[1]Anuku Arjuna Rao

[2]P. Mallikarjuna Rao

[3]D. Vijaya Kumar

Hybrid TCN-Based Bi-GRU-LSTM for Enhanced Long-Term Electric Load Forecasting

Journal of Electrical Systems

Abstract: - Electric load forecasting plays a vital role in the effective management and planning of energy within power systems. his research provides an in-depth assessment of several deep learning frameworks for electric load forecasting, concentrating on four specific models: LSTM, GRU-LSTM, Bi-GRU-LSTM, and an innovative TCN-Bi-GRU-LSTM architecture. The models underwent training and testing using a real-world dataset, with input features derived from timestamps to capture temporal trends. The forecasting effectiveness was evaluated over daily, weekly, and monthly timeframes utilizing metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Although the conventional LSTM and GRU-LSTM models showed constraints in their accuracy for long-term forecasts, the TCN-Bi-GRU-LSTM model successfully leveraged convolutional layers to understand intricate temporal relationships, leading to enhanced performance, especially in monthly predictions. The results suggest that the combination of convolutional layers with recurrent architectures significantly boosts the model's capability to identify long-range patterns, thus enhancing load forecasting precision. This study advances the field of energy forecasting by highlighting the advantages of hybrid architectures in tackling the difficulties associated with long-term load prediction

Keywords: Load Forecasting, LSTM, GRU, Bidirectional RNN, TCN, Deep Learning, Hybrid Models, Time Series Forecasting.

I. INTRODUCTION

Short-term load forecasting (STLF) is a crucial component of modern power system management and operation, enabling utilities to predict electricity demand over short time horizons, typically ranging from a few minutes to several days ahead. Accurate load forecasting facilitates effective resource allocation, operational planning and enhances the reliability and efficiency of power systems. Traditional methods for STLF include statistical approaches such as autoregressive integrated moving average (ARIMA) models and exponential smoothing. However, these techniques often struggle to capture the complex, nonlinear patterns present in load data, especially with the increasing integration of renewable energy sources and changing consumer behavior [1,2].

In recent times, deep learning methods have surged in popularity because of their proficiency in capturing intricate relationships within data. Among these methodologies, Long Short-Term Memory (LSTM) networks have emerged as a proficient approach for Short-Term Load Forecasting (STLF), attributable to their capacity to comprehend temporal dependencies and competently handle sequential datasets. Long Short-Term Memory networks (LSTMs) exemplify a distinct subclass of recurrent neural networks (RNNs) meticulously designed to mitigate issues such as vanishing and exploding gradients, phenomena that commonly affect conventional RNN architectures [3,4].

Many researchers have shown the efficacy of LSTM in predicting short-term loads, frequently surpassing traditional forecasting models in accuracy [5]. To improve the efficacy of LSTM, researchers have investigated hybrid models, including the Gated Recurrent Unit LSTM (GRU-LSTM). This model merges the benefits of GRUs, which possess a more straightforward architecture compared to LSTMs, with the robust learning abilities of LSTM. GRU-LSTM models have demonstrated enhanced forecasting accuracy by adeptly capturing short-term

[3]*Corresponding author:

[1]Research Scholar, Department of Electrical Engineering, AU college of Engineering, Andhra University, Visakhapatnam, Andhra Pradesh, India

[2]Professor, Department of Electrical Engineering, AU college of Engineering, Andhra University, Visakhapatnam, Andhra Pradesh, India

[3]*Professor, Department of Electrical and Electronics Engineering, Aditya Institute of technology and Management, Tekkali, Srikakulam District, Andhra Pradesh, India

dependencies and trends in load data [6,7]. Further development of these methods has led to the creation of Bidirectional GRU-LSTM (Bi-GRU-LSTM) models, which leverage the bidirectional processing features of GRUs. This configuration enables the model to simultaneously assess both past and future input data, thus improving its comprehension of the fundamental temporal dynamics of the load[8]. The integration of bidirectional networks has shown to be advantageous for scenarios where context from both ends of the sequence is essential for precise predictions [9].Moreover, the combination of Temporal Convolutional Networks (TCNs) with Bi-GRU-LSTM frameworks has become increasingly popular. TCNs employ convolutional layers featuring dilated convolutions to more effectively capture long-range dependencies compared to conventional RNN architectures[10,11]. The TCN-enhanced Bi-GRU-LSTM model harnesses the advantages of both frameworks, delivering superior feature extraction, greater training stability, and enhanced generalization abilities, particularly in the realm of short-term load forecasting[12]. Eventually, the development of STLF techniques through the integration of deep learning methods such as LSTM, GRU-LSTM, Bi-GRU-LSTM, and TCN-based Bi-GRU-LSTM signifies remarkable progress in both forecasting precision and efficiency. These models not only overcome the shortcomings of conventional techniques but also adjust to the intricacies of contemporary power systems, rendering them vital instruments for effective energy management and strategic planning.

Traditional techniques like Auto-Regressive Integrated Moving Average (ARIMA) and Exponential Smoothing models have been extensively utilized for load forecasting. Nevertheless, these methods are based on linear assumptions and struggle to effectively capture non-stationary trends in load data. As a result, machine learning models such as Support Vector Regression (SVR) and Random Forest have been adopted, yet they still have limitations when it comes to addressing sequential dependencies over extended timeframes. Recurrent Neural Networks (RNNs) have demonstrated their effectiveness in forecasting time series due to their capacity to capture sequential dependencies. Nevertheless, traditional RNNs encounter challenges with vanishing and exploding gradients during training, particularly with lengthy sequences.

To address these issues, Long Short-Term Memory (LSTM) networks were introduced [3]. An LSTM model for short-term load forecasting showed improvements over traditional methods and better peak load predictions. The importance of feature selection and pre-processing in improving forecast performance was emphasized [14]. LSTM networks are a type of recurrent neural network designed to preserve information over long periods. Their architecture enables them to effectively capture temporal dependencies in sequential data, making them suitable for short-term load forecasting (STLF). LSTM techniques were utilized for short-term load predictions in a smart grid environment[15]. This study demonstrated the model's capacity to learn from historical load data, leading to improved forecasting accuracy. A comparison revealed LSTM's superiority over conventional time series methods, yielding significantly enhanced results.

The Gated Recurrent Unit (GRU) is a simplified version of Long Short-Term Memory (LSTM) that reduces complexity while maintaining similar performance. The integration of GRU and LSTM exploits their respective advantages for load forecasting. The GRU is a streamlined adaptation of LSTM that simplifies its architecture while achieving comparable performance. Hybrid models that blend GRU and LSTM leverage the strengths of both to enhance forecasting accuracy. Hybrid GRU-LSTM model for Short-Term Load Forecasting (STLF) [16], demonstrated enhanced robustness and accuracy through the integration of GRU and LSTM. Their results indicated superior performance compared to independent LSTM and GRU models. The hybrid GRU-LSTM model for STLF[16] showcased its capability to effectively capture complex temporal patterns in load data, thereby outperforming standalone GRU and LSTM models. The study further highlighted the importance of feature engineering and hyper parameter optimization in enhancing model performance.

Bidirectional RNNs effectively gather information from both historical and forthcoming states. This capability is utilized by the model to improve short-term load forecasting performance and to deepen the comprehension of temporal dependencies by analysing sequences in both directions. The author [17] introduced a Bi-GRU-LSTM model that outperformed unidirectional models in forecasting accuracy. Their experiments on various datasets confirmed the bidirectional method's effectiveness in recognizing complex load data patterns. Bi-GRU-LSTM model for short-term load forecasting [18], showcased bidirectional processing benefits. Testing on real-world datasets proved the model's robustness and accuracy in identifying historical load patterns and trends.

The growing adoption of renewable energy sources and the increasing intricacy of electrical grids render load forecasting an essential aspect of contemporary power systems. As utilities and grid operators work to sustain a balance between demand and supply, accurate load forecasts facilitate improved planning for resource allocation, scheduling of power generation units, and reduction of energy losses. The incorporation of distributed energy resources (DERs), electric vehicles (EVs), and energy storage systems necessitates reliable predictions to prevent grid instability and enhance energy market functions. Nonetheless, conventional forecasting methods, including ARIMA, Holt-Winters, and exponential smoothing, fall short of effectively modeling nonlinearities and temporal dependencies present in load data.

Temporal Convolutional Networks (TCN) efficiently process sequential data using convolutional layers. Integrating TCN with Bi-GRU-LSTM improves feature extraction from time series data. TCNs provide a strong framework for load forecasting by enhancing feature extraction. The TCN and Bi-GRU-LSTM combination excels in managing long-range dependencies [19]. Researchers [19] introduced TCNs, showing they surpass traditional RNNs in time series tasks. They noted TCNs' strengths in handling long-range dependencies and training efficiency. Temporal Convolutional Networks (TCN) can effectively complement recurrent models, offering faster training and enhanced temporal feature extraction through dilated convolutions. A model that combines a Temporal Convolutional Network (TCN) with a Bi-directional Gated Recurrent Unit (Bi-GRU) and Long Short-Term Memory (LSTM) is noted for its ability to competently tackle these challenges. TCNs are well-regarded for their capability to capture long-range dependencies using a hierarchical structure, while Bi-GRU and LSTM networks excel at processing sequential data, making them particularly suited for time series forecasting. Therefore, integrating these advanced architectures can enhance the accuracy of short-term load forecasts, ultimately contributing to more efficient energy management and sustainable grid operations. Recent research has investigated hybrid models that integrate the advantages of various architectures. For instance, in GRU-LSTM hybrids, researchers [20] have employed to improve forecasting precision by utilizing the efficiency of GRUs alongside the memory functions of LSTMs. Nevertheless, these models continue to encounter scalability challenges when handling larger datasets.

Accurate forecasting of electrical load is crucial for achieving optimal performance, stability, and cost-effectiveness within power systems. Nevertheless, various challenges emerge in practical situations such as (i)Temporal Dependencies: Load data displays both short-term and long-term relationships that must be effectively captured (ii) Seasonality and Trends: Daily, weekly, and seasonal fluctuations create non-stationary patterns, complicating the forecasting process. (iii) Uncertainty in Data: Abrupt shifts in demand driven by human behavior, weather conditions, or unforeseen events introduce volatility into load profiles.

The problem addressed in this paper involves the development of robust architecture capable of handling temporal dependencies, seasonality, and volatility to improve forecasting accuracy. Specifically, this work examines four different architectures: LSTM which captures long-term relationships, GRU-LSTM which is utilizing both the efficiency of GRUs and the memory capabilities of LSTMs, Bidirectional GRU-LSTM that analyses sequences in both forward and reverse directions, and TCN-Bi-GRU-LSTM merges convolutional networks for feature extraction with recurrent models for sequential learning. This paper proposes a Temporal Convolutional Network (TCN)-based Bidirectional Gated Recurrent Unit (Bi-GRU) and Long Short-Term Memory (LSTM) model specifically designed for short-term load forecasting, which has demonstrated superior performance compared to conventional LSTM and GRU models, thereby underscoring the significance of hybrid methodologies in effectively capturing the intricacies inherent in time series datasets. Investigations have been carried out on real-time load data for the year 2023 from the available reports on the website of the Andhra Pradesh State Load Dispatch Centre [21]. Subsequent research attempts will investigate the incorporation of supplementary variables such as meteorological data and seasonal variations to further refine the precision of forecasting outcomes.

II. METHODOLOGY

The proposed approach consists of several essential stages: gathering and preparing data, developing the model, training, and assessing, and establishing a forecasting strategy. Every stage is vital for achieving accurate and dependable load forecasting. In this work, four different models are evaluated based on their performance metrics. The architecture of the learning framework of the work is shown in Fig.1

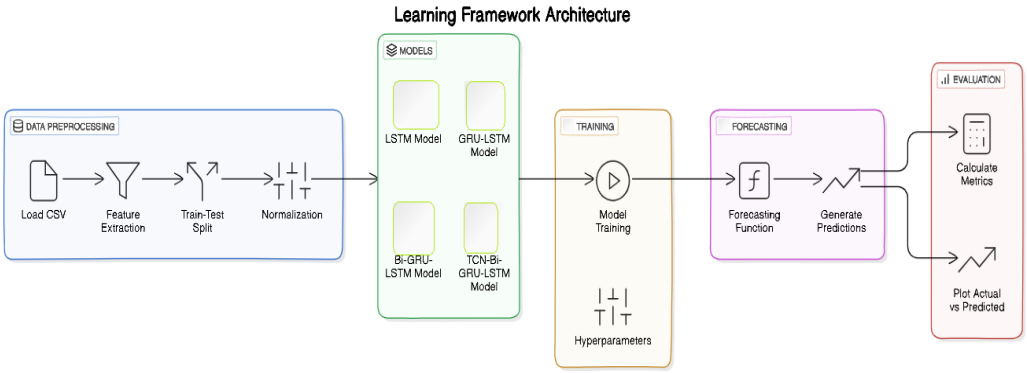

Fig.1 Learning Framework Architecture of the proposed work

2.1 Dataset Description

The dataset employed in the present investigation comprises hourly load data, with timestamps extending over several months. Variables such as hour, day, month, and day of the week were extracted from the Datetime field to clarify the temporal dependencies that affect electricity consumption.

2.2 Data Preprocessing

- Normalization: The process of Min-Max Scaling was employed to guarantee numerical stability throughout the training phase.
- Train-Test Split: A proportion of 80% of the dataset was allocated for the training process, while the remaining 20% was designated for the testing phase.
- Reshaping: The input dataset was reshaped to conform to the specifications required by LSTM/GRU architectures, specifically structured as samples, time steps, and features.

2.3 Model Architectures

This section offers a comprehensive theoretical overview of the deep learning architectures employed in load forecasting. The mathematical principles underlying LSTM, GRU, Bidirectional GRUs, TCNs, and the hybrid TCN-Bi-GRU-LSTM model are examined in detail. Each model uniquely captures temporal dependencies, with their specific layers meticulously designed to manage sequential data efficiently. Fig.2 Shows the Flow Chart for the proposed methodology

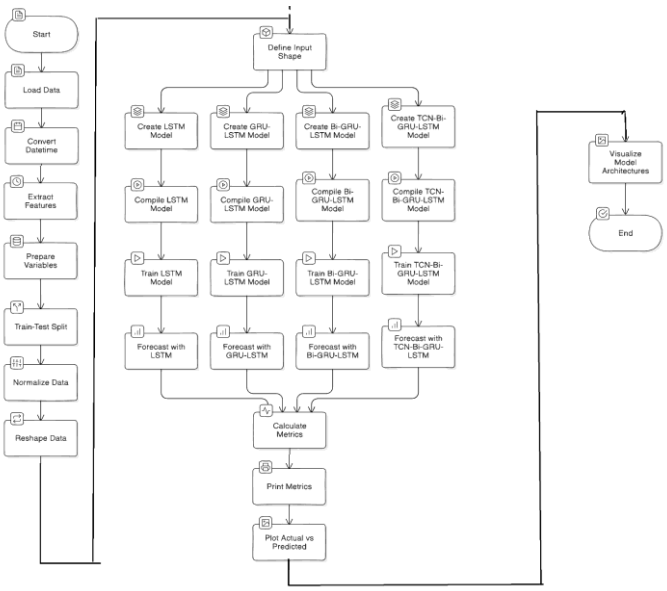

Fig.2 Flow Chart for the proposed methodology

2.3.1 Long Short-Term Memory (LSTM)

LSTM is a distinctive form of Recurrent Neural Network (RNN) specifically designed to capture long-term dependencies by addressing the vanishing gradient problem. The main innovation in LSTM is the integration of memory cells and gates: the input gate, forget gate, and output gate. These gates regulate the flow of information, deciding which portions of the historical data should be kept or eliminated.

At time step 't', when provided with the input 'xt', the LSTM performs the following computations:

a. Forget gate: This gate regulates how much of the preceding cell state 'Ct−1' needs to be retained or eliminated for the next time step. This is advantageous when certain past patterns lose their significance. The function f(t) in eq (1) denotes the output produced by the forget gate. When f(t)=0, all previous information is fully eliminated; on the other hand, if f(t)=1, it is wholly retained.

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

- σ denotes the sigmoid activation function:
- $\sigma(x)=\frac{1}{1+e^{-x}}$ It generates outputs between 0 and 1, acting as a mechanism to either retain or eliminate information.
- W_f and b_f represent the weights and bias associated with the forget gate.
- h_{t-1} is the hidden state from the preceding time step, while x_t represents the current input.

b. Input gate: The input gate determines which elements of the new input are to be incorporated into the cell state C_t. It employs two key components: the gate activation in eq(2) and the candidate cell state in equation (3). Here, ReLU activation function returns the input value directly if it is positive and zero otherwise. ReLU enhances the ability of LSTMs to manage long-range dependencies more efficiently by alleviating vanishing gradient problems.

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

$$\tilde{C}_t=\mathrm{ReLU}(W_C\cdot[h_{t-1},x_t]+b_c) \quad (3)$$

C. Update Cell State (Merge Historical and Present Data): the cell state C_t serves as the memory for the LSTM, gathering information over time as in eq(4). The revised cell state is a blend of the prior state (adjusted by the forget gate) and the new candidate state (modulated by the input gate).This procedure guarantees that the LSTM preserves valuable historical information while incorporating pertinent new data.

$$C_t=f_t\odot C_{t-1}+i_t\odot\tilde{C}_t \quad (4)$$

- ⊙ signifies element-wise multiplication.
- $f_t\odot C_{t-1}$, Retains a segment of the previous state.
- $i_t\odot\tilde{C}_t$, Incorporates fresh information from the input.

D. Output Gate: The output gate regulates the amount of the modified cell state that will be represented as the hidden state 'h_t'. The hidden state serves both for making predictions and as an input for the subsequent time step. Output gate activation is as described by eq(5) and hidden state is computed as in eq(6). The hidden state 'h_t' transmits information to the next step and is applicable for activities such as forecasting at each interval.

$$o_t=\sigma(W_o\cdot[h_{t-1},x_t]+b_o) \quad (5)$$

$$h_t=o_t\odot\mathrm{ReLU}(C_t) \quad (6)$$

At every time step '*t*',the forget gate determines the extent of past information to retain, the input gate identifies new pertinent information, the cell state is refreshed by merging the prior and present information, the output gate calculates the hidden state for the subsequent step. LSTMs possess the ability to retain information across multiple time steps thanks to the memory cell. The gates manage the flow of information, ensuring that gradients do not diminish excessively during back propagation. By fine-tuning the gates, the LSTM can concentrate on significant segments of the sequence, even when the dependencies are long-range. This capability renders LSTMs

an effective tool for applications such as load forecasting, where both short-term patterns (hourly trends) and long-term dependencies (seasonal fluctuations) must be effectively captured.

2.3.2 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is an optimized variant of the LSTM that enhances computational efficiency while preserving robust performance in sequence-learning applications. GRUs eliminate certain complexities associated with LSTM, like distinct memory cells, and utilize two gates instead: the update gate and the reset gate. This streamlining decreases the parameter count, resulting in faster training times for GRUs. Unlike Long Short-Term Memory networks (LSTMs), which have separate mechanisms for managing input, forgetting, and output, Gated Recurrent Units (GRUs) integrate multiple functionalities into two main gates: the update gate, which controls how much past information is retained and how much new information is taken in, and the reset gate, which specifies how much of the previous hidden state should be eliminated. The generated learning framework is as in Fig.4

Let,

- x_t is the input at time step t
- h_{t-1} is the hidden state from the preceding time step
- W_z, W_r, W_h are Weight matrices corresponding to the update, reset, and candidate hidden state, respectively
- b_z,b_r,b_h are bias terms linked to each gate

a. Update gate: The update gate, 'z_t' determines how much of the previous hidden state ht−1 will be retained and how much new information will be added as in eq(7).

$$z_t=\sigma(W_z\cdot[h_{t-1},x_t]+b_z) \quad (7)$$

- 'σ': The sigmoid activation function produces outputs that range from 0 to 1.
- 'z_t': The update gate value at time step t. When it approaches 1, the previous state ht−1 is preserved; when it nears 0, additional new information is integrated.

b. Reset Gate: The reset gate, referred to as 'r_t', indicates the extent to which the previous hidden state, 'h_{t-1}', should be disregarded in the calculation of the new candidate hidden state as illustrated in eq(8). As 'r_t' approaches 0, the model essentially discards the previous hidden state. This functionality allows the GRU to concentrate on pertinent recent inputs instead of long-term dependencies when it is considered necessary.

$$r_t=\sigma(W_r\cdot[h_{t-1},x_t]+b_r) \quad (8)$$

C. Candidate Hidden State: The candidate hidden state,$\tilde{h}_t$signifies the prospective new hidden state shaped by the effect of the reset gate as in eq(9). By utilizing the reset gate 'r_t' on the previous hidden state 'h_{t-1}', the GRU can selectively disregard aspects of the past that do not pertain to the current input.

$$\tilde{h}_t=\mathrm{ReLU}(W_h\cdot[r_t\odot h_{t-1},x_t]+b_h) \quad (9)$$

ReLU: This activation function returns the input value directly if it is positive and zero otherwise.

⊙: This denotes multiplication performed element by element.

D. Hidden State Update : The final hidden state,'h_t' represents a weighted combination of the prior hidden state,'h_{t-1}' and the candidate hidden state,$\tilde{h}_t$ as shown in eq(10).The weights are established by 'z_t'. When 'z_t' is near 1, $\tilde{h}_t$ significantly takes over 'h_{t-1}'. Conversely, if 'z_t'is near 0, 'h_{t-1}' is predominantly preserved.

$$h_t=(1-z_t)\odot h_{t-1}+z_t\odot\tilde{h}_t \quad (10)$$

GRUs function instinctively, evaluating the extent to which the prior state should be maintained. They ascertain whether the previous hidden state should be eliminated as they move towards the new state. A potential new state is influenced by the interaction between the reset gate and the current input. It embodies a fusion of the past state and the new candidate state, governed by the update gate.

GRUs are particularly effective for handling time-series data, where the model needs to learn the connections between historical and future values. For instance, in the context of load forecasting, GRUs can identify daily or weekly trends by effectively balancing the retention of previous information with the capacity to disregard outdated, irrelevant data. By eliminating the distinct memory cell found in LSTMs, GRUs offer a more straightforward architecture with a reduced number of parameters. With a lesser number of gates and computations, GRUs provide quicker training times and enhanced computational efficiency. Despite their streamlined nature, GRUs frequently achieve performance on par with or superior to that of LSTMs, particularly when dealing with shorter sequences.

2.3.3 Bidirectional GRU (Bi-GRU)

The Bi-GRU represents an enhancement of the GRU framework that analyses the input sequence in both forward and backward orientations. This capability enables the model to grasp both historical and forthcoming dependencies at the same time, thereby enhancing its forecasting ability, particularly for tasks where upcoming events influence the present prediction, such as peak load forecasting. A typical GRU analyses sequences in a single direction i.e., from past to future, indicating that the hidden state at time '*t*' relies on the preceding hidden state 'h_{t-1}'. Nevertheless, in numerous tasks, it is essential to consider both past and future contexts to achieve precise predictions. A Bi-GRU addresses this limitation by utilizing two GRUs. The forward GRU processes the input sequence from left to right i.e., from past to future. The backward GRU processes the input sequence from right to left i.e., from future to past. The outputs generated from both the forward and backward passes are combined to create the final hidden state, effectively capturing dependencies in both directions.

a. Input Sequence and Hidden States:

Let,

- x_t is defined as the input at the temporal step t.
- $\overrightarrow{h_t}$ is designated as the forward hidden state at the temporal step t.
- $\overleftarrow{h_t}$ is as the backward hidden state at the temporal step t.

Calculation of Forward Hidden State: The forward GRU analyses the sequence from start to finish i.e., past to future. At every time step '*t*', the forward hidden state is computed as shown in eq(11). $\overrightarrow{h_{t-1}}$, represents the hidden state from the preceding time step. The forward GRU effectively captures dependencies from prior inputs.

$$\overrightarrow{h_t}=\mathrm{GRU}\left(x_t,\overrightarrow{h_{t-1}}\right) \tag{11}$$

Calculation of Backward Hidden State: The backward GRU analyzes the sequence in reverse order, moving from the end to the beginning (from future to past). At every time step t, the hidden state for the backward pass is determined as shown in eq(12). Here, $\overleftarrow{h_{t+1}}$, represents the hidden state from the subsequent time step. The backward GRU effectively captures dependencies from inputs that occur later in the sequence.

$$h_t=\mathrm{GRU}\left(x_t,\overleftarrow{h_{t+1}}\right) \tag{12}$$

Final Hidden State: After processing the input sequence in both directions, the hidden states derived from the forward and backward passes are merged to create the final hidden state at each time step, as shown in eq(13). Here, 'h_t', encompasses information from both the forward and backward contexts at time '*t*'. This allows the Bi-GRU to effectively capture causal, where the past affects the future and non-causal, where the future influences past dependencies present in the data.

$$h_t=\left[\overrightarrow{h_t},\overleftarrow{h_t}\right] \tag{13}$$

Bi-GRUs are capable of capturing relationships that rely on both preceding and subsequent time steps. For instance, peak load forecasting gains from understanding both morning and afternoon usage trends. It involves predicting electricity demand based on historical and forthcoming data patterns. The Bidirectional GRU (Bi-GRU) analyzes input sequences in both forward and reverse directions, thereby capturing dependencies from both past and future contexts. By merging the forward and backward hidden states, Bi-GRUs improve the model's capacity to manage intricate sequential patterns. With a reduced number of parameters compared to Bidirectional LSTMs, they achieve an optimal balance between computational efficiency and predictive accuracy.

2.3.4 Temporal Convolutional Network (TCN)

A Temporal Convolutional Network (TCN) represents a specialized computational architecture formulated to manage sequential data, such as time series. In contrast to conventional recurrent neural networks (RNNs), TCNs leverage convolutional layers to effectively model temporal dependencies, thereby facilitating accelerated and more robust training processes. The two principal components of TCNs are: 1. Causal Convolutions – which preserve the temporal sequence, ensuring that predictions at the present time step are contingent solely upon prior inputs. 2. Dilated Convolutions – which proficiently capture long-range dependencies without augmenting the depth of the network. Causal Convolutions guarantee that the output at a given time step t is solely influenced by prior inputs, excluding future data. This retention of the temporal structure of the sequence is imperative for applications such as forecasting. Dilated Convolutions empower the model to encompass an extended time window by omitting inputs at regular intervals. This capability enables the network to effectively capture long-term dependencies without necessitating excessively deep architectures.

Let 'x_t' be the Input at time t, 'y_t' be the Output at time t, 'w_i' be the Filter weight for the i^{th} element, 'd' be the dilation factor that specifies the degree of separation between the elements of the input, 'k' be the Filter sizei.e., the number of input elements the filter considers and '$x_{t-d.i}$' be the input element considered for the 'i^{th}' filter weight. The convolution process at time 't' using a dilated causal filter is described aseq(14).This equation guarantees that, for an output at time t, the filter exclusively takes into account elements from the past. By increasing the dilation factor, 'd', the network can effectively capture long-range dependencies.

$$y_t = \sum_{i=0}^{k-1} w_i \cdot x_{t-d \cdot i} \tag{14}$$

Following each convolutional layer, Batch Normalization is used to standardize theactivations within a "mini-batch" to achieve zero mean and unit variance. Enhance training efficiency by minimizing internal covariate shifts. Increase stability by mitigating the issues of exploding or vanishing gradients.

Temporal Convolutional Networks (TCNs) exhibit a remarkable capacity for capturing long-term dependencies through the utilization of dilated convolutions, all while maintaining a consistent model architecture without necessitating an increase in depth. In contrast to recurrent neural networks (RNNs), TCNs operate independently of recurrent connections, mitigating challenges associated with gradient issues such as vanishing or exploding gradients. The application of convolutions facilitates the concurrent processing of the entire sequence, which contributes to a more expedited training process compared to the sequential nature of RNNs. Furthermore, causal convolutions guarantee that the predictive outcomes adhere to the chronological ordering intrinsic to the data.

2.3.4 Proposed Model Architecture -Hybrid TCN-Bi-GRU-LSTM Model

The TCN-Bi-GRU-LSTM architecture is a hybrid neural network framework that combines the advantages of TCNs, Bi-GRUs, and LSTMs. This configuration empowers the model to capture local dependencies via convolutions and long-range dependencies through recurrent units, making it especially efficient for applications like time series forecasting. The model employs a layered design in which every component has a distinct function in the extraction, modelling, and prediction of patterns within sequential data. The learning Framework of the proposed model is as shown in Fig.3

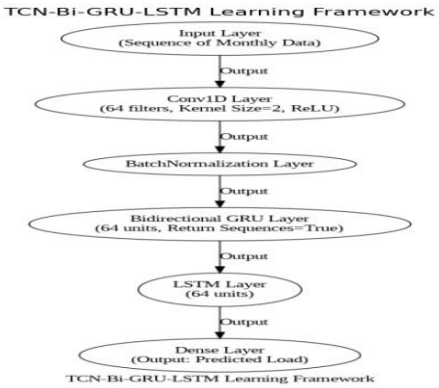

Fig.3 Learning Framework of the Proposed TCN-Bi-GRU-LSTM Model

a. Extracting Local Patterns with Conv1D Layer: The 1D Convolutional layer (Conv1D) identifies short-term or local dependencies present in the input sequences. It moves a filter along the sequence, recognizing patterns within predetermined-length windows (e.g., daily or hourly trends). In eq(15) Conv1D utilizes a filter on the input sequence 'x_t', resulting in an output 'y_t'. This layer plays a vital role in capturing local temporal patterns within the data, such as sudden increases or decreases.

$$y_y = \text{Conv1D}(x_t) \tag{15}$$

b. Stabilized training through Batch Normalization: Following the Conv1D layer, Batch Normalization guarantees that activations are standardized across mini-batches, enhancing training speed and mitigating overfitting. It minimizes internal covariate shifts, which helps the network to converge more consistently.

C. Capturing Past and Future Dependencies using Bidirectional GRU Layer: The Bi-GRU layer analyses the input sequence in both forward and backward orientations as illustrated in eq(16). This enables the model to grasp future context alongside past dependencies, which is especially beneficial for forecasting loads affected by upcoming events e.g., peak electricity demands. GRUs are efficient recurrent units that merge the forget and input gates, resulting in faster and more computationally efficient performance compared to LSTMs. The bi-directionality allows the model to comprehend influences from both past and future.

$$h_t = B_i\text{-GRU}(y_t) \tag{16}$$

D. Processing Long-Term Dependencies by adding LSTM Layer: The LSTM layer is incorporated to effectively capture long-range dependencies within the data. LSTMs manage vanishing gradients efficiently, rendering them ideal for recognizing patterns over extended sequences as in eq(17). This layer guarantees that the network preserves crucial information from previous steps, even when dealing with lengthy sequences.

$$O_t = \text{LSTM}(h_t) \tag{17}$$

e. Generating the Final Prediction via Dense Layer: The output from the LSTM layer is directed through a Dense (fully connected) layer to produce the ultimate prediction as in eq(18).Here, 'y^t', represents the predicted value, W_o denotes the weight matrix, and b_o signifies the bias term. The Dense layer consolidates the features acquired by the preceding layers to deliver the final output (for instance, a forecasted load or time-series value).

$$\hat{y}t = W_o \cdot o_t + b_o \tag{18}$$

Integrating TCN, Bi-GRU, and LSTM enhances the model by allowing Conv1D (TCN) to identify local patterns within brief time intervals, thereby elevating feature extraction. The Bidirectional GRU effectively captures dependencies in both directions, illustrating how past and future events impact the present time step. LSTM proficiently manages long-term dependencies, guaranteeing that the network preserves crucial information throughout extended sequences. This synergy ensures that the model can effectively generalize for intricate time series data, utilizing local, global, and bidirectional dependencies to yield reliable predictions. In load forecasting, where electricity usage is influenced by both short-term trends (such as daily patterns) and long-term behaviours (like seasonal variations), this architecture excels as Conv1D effectively captures hourly or daily changes. Bi-GRU considers past and future occurrences that affect load patterns. LSTM guarantees that the model can manage sequences that span across various time frames, including multi-week or multi-month predictions.

III. Training Process, Prediction and Evaluation

3.1 Loss Function

In deep learning frameworks like LSTM, GRU, or their hybrid configurations (for instance, TCN-Bi-GRU-LSTM), the choice of loss functions and optimization techniques is vital for effective model training.

a. Mean Squared Error (MSE) and Mean Absolute Error (MAE)

The MSE represents one of the most extensively employed loss functions within the domain of regression analysis, wherein the objective is to reduce the discrepancy between the forecasted values and the corresponding actual values using eq(19). The characteristics of MSE incorporate the squaring of differences, which intensifies larger errors and renders the model more responsive to significant discrepancies. Calculating gradients is

straightforward, facilitating seamless optimization. The errors are averaged across all data points to derive a significant metric for assessing model performance.

In addition to MSE, MAE serves as another prevalent loss function employed in regression tasks. The MAE quantifies the average of the absolute discrepancies between predicted values and actual outcomes using eq(20). Unlike MSE, it does not square the errors, making it less responsive to significant deviations. The characteristics of MAE encompass Linearity, by taking the absolute value of the error rather than squaring it, MAE assigns equal importance to all errors, irrespective of their magnitude. In contrast to MSE, which exaggerates larger errors, MAE is more robust in handling outliers due to its non-squaring of deviations. The absolute value causes MAE to be non-differentiable at zero. Nevertheless, in practical applications, this challenge is addressed by utilizing subgradients during the optimization process. In this work, MSE and MAE are used to analyse and compare the performance of the models.

$$MSE=\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i-y_i\right)^2 \tag{19}$$

$$MAE=\frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i-y_i\right| \tag{20}$$

Where

- $\hat{y}_i$ is the predicted value at index i
- $y_i$is the actual value at index i
- n is the total number of data points

3.2 ADAM optimizer

The Adam optimizer enhances stochastic gradient descent (SGD). It merges the benefits of Momentum and RMSProp. Adam adjusts weights using gradients to minimize loss. It tailors the learning rate for each parameter, making it suitable for complex models. It also corrects bias during initial noisy training steps. Like MSE, Adam optimizes the MAE loss by iteratively adjusting parameters based on gradients. However, MAE's absolute values lead to different gradient calculations than MSE. Adam maintains moving averages of gradients and their squares for smoother updates. It adjusts learning rates for each parameter, enhancing performance on sparse data. The optimizer remains consistent even in the early training stages. The weights are updated using eq(21).

$$\theta_{t+1}=\theta_t-\eta\cdot\nabla_{\theta_t}J(\theta_t) \tag{21}$$

where,

- θ_t are the model parameters(weights) at time 't'
- θ_{t+1} are the updated parameters for the next step
- $\nabla_{\theta_t} J(\theta_t)$ gradient of the loss function $J(\theta_t)$to the parameters
- η is the learning rate, which controls the size of the update step.

Optimization Process:

- Randomly initialize weights or use a set scheme.
- Predict the output using the current weights.
- Calculate loss from predictions and true labels.
- Compute the gradient of the loss function to weights.
- Adjust the weights using the Adam optimizer.
- Repeat the process for many epochs until convergence.

3.3 Training and Evaluation

3.3.1 Dataset: The data utilised for training and assessment encompasses hourly load information from a regional utility spanning one year. This information is partitioned into 70% for training, 15% for validation, and 5% for testing.

3.3.2 Learning Framework Architectures:

a. LSTM Model: Fig.4 shows the LSTM model's learning framework. The following describes the layers and their roles in processing input data.

Layer-1(LSTM):

- **Input Shape:** (None, 4, 1), The input for this layer consists of a sequence with a length of 4 and 1 feature at each time step. The None indicates that the batch size can be variable (i.e., the number of sequences processed simultaneously).
- **Output Shape:** (None, 4, 64) This LSTM layer produces 64 units (features) for each of the 4-time steps. As it returns sequences, the time dimension (4) is preserved in the output.

This layer handles the input sequence (4-time steps, 1 feature) and generates 64 features for each time step

Layer-2(LSTM):

- **Input Shape:** (None, 4, 64) The input here is derived from the output of the preceding LSTM layer.
- **Output Shape:** (None, 64) This LSTM layer provides only the final hidden state (i.e., it does not

return sequences). This hidden state encompasses 64 units, encapsulating the sequential

information processed up to this point.

Further processes the sequence and yields only the last hidden state with 64 units.

Layer-3(Dense):

- **Input Shape:** (None, 64) The Dense layer receives the 64 units from the prior LSTM layer.
- **Output Shape:** (None, 1) It produces a single unit, which is likely a regression output (e.g., estimating a continuous value like load or temperature).

It transforms the 64 hidden units into one output, which may signify a regression prediction.

This model is probably employed for time-series forecasting or load prediction, wherein the LSTM

layers effectively capture temporal dependencies, and the concluding Dense layer forecasts a continuous

target value.

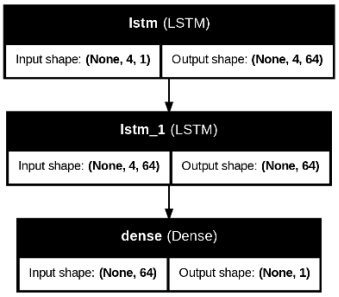

Fig.4 Learning Framework Architecture for LSTM Model

GRU-LSTM Model: Fig.5 shows the learning framework of the GRU-LSTM model. The description of the layers and their collaborative functioning in processing sequential data is as follows:

Layer-1 (GRU) :

- **Input Shape:** (None, 4, 1) The input consists of a sequence with a length of 4 and 1 feature for each time step. None indicates that the batch size can vary.
- **Output Shape:** (None, 4, 64) This GRU layer generates 64 units (features) for each of the 4-time steps. The GRU layer produces sequences, which means it maintains the time dimension in its output.

This layer captures temporal dependencies present in the input sequence and outputs 64 features per time step.

Layer-2 (LSTM):

- **Input Shape:** (None, 4, 64) The input is derived from the preceding GRU layer, which offered 64 features across the 4-time steps.

- **Output Shape:** (None, 64) This LSTM layer only outputs the final hidden state. This process condenses the sequential information, thereby reducing the time dimension.

Further processes the sequence and provides the last hidden state, encapsulating the sequence's essence.

Layer-3 (Dense):

This layer Converts the 64 units into a single continuous value, making it ideal for time-series forecasting or load prediction.

InHybrid GRU-LSTM, GRU effectively manages shorter-term dependencies and is quicker to train. LSTM will be more adept at capturing long-term dependencies, although it is more intricate than GRU. This hybrid architecture harnesses the efficiency of the GRU and the advanced pattern recognition capabilities of the LSTM, resulting in a well-rounded model that is suitable for tasks requiring both short-term and long-term dependencies.

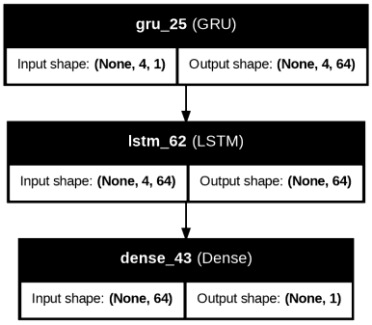

Fig.5 Learning Framework Architecture for GRU-LSTM Model

c. Bi-GRU-LSTM Model: Fig.5 shows the learning framework of the Bi-GRU-LSTM model consisting of three main layers

Layer-1 (Bidirectional LSTM):

- **Input shape:** (None, 4, 1)the input has sequences of length 4, with each time step containing 1 feature. None signifies the batch size, which can vary.

- **Output shape:** (None, 4, 128) Following processing, the layer produces a sequence of length 4, with each time step represented by 128 units. Given that it is a Bidirectional LSTM, this output is likely a combination of forward and backward hidden states.

Layer-2 (LSTM) :

- Input shape: (None, 4, 128) This layer accepts the output from the Bidirectional LSTM, which comprises a sequence of 4-time steps, each possessing 128 features.

- Output shape: (None, 64) The LSTM condenses this information into a single output vector of 64 units, removing the sequence dimension.

Layer-3 (Dense):

- Input shape: (None, 64) The Dense layer takes in the 64-unit output from the LSTM.

- Output shape: (None, 1) This layer compresses the output to a singular value, which likely acts as the model's final prediction.

This model integrates the advantages of both Bidirectional and standard LSTM layers, which are frequently employed for sequence processing tasks such as time-series forecasting or natural language processing. The concluding Dense layer is likely intended for regression or binary classification.

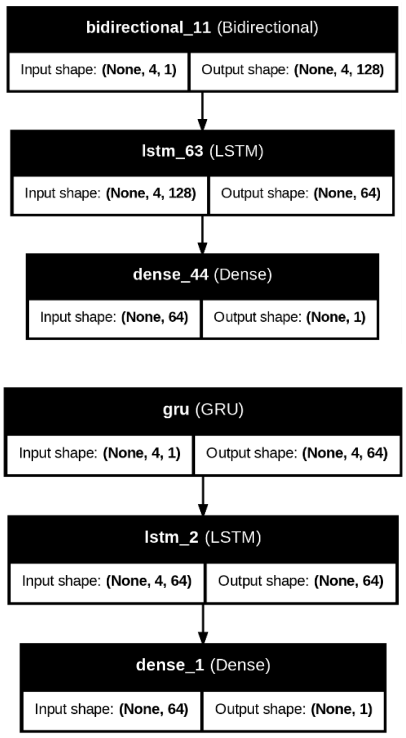

Fig.5 Learning Framework Architecture for Bi-GRU-LSTM Model

a. **TCN-Bi-GRU-LSTM Model:** Fig.6illustrates a more intricate learning framework of the TCN-based Bi-GRU-LSTM model that integrates both convolutional and recurrent layers. Below is a comprehensive summary of the layers constituting the model.

Input Layer(None,6,1):

➢ Input shape: (None, 6, 1) —This allows processing of various sequence counts, with each sequence having 6 time steps and 1 feature. This layer defines the input data's dimensionality, suitable for univariate time series, but can be modified for multivariate data by adding features.

1D Convolutional Layer(None,4,128) :

➢ The 1D convolutional layer uses 128 filters on the input sequence. These filters extract local features from the input over small time windows. They may identify patterns or changes in the input signal.

➢ The output shape (None, 4, 128) shows a sequence length reduction to 4 time steps with 128 feature maps per step. This reduction is likely due to the convolution kernel size, probably 3.

➢ Conv1D is effective for detecting local patterns in time-series data like spikes or shifts. It is often used for time-series classification, audio processing, or any sequential data with local dependencies.

Batch Normalization (None,4,128):

➢ Batch Normalization adjusts and scales layer outputs. It modifies mean and variance of input data during training. This normalization stabilizes and accelerates training.

➢ Output shape: (None, 4, 128) The output shape remains (None, 4, 128) since it does not alter data shape.

➢ Batch normalization reduces internal covariate shift, promoting faster convergence and minimizing local minima issues. It also serves as regularization, which may lower overfitting.

Bidirectional LSTM(None,4,64):

➢ This is a Bidirectional LSTM layer. In contrast to standard LSTMs, Bidirectional LSTMs allow data to flow in both directions, providing context from both past and future. This feature is especially beneficial when future data can impact the output at each time step.

- The output shape (None, 4, 64) indicates that there are 64 features per 4 time steps after processing. The 64 features derive from the bidirectional nature of the LSTM, with each direction contributing 32 units.
- Bidirectional layers are highly advantageous for tasks that require understanding past and future contexts simultaneously. For instance, in areas like speech recognition or financial predictions, future events can influence the interpretation of prior data.

LSTM Layer (None,96)

- This is a standard LSTM layer, a type of recurrent neural network. LSTMs effectively retain information, solving the vanishing gradient issue found in regular RNNs.
- Here, the LSTM compresses the previous layer's output into a 96-unit vector.
- The output shape (None, 96) indicates that the LSTM provides a 96-feature vector for each sequence, capturing learned dependencies from the input.
- LSTMs excel at identifying long-term dependencies in sequential data, making them valuable for tasks like time-series forecasting, where patterns may be delayed or spread out.

Dropout Layer (None, 96)

- Dropout is a technique to reduce overfitting. It randomly ignores a percentage of neurons during training, promoting redundant learning and reducing reliance on specific features.
- The shape (None, 96) stays the same as dropout does not change data dimensions.
- Dropout is essential for improving model generalization to new data. Without it, the model risks memorizing training data, leading to poor test performance.

Dense Layer:

- The dense layer is a fully connected layer with one output neuron. It processes 96 features from the previous layer to generate a single output.
- The shape (None, 1) signifies that the model yields one value per sequence, which can represent a continuous value, a probability, or another scalar output.
- This layer is the model's final output, used for predictions in tasks like time-series forecasting or classification.

This architecture integrates various layers for effective time-series data processing, Conv1D captures local patterns. LSTM captures long-term dependencies. Bidirectional LSTM utilizes both past and future context. Dropout and Batch Normalization enhance generalization and stability. The final Dense layer outputs a single scalar for regression or binary classification.

3.3.2 Hyperparameters: The models were optimized using the Adam algorithm, while hyperparameters like learning rate and batch size were adjusted through cross-validation and are listed in Table.I.

Table.I List of hyperparameters

S.No	Description	Value
1	Optimizer	ADAM
2	Batch Size	64
3	Learning Rate	0.001
4	Epochs	50

3.3.3 Prediction and Evaluation:

Once the model has undergone training, it is utilized to predict the load values for the test dataset. This procedure entails feeding the test sequences into the trained model. Each model is evaluated by utilising metrics like RMSEand MAE for short-term forecasts across i.e., daily, weekly, and monthly time frames.

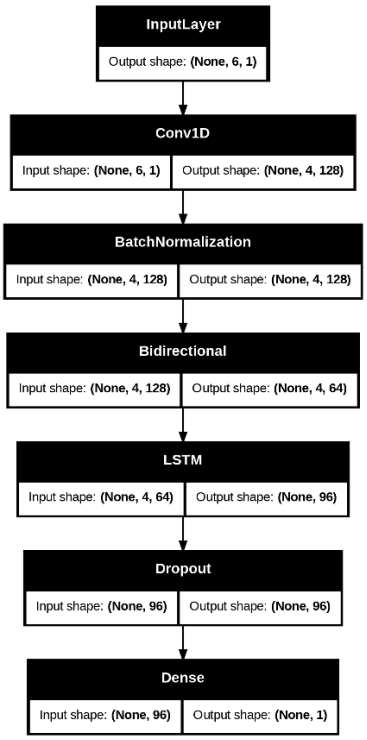

Fig.6 Learning Framework Architecture for TCN-Bi-GRU-LSTM Model

3.3.4 Psuedocode

Step 1: Import Libraries

Commence by importing essential libraries pertinent to data manipulation, visualization, and machine learning endeavors:

- numpy, pandas, matplotlib

- sklearn (designated for preprocessing and performance metrics)

- tensorflow (utilized for deep learning architectures)

- IPython (employed for the visualization of model architectures)

Step 2: Load and Process Data

Initiate the process by loading CSV data containing timestamps and load values.

Transform the 'Datetime' column into a standardized datetime format.

Extract relevant features from the Datetime variable:

- Hour, Day, Month, Day of the Week

Delineate input features (X) from the target variable (y).

Step 3: Train-Test Split

Divide the dataset into an 80% training subset and a 20% testing subset.

- X_train, y_train (designated as the training set)

- X_test, y_test (designated as the testing set)

Step 4: Normalize Data

Utilize MinMaxScaler for the normalization of the dataset:

- Fit the scalers on the training dataset and subsequently transform both training and testing datasets.

- Reshape the input data to conform to the (samples, timesteps, features) framework.

Step 5: Define Models

1. LSTM Model:

- Incorporate two stacked LSTM layers, each comprising 64 units.

- Integrate a dense layer designated for output.

- Compile the model utilizing the Adam optimizer alongside mean squared error (MSE) as the loss function.

2. GRU-LSTM Hybrid Model:

- Implement a GRU layer succeeded by an LSTM layer.

- Incorporate a dense output layer.

3. Bidirectional GRU-LSTM Model:

- Employ a bidirectional GRU layer followed by an LSTM layer.

- Integrate a dense output layer.

4. TCN-Bidirectional-GRU-LSTM Model

- Utilize a Conv1D layer for the extraction of temporal patterns.

- Apply Batch Normalization.

- Incorporate both bidirectional GRU and LSTM layers.

- Integrate a dense layer for output.

Step 6: Train Models

For each designated model:

- Fit the model on the training dataset for a duration of 50 epochs, utilizing a batch size of 32.

- Employ validation data for the assessment of performance.

- Archive training history for subsequent analysis.

Step 7: Forecasting Function

Function: forecast(model, X, steps, freq, start_date)

- Generate predictions for a predetermined number of future steps.

- Utilizes the most recent prediction as input for subsequent predictions.

- Return a DataFrame containing the predicted load alongside corresponding timestamps.

Step 8: Generate Predictions

Establish forecast horizons:

- Day (24 hours), Week (7 * 24 hours), Month (30 * 24 hours)

For each model and defined forecast horizon:

- Generate predictions by employing the forecast function.

- Store predictions for evaluative purposes.

Step 9: Calculate Performance Metrics

Function: calculate_metrics(y_true, y_pred)

- Compute RMSE and MAE as a percentage of the mean actual load.

For each model and designated forecast period:

- Display RMSE and MAE performance metrics.

Step 10: Visualize Results

Function: plot_forecasts(actual, predicted, title)

- Illustrate actual versus predicted values for comparative analysis.

For each model and designated forecast period:

- Execute the comparative visualization utilizing matplotlib.

End of Pseudocode

IV. Results and Discussions

The four models viz., LSTM, GRU-LSTM, Bi-GRU-LSTM, and TCN-Bi-GRU-LSTM were assessed for their electrical load prediction over daily, weekly, and monthly intervals by considering Fig.7 as actual load data set. The forecasts produced by each model were evaluated against the actual load values across daily, weekly, and monthly time frames. The learning curves (loss vs. epochs) for each model offer valuable insights into their ability to learn from the data as time progresses. The learning curves illustrate both training loss and validation loss, enabling to assess whether the models were effectively simplifying or surrendering to overfitting.

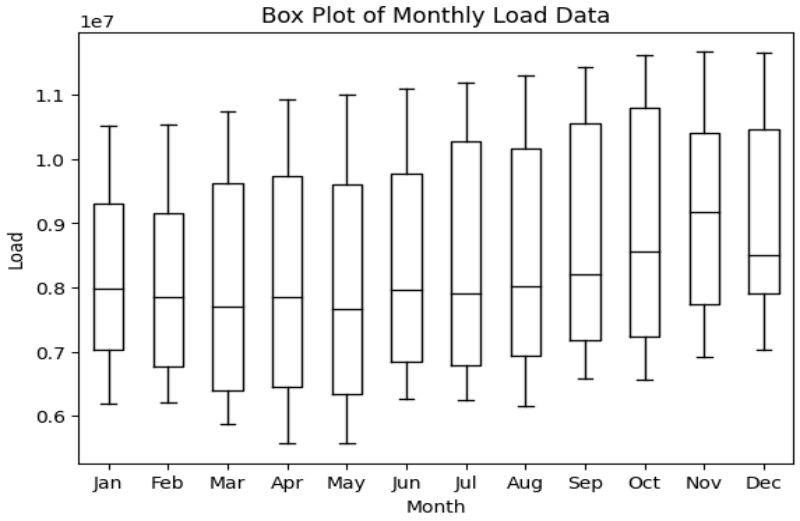

Fig.7 Actual Load

4.1 LSTM Model

The LSTM design included two successive LSTM layers. While LSTMs are proficient at recognizing sequential patterns, this setup is rather basic and may fall short in effectively managing intricate long-term dependencies. The LSTM model exhibited a relatively smooth learning curve as shown in Fig.8; however, there was a noticeable gap between the training and validation losses. This indicates that the model may be overfitting to the training data, capturing specific patterns that fail to generalize to unseen data (validation).

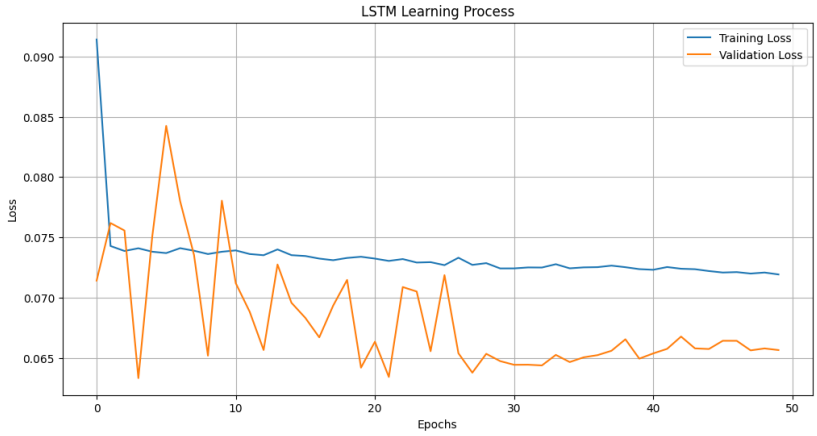

Fig.8 Learning Process of LSTM Model

Daily Forecast: The RMSE is 20.62% and MAE is 20.58% values were moderate, suggesting that the model was able to capture daily trends quite effectively.

Weekly and Monthly Forecasts: The %RMSE and % MAE are 17.67% & 15.71% for weekly and 18.44% & 16.65% for monthly forecast which were higher and indicates that the LSTM encounters difficulties with long-term predictions. Although the LSTM effectively handles sequential data, it may face challenges related to vanishing gradients as the time series lengthens.

4.2 GRU-LSTM Model:

Integrating a GRU layer before the LSTM leveraged the benefits of both GRUs which provide faster training and a more efficient architecture and LSTMs that are recognized for their exceptional long-term memory capabilities. However, it still constitutes a rather traditional architecture.

This model exhibited a modest enhancement in generalization when compared to the LSTM model. However, as seen in Fig.9 as the number of epochs rose, a noticeable disparity between training and validation loss remained, suggesting a slight occurrence of overfitting.

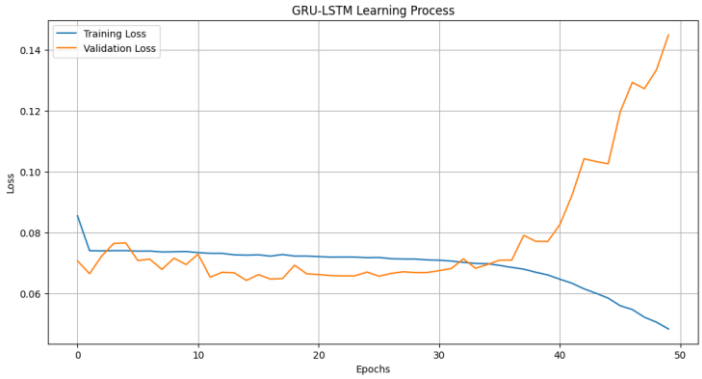

Fig.9 Learning Process of GRU-LSTM

Daily Forecast: The GRU-LSTM model demonstrated modest enhancements compared to the LSTM model, attributed to the GRU's ability to handle sequential data while avoiding the vanishing gradient problem more effectively than LSTMs yet the %RMSE and %MAE are 24.94 and 24.91 respectively

Weekly and Monthly Forecasts: Even though it performed better than the LSTM model, the %RMSE and %MAE respectively remained relatively elevated 19.47 and 17.19, particularly for longer-term forecasts.

4.3 Bi-GRU-LSTM Model:

The bidirectional GRU layer analyses the input sequence in both forward and backward directions, offering greater context at every time step. This improves the model's capability to identify patterns, particularly in noisy or non-stationary data.

The learning curves shown in Fig.10 for this model exhibited greater stability, with the validation loss closely mirroring the training loss. This suggests that the bidirectional GRU layer enhanced the model's ability to generalize, thereby minimizing overfitting.

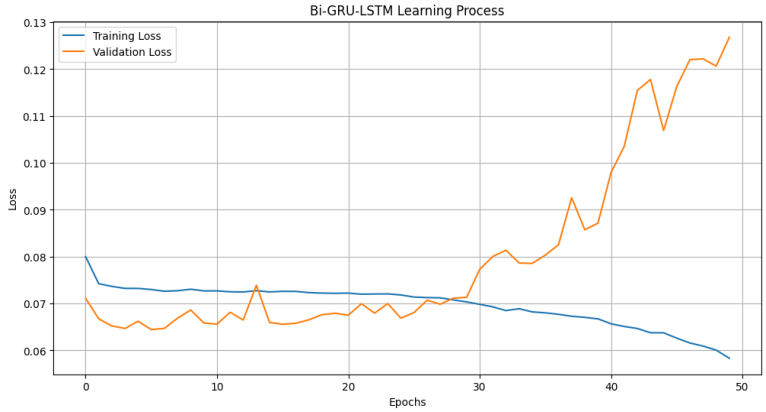

Fig.10 Learning Process of Bi-GRU-LSTM Model

Daily Forecast: The bi-directional characteristic of the GRU layer allowed this model to recognize patterns more efficiently by analyzing data in both forward and backward directions. The %RMSE stands at 25.21 and %MAE is recorded at 24.49.

Weekly and Monthly Forecasts: This model outperformed both the LSTM and GRU-LSTM models, showing that the bidirectional GRU enabled it to consider a wider context from the time series. The %RMSE and %MAE for weekly are 20.61 and 17.28, while for monthly they are 21.39 and 17.28.

4.4 .TCN-Bi-GRU-LSTM Model:

This architectural design is notable for its complexity, combining Temporal Convolutional Networks (TCN) with Bidirectional GRU and LSTM layers. The TCN utilizes convolutional filters to identify short-term trends in the dataset, whereas the GRU and LSTM layers focus on long-term dependencies. This combined architecture allows the model to learn patterns more adaptively and effectively across different temporal scales.

The TCN-Bi-GRU-LSTM model exhibited the most stable and uniform learning curve, showing negligible disparity between training and validation loss as seen in Fig.11. This suggests that the model effectively absorbed insights from the data without overfitting, thus establishing itself as the most resilient among all models.

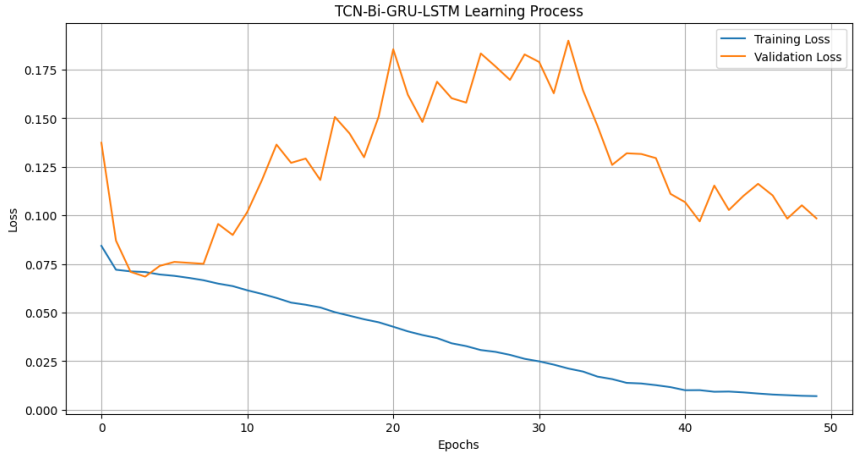

Fig.11 Learning Process of TCN-Bi-GRU-LSTM Model

Daily, Weekly, and Monthly Forecasts: This analytical framework consistently exhibited superior performance compared to alternative models across all temporal horizons. The integration of Temporal Convolutional Networks (TCN), which employ convolutional filters on the sequential data, facilitated the model's capacity to proficiently learn both short-term and long-term dependencies. The TCN effectively captures intricate, localized

patterns, whereas the GRU and LSTM layers adeptly manage more extensive, temporal dependencies.The TCN-Bi-GRU-LSTM model demonstrated superior overall efficacy, especially excelling in long-term predictive analyses where alternative models encountered significant difficulties. The %RMSE and % MAE are recorded as 10.23 & 9.18 for Daily forecast, 20.13 & 16.94 for Weekly forecast and for monthly forecast %RMSE and % MAE is recorded as 17.99 and 15.18.

4.3. Comparison of Forecasts for Four Models

Daily Forecast: The quarter hourly predictions for one day are shown from Fig. 12 to 15 for LSTM, GRU-LSTM, Bi-GRU-LSTM and TCN-Bi-GRU-LSTM models, respectively. The four models demonstrated impressive capabilities in predicting daily load patterns, with the TCN-Bi-GRU-LSTM showing the closest correlation to the actual data. The models were effective in reflecting the daily fluctuations in load, although the simpler models such as LSTM, GRU-LSTM exhibited around discrepancies with the actual values.

Weekly Forecast: The performance of the simpler models,LSTM, GRU-LSTM declined over the weekly period as can be seen in Fig.16 and Fig.17 resulting in considerable discrepancies from the actual data. In contrast, the Bi-GRU-LSTM and TCN-Bi-GRU-LSTM models were able to follow the weekly trends with greater precision, particularly regarding the detection of peaks and troughs in the load pattern as seen in Fig.18 and Fig.19, respectively.

Monthly Forecast: The monthly time frame presented the most significant challenge. While can be seen in Fig.20 and Fig.21, the LSTM and GRU-LSTM models experienced substantial deviations from the actual data, the TCN-Bi-GRU-LSTM maintained a commendable level of Although, Fig.22, the Bi-GRU-LSTM model was more effective in capturing long-term dependencies compared to the LSTM and GRU-LSTM models, it fell short of the performance exhibited by the TCN-Bi-GRU-LSTM model accuracy.

The incorporation of convolutional layers within the TCN architecture enabled the model, Fig.23 to grasp larger temporal dependencies, enhancing its effectiveness in forecasting long-term trends. The Fig.24 shows the comparison of %RMSE and %MAE for different models across day, week, and month forecasts. Table.II shows the numerical comparison of %RMSE and %MAE for different models across day, week and monthly forecasts.

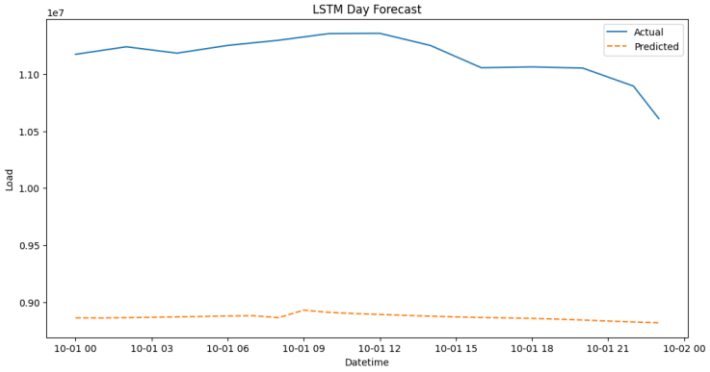

Fig.12 Quarter Hourly load on 10-01-2023 using LSTM Model

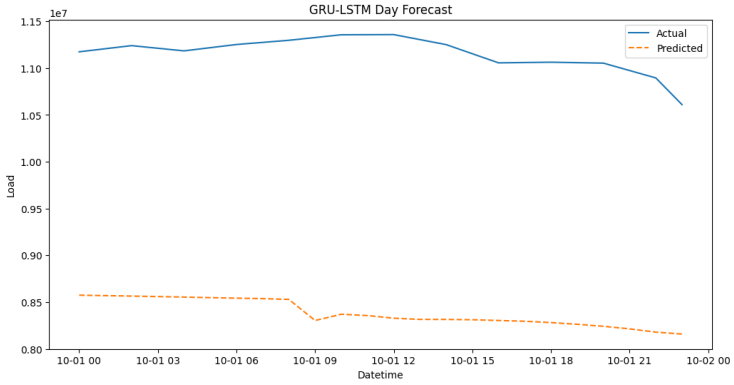

Fig.13 Quarter Hourly Load using GRU-LSTM Model

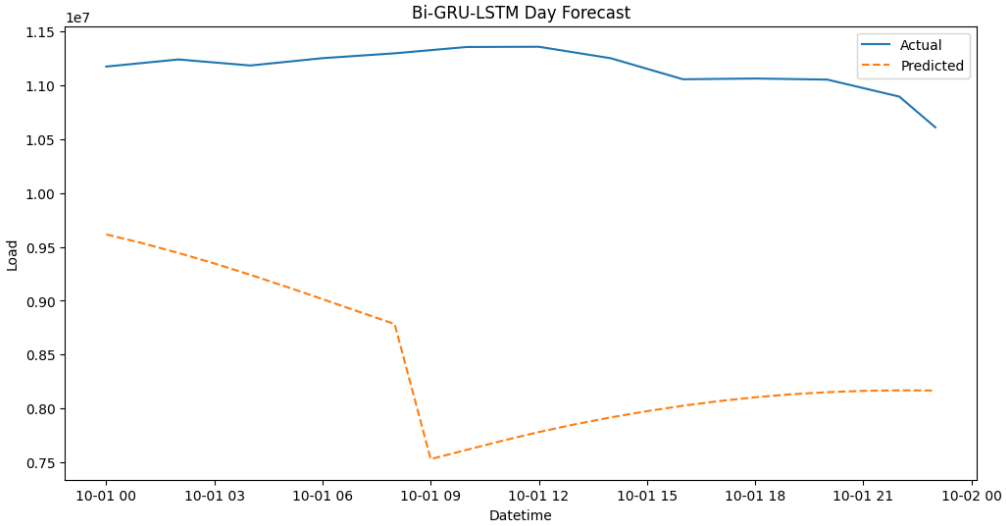

Fig.14 Quarter Hourly Load using Bi-GRU-LSTM Model

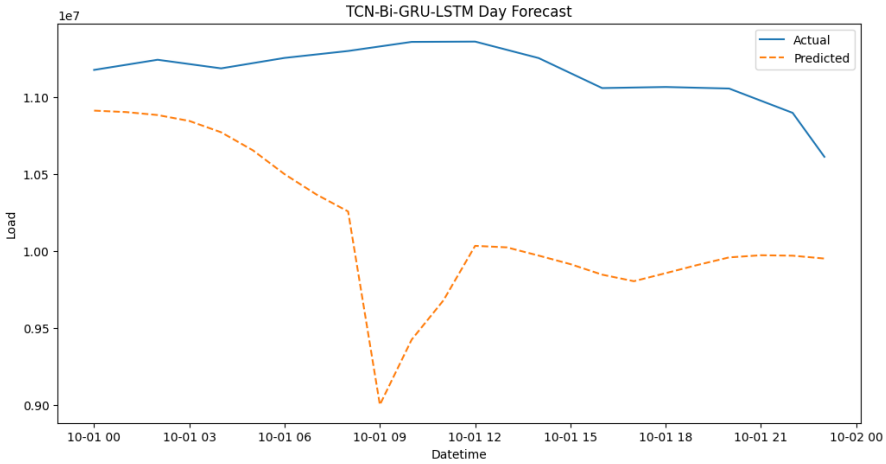

Fig.15 Quarter Hourly Load using TCN-Bi-GRU-LSTM Model

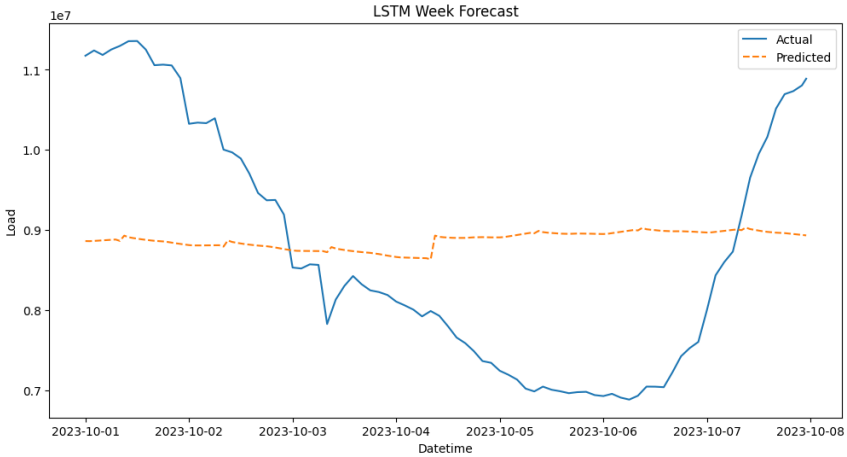

Fig.16 Week Ahead Load using LSTM Model

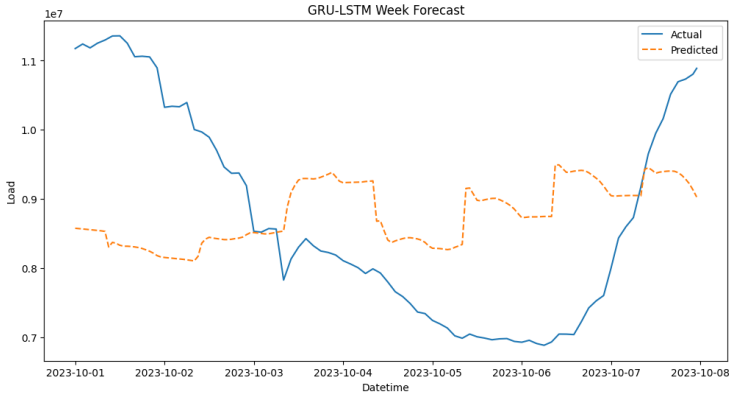

Fig.17 Week Ahead Load using GRU-LSTM Model

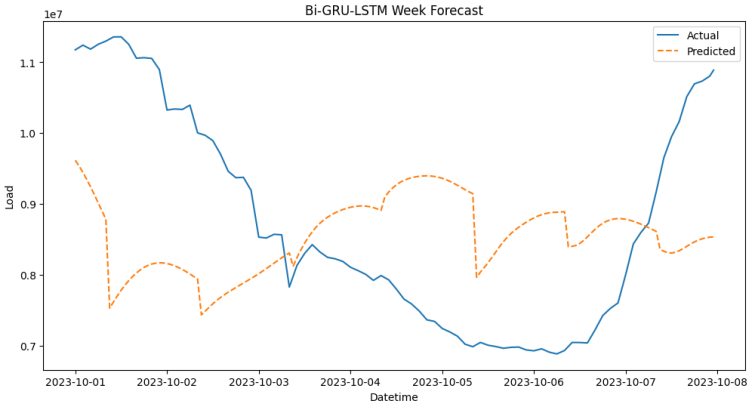

Fig.18 Week Ahead Load using Bi-GRU-LSTM Model

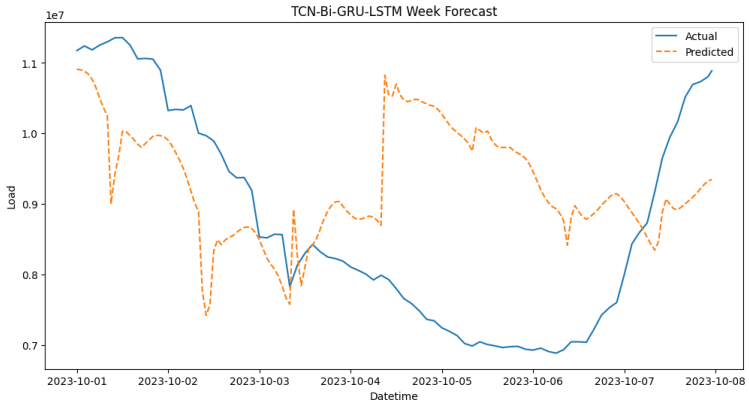

Fig.19 Week Ahead Load using TCN-Bi-GRU-LSTM Model

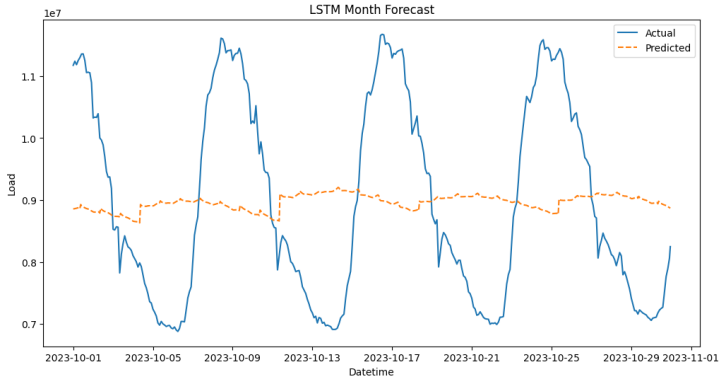

Fig.20 Month Ahead Load using LSTM Model

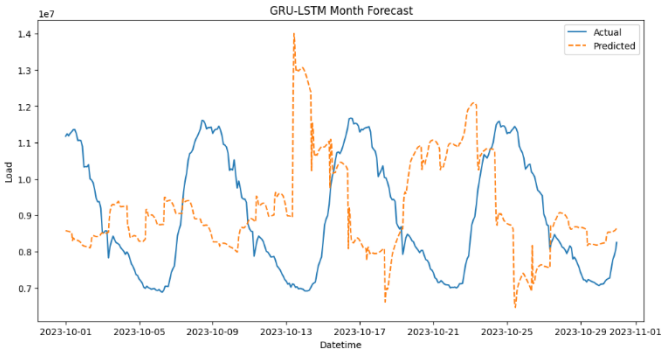

Fig.21 Month Ahead Load using GRU-LSTM Model

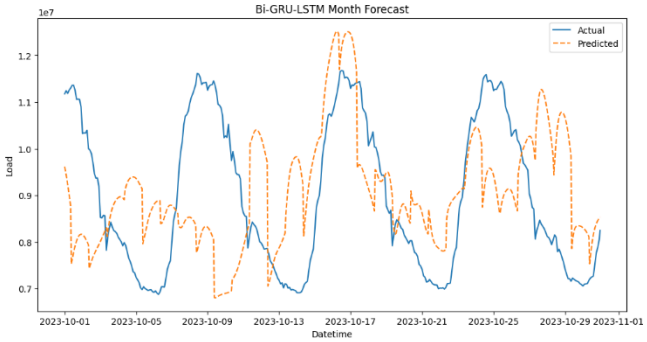

Fig.22 Month Ahead Load using Bi-GRU-LSTM Model

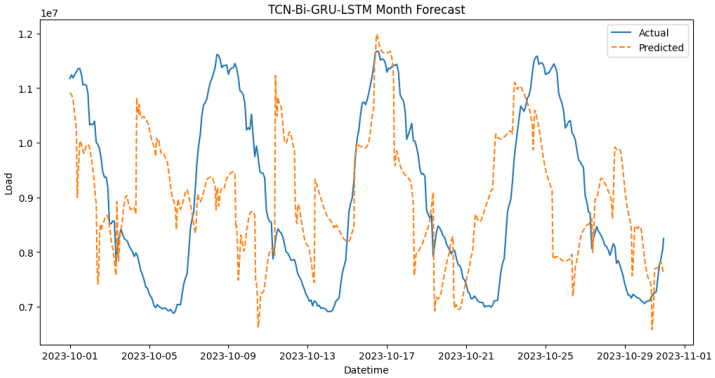

Fig.23 Month Ahead Load using TCN-Bi-GRU-LSTM Model

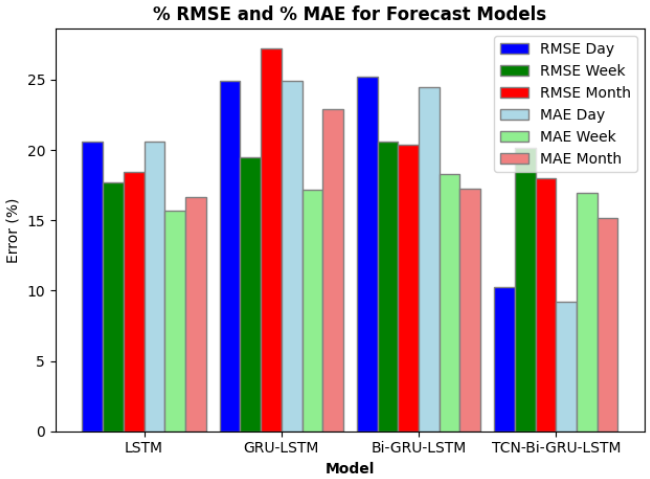

Fig.24. Comparison of % RMSE and % MAE across models

Table.II Numerical Comparison of %RMSE & %MAE

Model	Horizon	RMSE(%)	MAE(%)
LSTM	Day	20.62	20.58
	Week	17.67	15.71
	Month	18.44	16.65
GRU-LSTM	Day	24.94	24.91
	Week	19.47	17.19
	Month	27.26	22.92
Bi-GRU-LSTM	Day	25.21	24.49
	Week	20.61	17.28
	Month	20.39	17.28
TCN-Bi-GRU-LSTM	Day	10.24	9.18
	Week	20.13	16.94
	Month	17.99	15.18

V. CONCLUSION

This research underscores the efficacy of sophisticated deep learning architectures in the field of electric load forecasting, highlighting both the advantages and drawbacks of various models. The primary emphasis was placed on four unique architectures: LSTM, GRU-LSTM, Bi-GRU-LSTM, and the newly proposed TCN-Bi-GRU-LSTM. Through rigorous training and assessment on a thorough dataset, several key insights can be observed. Models showed differing accuracy levels depending on the forecasting horizon, with LSTM and GRU-LSTM struggling with monthly load predictions, highlighting the need for careful model selection. The choice of architecture significantly impacts forecasting performance, with TCN-Bi-GRU-LSTM excelling in learning long-term dependencies and improving accuracy for long-term predictions. The success of TCN-Bi-GRU-LSTM suggests that combining convolutional and recurrent neural networks effectively captures both local features and sequential data.Accurate load forecasting is essential for grid management and resource allocation; the findings can help energy providers enhance forecasting models for better decision-making. Finally, this research emphasizes the significance of employing advanced and hybrid deep learning architectures for electric load forecasting. By showcasing the potential of the TCN-Bi-GRU-LSTM model, this research contributes to the continuous advancement of predictive modelling techniques in the energy sector, ultimately striving for more efficient and sustainable energy management practices.

Future Scope: Future investigations will delve into additional refinements of the TCN-Bi-GRU-LSTM model, including the integration of attention mechanisms to prioritize significant historical data points. Moreover, broadening the dataset to encompass a wider range of conditions such as seasonal variations and economic indicators that could enhance the model's generalizability and precision.

REFERENCES

[1] Hodge. B. M., & Reddy. P., "Statistical methods for electricity load forecasting." *IEEE Transactions on Power Systems*, Vol.22(4), pgs.1180-1186, 2007.

[2] Bakar. A. A., & Yusof. N., "Review of forecasting methods for electricity demand." *Renewable and Sustainable Energy Reviews*, Vol.54, pgs.174-182,2015.

[3] Hochreiter. S., &Schmidhuber.J., "Long short-term memory." *Neural Computation*, Vol.9(8), pgs.1735-1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[4] Sutskever. I., Vinyals. O., & Le. Q. V., "Sequence to sequence learning with neural networks." In *Advances in Neural Information Processing Systems*, 27,2015.

[5] García-Gonzalez. J., et al., "Deep learning for short-term load forecasting: A review." *IEEE Access*, 8, 162386-162399,2020.

[6] Cheng, C., & Wu, J.. "A hybrid deep learning model for short-term load forecasting." *IEEE Access*, Vol.9, pgs.56464-56475,2021.
[7] Zhang, L., & Zhao, H. (2019). "Load forecasting using a hybrid model based on GRU and LSTM." *Energies*, 12(12), 2345.
[8] Chen, C., Zhang, Y., & Wu, J. (2021). "Short-term load forecasting using a bidirectional GRU-LSTM model." *IEEE Access*, 9, 86814-86823.
[9] Lechner, A. M., et al. (2018). "Short-term load forecasting with recurrent neural networks." *Applied Energy*, 232, 142-154.
[10] Bai, S., Kolter, J. Z., & Stadler, M. (2018). "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv preprint arXiv:1803.01271*.
[11] Yu, F., & Koltun, V. (2016**)**. "Multi-Scale Context Aggregation by Dilated Convolutions." *arXiv preprint arXiv:1511.07122*.
[12] Zhang, L., Jiang, Y., & Zhao, Y. (2022). "A TCN-Bi-GRU-LSTM model for short-term load forecasting." *Applied Energy*, 305, 117938.
[13] Teng, J., Wang, Y., & Xu, Y. (2019). "Short-term load forecasting based on LSTM neural network." 2019 IEEE 2nd International Conference on Power and Energy Engineering (ICPEE), pp. 1-6.
[14] Kumar, P., Kumar, A., & Saini, J. R. (2021). "Short-term load forecasting using LSTM network in smart grid." *Energy Reports*, 7, 193-203.
[15] Zhang, H., Yu, C., & Liu, Y. (2020). "A hybrid deep learning model for short-term load forecasting." *Energy Reports*, 6, 1455-1465.
[16] Wang, Y., Xu, L., & Ma, J. (2021). "Short-term load forecasting based on a hybrid GRU-LSTM model." *Energies*, 14(10), 2763.
[17] Zhang, Y., Wang, Y., & Feng, J. (2021). "Bidirectional GRU-LSTM model for short-term load forecasting." *IEEE Transactions on Power Systems*, 36(1), 565-575.
[18] Chen, C., Zhang, Y., & Wu, J. (2021). "Short-term load forecasting using a bidirectional GRU-LSTM model." *IEEE Access*, 9, 86814-86823.
[19] Bai, S., Kolter, J. Z., & Stadler, M. (2018). "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv preprint arXiv:1803.01271*.
[20] Zhang, C., et al. (2020). Hybrid Deep Learning Models for Load Forecasting. *IEEE Access*, 8, 1-12.
[21] https://sldc.aptransco.co.in/reports