Of course, these conclusions are based on a limited dataset and may not generalize to other time series or forecasting tasks. Further analysis and experimentation would be necessary to confirm these findings and explore other models and techniques.

Table 3: Forecasting RMSE for models with horizontal of 1 day: Forecasting windows in this table is 120 day.

id	former	FEDformer	Informer	PatchTST	GRU	LSTM
Basemetal	0.0139	0.0127	0.0127	**0.0123**	0.0177	0.0497
Chimi	0.0088	0.0085	0.0085	**0.0082**	0.0171	0.0148
MetalOre	0.0088	**0.0081**	0.0091	0.0084	0.0093	0.0129
Othermine	0.0311	**0.0272**	0.0279	0.0277	0.0404	0.0336

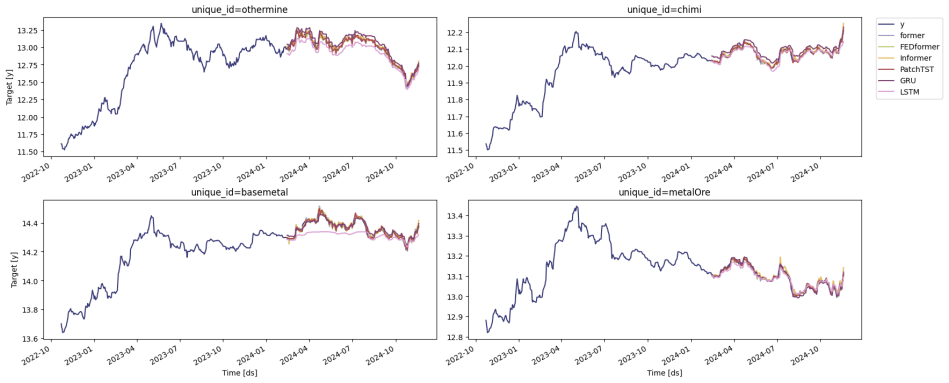


Figure 12: The data for the four indices have been predicted using the six models proposed in this paper for a 1-day . As shown in the performance comparison table of these models, the PatchTST and FEDformer models provide the closest predictions to the target variables.

Here are some observations and insights from the table 4 for weekly forecasting: The RMSE values are generally higher than those in the table 3, which had a horizon of 1 day. This is expected, as forecasting over longer horizons is typically more challenging.

The PatchTST model has the lowest RMSE values for 3 out of the 4 time series (basemetal, chimi, and oilproduct), indicating that it may be the most effective model for forecasting these time series over longer horizons. The FEDformer model has the lowest RMSE value for the metalOre time series, indicating that it may be a good choice for forecasting this particular time series. The former model has relatively high RMSE values across all time series, indicating that it may not be as effective for forecasting over longer horizons.

The GRU and LSTM models have relatively low RMSE values for some time series, but high RMSE values for others, indicating that they may be sensitive to the

specific characteristics of the time series.
Some potential conclusions that can be drawn from this table include: First, the PatchTST model may be the most effective model for forecasting these time series over longer horizons, such as 5 days. Second, the FEDformer model may be a good choice for forecasting certain time series, such as metalOre, over longer horizons. Third, the former model may not be as effective for forecasting over longer horizons, and may require further tuning or optimization to improve its performance. Fourth, the GRU and LSTM models may require careful selection of hyper-parameters and architecture to achieve good performance on these time series.
It's also worth noting that the RMSE values in this table are generally higher than those in the previous table, which suggests that forecasting over longer horizons is more challenging. This may be due to a variety of factors, including:

Increased uncertainty and volatility in the time series over longer horizons,

Decreased accuracy of the models due to the increased complexity of the forecasting task,

Insufficient training data or inadequate model capacity to capture the underlying patterns in the time series.

Table 4: Forecasting RMSE for models with horizontal of 5 days (weekly): Forecasting windows in this table is 120 day.

id	former	FEDformer	Informer	PatchTST	GRU	LSTM
Basemetal	0.0408	0.0307	0.0368	**0.0295**	0.0367	0.0538
Chimi	0.0290	**0.0207**	0.0251	0.0217	0.0233	0.0320
MetalOre	0.0298	**0.0210**	0.0247	0.0213	0.0216	0.0274
Othermine	0.0931	0.0669	0.0739	**0.0572**	0.0583	0.1027

4 conclusion

In the framework of classical portfolio theory, two sets of inputs are required to construct a portfolio of financial assets. The first is the variance-covariance matrix, which indicates the risk of each asset and their correlation and is typically modeled using the MGARCH family of models or the Multivariate Stochastic Volatility (MSV) model. The second is the mean equation, which represents the expected return of each asset over different horizons. In this study, we evaluated the performance of several deep learning models for expected return forecasting, including Former, FEDformer, Informer, PatchTST, GRU, and LSTM.
Four indices have been selected from over 32 significant stock market indices of

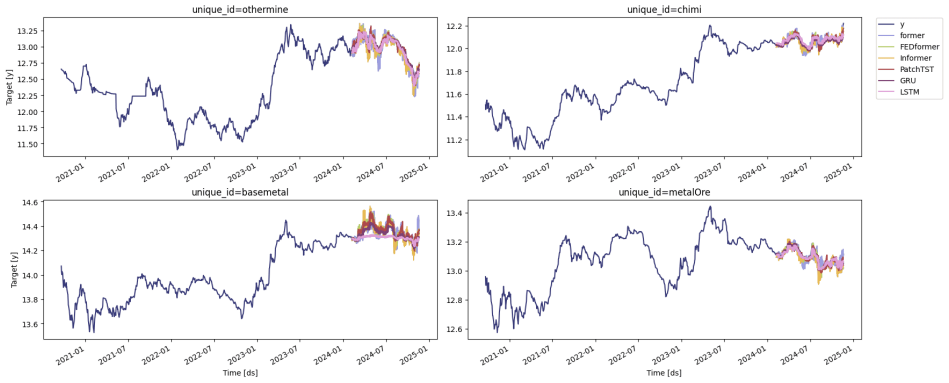


Figure 13: "In this figure, the data for the four indices are predicted using the six models proposed in this paper over a 5-day (weekly) forecasting horizon. As shown in the performance comparison table, the PatchTST and FEDformer models most closely align with the target variables. However, the figure also reveals that forecasting costs remain high, as evidenced by the relatively high RMSE values across all models."

the Tehran Stock Exchange. Four time series, corresponding to these sectors, were collected daily from April 2020, to November 2024, and represent four commodity-based companies. These industries account for more than 50% of the total corporate market value. This data possesses several characteristics. First, it pertains to commodity-based companies. Second, exchange rate shocks impact these companies, increasing the complexity of predicting these variables due to a wide array of factors . Third, unforeseen shocks, such as changes in regulations, further complicate the price behavior of these companies and their related indices examined in this paper. Consequently, the indices used for prediction in this paper represent some of the most complex data for forecasting.

Our results show that the PatchTST model outperforms the other models in terms of RMSE for forecasting horizons of both 1 day and 5 days. The FEDformer model also shows promising results, particularly for forecasting the metalOre time series. Our findings are consistent with the literature, which suggests that deep learning models can be efficient for time series forecasting tasks. For example, studies have shown that models such as LSTM and GRU can achieve good performance on time series forecasting tasks, particularly when combined with other techniques such as feature engineering and ensemble methods [24]. However, our results also highlight the importance of careful model selection and hyper-parameter tuning, as the performance of different models can vary significantly depending on the specific task and dataset.

The PatchTST model, in particular, has shown promising results in our study, and its performance is consistent with the literature. For example, in [18] found that

the PatchTST model outperformed other deep learning models on a time series forecasting task, and attributed its success to its ability to capture long-term dependencies in the data.

Overall, our study contributes to the growing body of literature on deep learning for time series forecasting, and highlights the importance of careful model selection and hyper-parameter tuning for achieving good performance. Our results also suggest that the PatchTST model may be a promising choice for time series forecasting tasks, particularly when combined with other techniques such as feature engineering and ensemble methods.

There are several directions for future research based on our study. One potential direction is to explore the use of other deep learning models, such as transformers and graph neural networks, for time series forecasting tasks. Another direction is to investigate the use of ensemble methods, such as stacking and bagging, to combine the predictions of multiple models and improve overall performance. Finally, future research could also focus on developing more robust and interpretable models, such as those that incorporate domain knowledge and uncertainty quantification.

Bibliography

[1] K. Benidis, S. S. Rangapuram, V. Flunkert, Y. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, F.-X. Aubet, L. Callot, and T. Januschowski, *Deep learning for time series forecasting: Tutorial and literature survey*, 2022.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, in Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1–12.

[3] J. Kenton, R. Rajpurkar, J. Hinton, and J. L. Ba, *BERT: Pre-training of deep bidirectional transformers for language understanding*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019, pp. 1–12.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 1–12.

[5] X. Dong, J. Li, D. Yu, F. Seide, and M. L. Seltzer, *Speech recognition with deep learning: A review*, IEEE Transactions on Audio, Speech, and Language Processing, 26 (2018), pp. 1–13.

[6] B. Lim and S. Zohren, *Deep learning for time series forecasting: A survey*, Journal of Forecasting, 40 (2021), pp. 1–23.

[7] Y. Tay, D. Bahri, D. Metzler, D. Juan, Z. Zhao, and C. Zheng, *Efficient transformers for natural language processing*, in Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2022, pp. 1–12.

[8] J. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso, *Deep learning for time series forecasting: A survey*, Journal of Forecasting, 40 (2021), pp. 1–23.

[9] S. Tuli, S. K. Singh, S. K. Singh, and R. Buyya, *Anomaly detection in time series data using deep learning*, Journal of Intelligent Information Systems, 58 (2022), pp. 1–15.

[10] X. Kong, Z. Chen, W. Liu, K. Ning, L. Zhang, S. M. Marier, Y. Liu, Y. Chen, and F. Xia, *Deep learning for time series forecasting: A survey*, International Journal of Machine Learning and Cybernetics, (2025).

[11] Y. W. Xiong, K. Tang, M. Ma, J. Zhang, J. Xu, and T. Li, *Modeling temporal dependencies within the target for long-term time series forecasting*, arXiv preprint arXiv:2406.04777v2 [cs.LG], 2024.

[12] Y. Wu, L. Zhang, and Y. Zhang, *Deep learning for time series forecasting: A review*, Journal of Forecasting, 40 (2021), pp. 1–23.

[13] X. Xu, J. Li, L. Zhang, and Y. Zhang, *Anomaly detection in time series data using deep learning*, Journal of Intelligent Information Systems, 58 (2022), pp. 1–15.

[14] A. Casolaro, V. Capone, G. Iannuzzo, and F. Camastra, *Deep learning for time series forecasting: Advances and open problems*, Information, 14 (2023), pp. 598. DOI:10.3390/info14110598.

[15] G. Zerveas, L. Zhang, and Y. Zhang, *Deep learning for time series classification: A review*, Journal of Forecasting, 40 (2021), pp. 1–23.

[16] P. Lara-Benítez, M. Carranza-García, and J. C. Riquelme, *An experimental review on deep learning architectures for time series forecasting*, International Journal of Neural Systems, 31 (2021).

[17] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, *Informer: Beyond efficient transformer for long sequence time-series forecasting*, Journal of Machine Learning Research, (2021), pp. 1–23.

[18] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, *A time series is worth 64 words: Long-term forecasting with transformers*, in Proceedings of the International Conference on Learning Representations (ICLR), 2023, pp. 1–12.

[19] Y. Wang, J. Li, Y. Zhang, H. Xiong, and W. Zhang, *FEDformer: Frequency enhanced decomposed transformer for long-term time series forecasting*, in Proceedings of the International Conference on Learning Representations (ICLR), 2022, pp. 1–12.

[20] Y. Wu, S. Li, S. Zhang, J. Li, H. Xiong, and W. Zhang, *Autoformer: Decomposition transformers for long-term time series forecasting*, in Proceedings of the International Conference on Learning Representations (ICLR), 2022, pp. 1–12.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.

[22] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using RNN encoder-decoder for statistical machine translation*, in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

[23] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[24] S. Bai, J. Z. Kolter, and V. Koltun, *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*, arXiv preprint arXiv:1803.01271, 2018.

[25] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, *Patch time series transformer in Hugging Face - Getting started*, Hugging Face Blog, 2023.

[26] Towards Data Science, *PatchTST: A breakthrough in time series forecasting*, Towards Data Science, 2023.

[27] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and practice*, 3rd ed., OTexts: Melbourne, Australia, 2021. `OTexts.com/fpp3`.

[28] C. Bergmeir and J. M. Benítez, *On the use of cross-validation for time series predictor evaluation*, Information Sciences, 191 (2012), pp. 192–213. DOI:10.1016/j.ins.2011.12.028.

[29] J. Bergstra and Y. Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research, 13 (2012), pp. 281–305.

[30] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, *Efficient and robust automated machine learning*, in NeurIPS, 2019.

[31] Optuna Development Team, *Optuna hyperparameter optimization guide*, Documentation, `https://optuna.org/`, 2023.

[32] Ray Team, *Hyperparameter tuning with Ray Tune*, Documentation, `https://docs.ray.io/en/latest/tune/`, 2023.

[33] D. Maclaurin, D. Duvenaud, and R. P. Adams, *Gradient-based hyperparameter optimization through reversible learning*, in Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015. `https://proceedings.mlr.press/v37/maclaurin15.html`.

[34] T. Bollerslev, R. F. Engle, and J. M. Wooldridge, *A capital asset pricing model with time-varying covariances*, Journal of Political Economy, 96 (1988). DOI:10.1086/261527.

[35] M. Asai, C.-L. Chang, and M. McAleer, *Realized volatility and MGARCH models: A review*, Econometrics, 9 (2021).

[36] R. Taleblou and P. Mohajeri, *Modeling the daily volatility of oil, gold, dollar, bitcoin and Iranian stock markets: An empirical application of a nonlinear space state model*, Iranian Economic Review, (2023).

[37] G. Kastner and S. Frühwirth-Schnatter, *Ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC estimation of stochastic volatility models*, Computational Statistics Data Analysis, 76 (2014), pp. 408–423.

[38] F. Chollet, *Deep learning with Python*, 2nd ed., Manning Publications, Shelter Island, NY, 2021.

[39] M. Rezaei, N. Neshat, A. Jafari Nodoushan, and A. Ahmadzadeh, *The artificial neural networks for investigation of correlation between economic variables and stock market indices*, Journal of Mathematics and Modeling in Finance, 3 (2023), pp. 19–35.

[40] M. Abdollahzadeh, A. Baagherzadeh Hushmandi, and P. Nabati, *Improving the accuracy of financial time series prediction using nonlinear exponential autoregressive models*, Journal of Mathematics and Modeling in Finance, 4 (2024), pp. 159–173.

[41] M. Goldani, *Comparative analysis on forecasting methods and how to choose a suitable one: Case study in financial time series*, Journal of Mathematics and Modeling in Finance, 3 (2023), pp. 37–61.

How to Cite: Reza Taleblou[1], *Comparing the performance of different deep learning architectures for time series forecasting*, Journal of Mathematics and Modeling in Finance (JMMF), Vol. 5, No. 1, Pages:63–87, (2025).