# Short-Term Load Forecasting Based on Graph Convolution and Dendritic Deep Learning

Chunyang Zhang, Yang Yu, Tengfei Zhang, *Senior Member, IEEE*, Keyu Song, Yirui Wang, and Shangce Gao, *Senior Member, IEEE*

***Abstract*—Short-term load forecasting (STLF) is a significant task to the planning, operation and control of future power systems. The increasing number of devices connected to the system has led to more complex characteristics and forms of load, which has brought considerable difficulties to the relevant methods in achieving higher load prediction accuracy and reliability. In this regard, this study proposes a deep learning model that combines graph convolutional network (GCN), gated recurrent unit (GRU), and dendritic neural model (DNM) to forecast electric load more accurately. Firstly, the sample load data is constructed into graph data with individual time steps as nodes. A GCN is used to extract the hidden features while allowing a full communication between the time steps feature data. A GRU is then used to capture the time-dependent relationship of the data. Finally, a dendritic layer instead of a fully connected layer is used as the output to integrate data features in depth. Experiments are conducted to verify the validity of the proposed model and compared it with several effective deep learning models, including CNN_LSTM, Transformer and Kolmogorov-Arnold Networks (KAN). The results show a significant improvement in prediction compared to the baseline models, with mean absolute percentage error($MAPE$) of 1.62% and 3.98%, coefficient of determination($R^2$) of 0.983 and 0.928 respectively on two load datasets at different levels of aggregation, nationally and regionally.**

***Index Terms*—Dendritic learning, graph convolution network, GRU, load forecasting.**

## I. Introduction

CONSIDERING the fact that the task of power system load forecasting is playing an increasingly important role in power systems, it has become a research object favored by researchers recently. Load refers to the amount of electrical energy consumed in an area over a specific period of time. Depending on the level of aggregation, loads are categorized as national, regional and residential loads, etc., and are usually measured in kilowatt-hours (kWh) or megawatt-hours (MWh). For example, the sum of all the electrical energy consumed by a country in a day is known as the daily load of it. Short-term load forecasting (STLF) refers to the task of forecasting load data in the next few hours or days. Since electricity cannot be stored in large quantities for a long time [1], short-term load forecasting can help power dispatching centers accurately predict future power demand. Industries can thereby reasonably arrange energy generation plans, ensure a balance between supply and demand, and avoid energy shortages or surpluses [2]. A load forecast error of 1% can lead to a loss of hundreds of thousands of dollars per GWh [3], so its accuracy plays a pivotal role in economic decision-making in the power system.

In recent years, deep learning models have made great strides in time series tasks such as STLF. At the beginning, some single models are proposed. Among them, recurrent neural network (RNN) is specifically proposed for processing time series data [4]. Its variant, long short-term memory network (LSTM) [5], has become the most successful model for time series processing to date. W. Kong et al. [6] use LSTM with clustering techniques to achieve excellent performance in short-term residential load forecasting tasks, achieving better performance than other traditional models. Then, the unit structure of LSTM is simplified to improve its information processing efficiency without affecting its representation ability, gated recurrent unit (GRU) is proposed, and it has been widely used in time series tasks due to its excellent performance [7], [8]. Some single models that excel at extracting features from raw data, such as convolutional neural network (CNN), one of the most widely used and effective feature extraction models in the history of deep learning, are also used for time series tasks [9], [10], [11]. But CNN can only process Euclidean data and is generally not effective in processing non-traditional complex structured data such as circuit topology and social network graphs. So graph convolutional neural network (GCN) is proposed to extract features of non-Euclidean data [12], [13]. Y. Hu et al. [14] develop a spatiotemporal graph convolutional network for hourly energy predictions, and inter-building impacts are considered in graph-based method for prediction. H. Mansoor et al. [15] construct graph data by selecting feeder nodes with similar characteristics at the feeder level, which improves the prediction accuracy. Despite these single models have been iteratively innovating on

Received 19 November 2024; revised 21 February 2025; accepted 1 April 2025. Date of publication 8 April 2025; date of current version 27 June 2025. This work was supported in part by the Key Research & Development Plan of Jiangsu Province, China under Grant BE2023838, in part by the National Natural Science Foundation of China under Grant 62073173 and Grant 62203238, and in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant JP25K03179. Recommended for acceptance by Dr. Simone Silvestri. *(Corresponding authors: Yang Yu; Tengfei Zhang.)*

Chunyang Zhang, Yang Yu, Tengfei Zhang, and Keyu Song are with the College of Automation & College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: zhangchunyang616@gmail.com; yuyang@njupt.edu.cn; tfzhang@126.com; forrieysky@163.com).

Yirui Wang is with the Faculty of Electrical Engineering and Computer Science, Zhejiang Key Laboratory of Mobile Network Application Technology, Ningbo University, Zhejiang 315211, China (e-mail: wangyirui@nbu.edu.cn).

Shangce Gao is with the Faculty of Engineering, University of Toyama, Toyama-shi 930-8555, Japan (e-mail: gaosc@eng.u-toyama.ac.jp).

Digital Object Identifier 10.1109/TNSE.2025.3558193

their respective structures and some results have been obtained, it is difficult to achieve breakthroughs in forecasting accuracy in time series forecasting tasks using only single models.

In order to cope with the bottlenecks encountered by single models in related tasks, researchers begin to combine the advantages of specific models to form hybrid models. J. Song et al. [16] use CNN to extract load data features, and then use LSTM to process the time series relationship, which has higher prediction accuracy than separate models. W. H. Chung et al. [17] combine CNN, LSTM and attention mechanism to successfully solve the problem of district heater load. GRU is combined with CNN to achieve good results in load prediction tasks in different application scenarios, with both better accuracy and efficiency [18], [19]. GCN is also combined with various timing processing modules, S. Arastehfar et al. [20] consider the similarity of electricity usage patterns between users' homes, and they cluster them to generate graph data with each user as a node, which is used as training data to complete short-term residential electricity load forecasting along with GCN and LSTM. Compared with models that do not consider user relationships, this method gets better results. Numerous experiments have proved that the time series forecasting performance of hybrid models is better than that of single models, which is due to the co-operation between the different data processing modules in hybrid models. For example, using only single models such as CNN or GCN, which are used for feature extraction, for temporal prediction can result in the model failing to capture important temporal relationships between time-step features, thus reducing final accuracy. In turn, using only single models such as LSTM or GRU, which are applicable to sequence data, runs the risk of having noisy and interfering raw data being used as input directly due to the lack of a feature extraction step, resulting in the model acquiring too many erroneous features. In other words, hybrid models allow the modules to complement for each other's shortcomings to get better results.

What's more, whether in single or hybrid models, the fully connected layer is usually used as the output layer of the model structure. However, this layer only uses the calculation of weights to transmit information, without considering the nonlinear mechanism of the dendrites in the neuron in more detail. Therefore, it has been pointed out that the mechanism is oversimplified and may lose or even misunderstand some important feature information during the calculation [21]. Responding to this, a dendritic neuron model (DNM) based on neurobiology has attracted a lot of attention and has demonstrated excellent performance in many previous similar tasks [22], [23], [24]. This dendritic network can effectively solve linear indivisibility problems and has strong nonlinear expression capabilities. In addition, it also has a unique neuron pruning function that can automatically degenerate unnecessary synapses and dendrites, reducing computational complexity while retaining the most important information in the data [25]. Based on this type of neuron mode, great successes have been made in the field of time series forecasting. T. Zhang et al. [26] first decompose the data of photovoltaic power generation using wavelet transform technology, then use DNM to learn each sub-variable, and finally integrate the results for prediction. J. Ji et al. [27] combine a dendritic regression network with states of matter search optimization algorithm to successfully apply DNM to a wind speed prediction task. However, DNM has not been widely used in the field of short-term load forecasting [28], so our choice of DNM as the output layer of the load forecasting network is an empirical one. Many other researchers are constantly trying to optimize the structure and parameters of DNM to make it perform better [29], [30], [31]. To this day, DNM has become a strong contender to replace the fully connected layer as the model output layer.

Based on the above discussion, a new deep learning structure called GGDNM is proposed, which aims to explore a method with superior STLF prediction accuracy. In this hybrid model, time steps are used as nodes to construct the original load data into graph data, give corresponding adjacency matrices according to the logical relationship between time steps, and create edges between nodes. Then, a GCN is used to directly extract the features of the time graph data, so that the information between time steps can be fully exchanged and mapped to a higher dimension. The features are then input into a GRU, which captures the key dependencies between time series and aggregates important information to the hidden layer of the last time step. Finally, we replace the traditional fully connected layer with a DNM layer with stronger nonlinear processing capabilities to integrate hidden features from abstract data with stronger representation capabilities, and obtain the final regression prediction value.

The main contributions of our research can be summarized as follows:

1) A novel deep learning model based on graph convolution and dendritic network is proposed, aiming to achieve superior short-term load prediction accuracy and reliability. It does not require extra feature information, and innovatively uses GCN to directly extract features from the original load data. Considering the computational efficiency of hybrid models, a simpler structured GRU is used to extract the temporal relationships of the feature maps. In addition, a DNM is used as the output layer instead of the traditional fully connected layer, which improves the overall representation ability of the model.
2) A new perspective is used to view the load time series data, which is regarded as a straightforward graph structure, where the time steps are used as nodes and the time series relationship as the edge. Adjacency matrix is given to construct such graph structure load data. Then a GCN is used to extract its features. This improves the shortcomings of the traditional convolutional feature extraction mode, which is fixed and inflexible, and does not require additional spatial topological information.
3) The traditional fully connected layer is replaced by a dendritic layer as the output layer of the model, which uses its powerful computing and nonlinear processing capabilities to enhance the integration ability of the entire model, allowing the model to capture hidden information in the data and improve the accuracy of short-term load prediction tasks.
4) Compared with other competitive models, the proposed hybrid model has achieved the best result in a experiment

on a load dataset, evaluated by four metrics, successfully improving the prediction accuracy. This proves that the data construction, feature extraction, relationship capture and information integration methods we employed are effective.

The rest of this paper is structured as follows: Section II presents the relevant theory and introduces the theoretical framework of the proposed model. Section III describes the dataset, implementation details of the experiments, and give the analysis of the results and performance comparison with other benchmark models comprehensively. Section IV presents the discussion, which includes an ablation experiment and describes the effectiveness of model components. Section V presents the conclusion and outlook.

## II. GGDNM Model

The proposed model is named GGDNM, and before formally presenting it, some basics and principles of the relevant model components need to be elaborated.

### A. Graph Convolutional Network (GCN)

GCN is an extension of CNN in non-Euclidean space. It is a feature extraction algorithm for graph data and is usually calculated using convolution in graph structure. GCN aggregates the information of adjacent nodes to the target node through convolution operation, so that the model can learn and capture the complex relationship between nodes. In this way, GCN can effectively use the topological information contained in the graph structure to improve the performance of the model. The algorithm principle is as follows.

Graph $G = (V, E)$ consists of a set of nodes $V$ and a set of edges $E$. The input feature matrix $H$ is constructed from all the node features, and an adjacency matrix $A$ is used to represent the edges connecting the nodes. The physical meaning is that there is a connection between certain nodes that can be used to transfer information. $H$ and $A$ are used as the input of the GCN, and the output $Y$ is obtained through the processing function $f$ of the GCN:

$$f(H, A) = Y. \tag{1}$$

The convolutional feedforward method for each layer of GCN is:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{l}W^{l}\right), \tag{2}$$

where $H^l$ is the input of the $l$th layer network, $H^l \in \mathrm{R}^{n\times d}$; $n$ is the number of nodes in the graph $G = (V, E)$, each node is represented by a $d$-dimensional feature vector; $A$ is the adjacency matrix of the undirected graph, $\tilde{A} = A + I_N$, $I_N$ is the $N$-order identity matrix; $\tilde{D}$ is the degree matrix of the undirected graph, $\tilde{D} = \sum_j \tilde{A}_{ij}$; $W^l \in \mathrm{R}^{d\times h}$ is the parameters to be trained; $h$ is the output dimension; $\sigma$ is the corresponding activation function.

The structure of a multi-layer GCN is shown in Fig. 1. The nodes in the input graph data are processed in parallel, and each node can obtain information from adjacent nodes through convolution operations to complete a single information aggregation. Each layer of the GCN performs the same convolution operation,

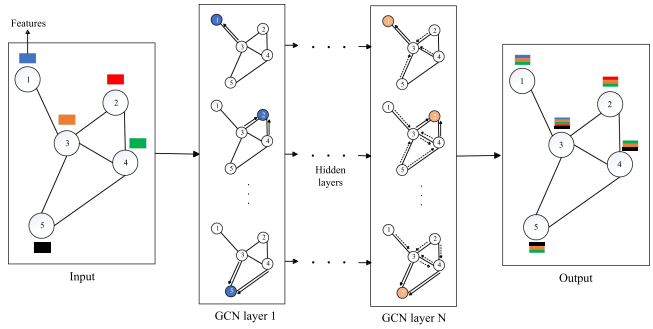


Fig. 1. The structure of GCN. Taking node 1 as an example, in the convolution operation of the first layer of the GCN, it obtains some feature information of node 3 (solid arrows indicate the transfer of information), and in the convolution operation of the Nth layer of the GCN, since node 3 has already aggregated the relevant information of the other nodes in the previous layers (dashed arrows), it will transfer this information together to node 1. Therefore, in the final output, the feature of each node data will contain the features of other nodes.

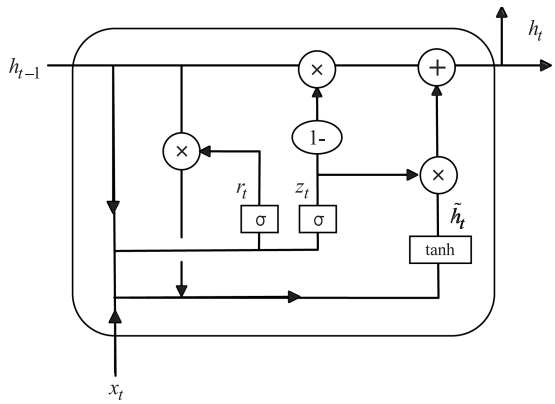


Fig. 2. The structure of GRU.

so that the original information in the nodes is passed further, and finally the output is obtained, each node will contain partial information about other nodes. The graph structure is exactly the same as the input, that is to say, the convolution operation of the GCN does not change the organization of the data.

### B. Gated Recurrent Unit (GRU)

GRU is a commonly used variant of RNN. Similar to LSTM, it controls the flow of information through a gating mechanism to capture long-term dependencies in time series. However, the GRU structure is simpler and has fewer parameters, so it is more computationally efficient. The structure of GRU is shown in Fig. 2. The core of GRU consists of two gates: the update gate and the reset gate.

The update gate controls how much of the hidden state from the previous time step should be retained in the current time step, and determines whether the current hidden state relies more on past information or on new input information:

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right), \tag{3}$$

where $z_t$ is the output of the update gate, $\sigma$ is the sigmoid activation function, $W_z$ is the weight matrix of the update gate,

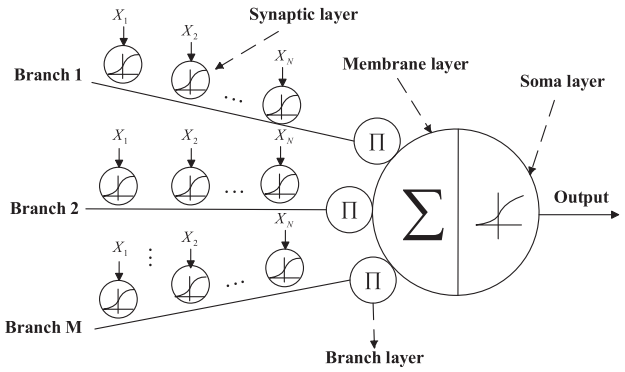


Fig. 3. The structure of DNM.

$h_{t-1}$ is the hidden state of the previous time step, and $x_t$ is the input of the current time step.

The reset gate determines how much of the hidden state of the previous time step should be forgotten, i.e. how the information from the previous time step is used when calculating the current candidate hidden state:

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right), \tag{4}$$

where $r_t$ is the output of the reset gate and $W_r$ is the weight matrix of the reset gate.

The candidate hidden state is calculated by the current input and the hidden state of the previous time step. By updating the output of the gate, the hidden state of the current time step is obtained by combining the candidate hidden state and the hidden state of the previous time step. The calculation formula is as follows:

$$\tilde{h}_t = \tanh\left(W \cdot [r_t \odot h_{t-1}, x_t]\right), \tag{5}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t, \tag{6}$$

where $\tilde{h}_t$ is the candidate hidden state, $W$ is the weight matrix, $h_t$ is the hidden state at the current time step, $h_{t-1}$ is the hidden state at the previous time step, and $z_t$ is the output of the update gate.

### C. Dendritic Neuron Model (DNM)

The non-linear mechanism of various synapses in the dendrites enables the synapses to play an important role in computing. The synaptic inputs from various neurons can be evenly distributed in the dendritic tree, and the stability and strong connection of the synapses, as well as the changes in neuronal excitability, also bring about strong plasticity. The process of imitating the way in which the dendrites in the biological brain process information is the working principle of DNM. DNM can be divided into a synaptic layer, a branch layer, a membrane layer and a soma layer, as is shown in Fig. 3.

$X_1, X_2, \cdots X_N$ represents a set of feature data input into the dendritic model, corresponding to the signal from the synapses of other cells. The synaptic layer contains two connection states: forward and backward connections. The sigmoid function is used in this research to represent different connection types. The node function formula from the $i$th input to the $j$th synaptic layer is:

$$Y_{ij} = \frac{1}{1 + e^{-k(w_{ij}x_i - \theta_{ij})}}, \tag{7}$$

where $Y_{ij}$ is the output, whose range becomes [0,1] under the influence of this function, $x_i$ is the $i$th input, $w_{ij}$ represents the parameters of the synapse, $k$ is a positive number, and $\theta_{ij}$ is the threshold for the activation of the synapse layer.

The role of the branch layer is to process and enhance the incoming signals from the synapse layer. It processes these signals by multiplication, increasing the DNM's ability to process complex data. The formula for the $j$th branch is:

$$Z_j = \prod_{i=1}^{N} Y_{ij}. \tag{8}$$

The function of the membrane layer is to accumulate signals from the branch layer. The input signals received from the branch layer are aggregated by the summation function. The output formula of this layer is as follows:

$$V = \sum_{j=1}^{M} Z_j. \tag{9}$$

The output of the membrane layer is transmitted to the soma layer, which is used to determine whether the neurons in the soma should be activated. If the output of this part exceeds a pre-set threshold, the soma produces a positive signal, otherwise it produces a negative signal:

$$O = \frac{1}{1 + e^{-k_s(V - \theta_s)}}, \tag{10}$$

where $k_s$ is a positive number, $V$ is the output of the the membrane layer, $\theta_s$ is the activation threshold of the soma and $O$ is the final output value of the model.

### D. GGDNM

The model architecture of the GGDNM proposed by this research is shown in Fig. 4.

GCNs are usually only used to process strictly traditional graph data, so researchers need to look for new data during the learning process, such as geographical location information and system topology, then artificially construct graph nodes and edges as additional feature data and send them to the neural network as input together with the original load data, ignoring the fact that the time series data itself can be regarded as a straightforward and logical graph structure data. This not only makes it difficult and more laborious to carry out the relevant load prediction task, but also puts aside the powerful feature extraction function of GCN for load data itself. Considering this, we manage to use GCN directly for feature extraction on the adjusted load data.

The original data is composed of multiple features including the load. GCN requires the existence of edges between the nodes of the input data to indicate that these nodes are related to each other, whereas the original sequence data do not have such explicit relationships with respect to each other, so this

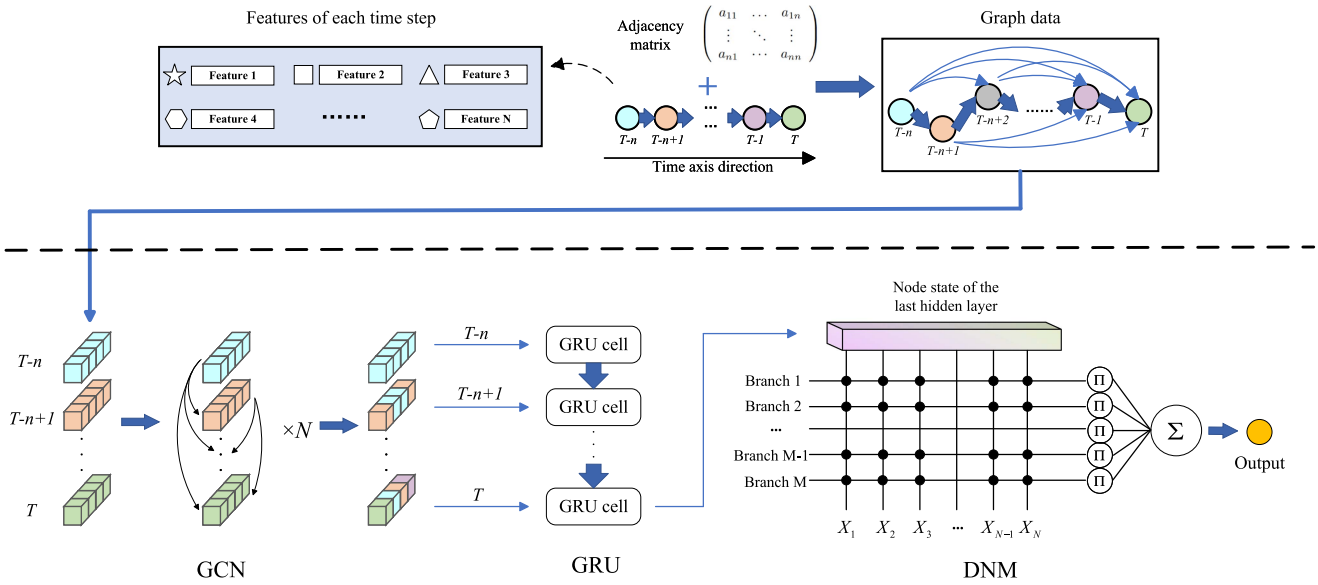


Fig. 4. The structure of the proposed GGDNM. Above the dashed line is the graph data construction module, which creates raw N-feature time series data into graph data represented by nodes and edges by integrating the load sequence with the corresponding adjacency matrices. Below the dashed line is the model learning module, the graph data is first fed into the GCN for information aggregation and dimension increasing operations, but the data structure of the time step is retained. Then the GRU captures the temporal relationships in the sequence data, and finally the DNM integrates the features to complete the prediction of the target value.

study describes the node-to-node topological relationships by giving an adjacency square matrix A, which is calculated by the following formula:

$$\mathrm{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \in R^{n\times n}, \tag{11}$$

$$a_{i,j} = \begin{cases} 1, \text{ if } i \le j \\ 0, \text{ if } i > j \end{cases}, \tag{12}$$

where $n$ is the length of the time series and $a_{ij}$ is an element in the adjacency matrix A. An element in the adjacency matrix with a value of 1 indicates that an edge exists between two nodes, and a value of 0 indicates the opposite. For example, if we decide to look back three time steps forward at the feature data, we need to additionally construct a 3*3 upper triangular matrix as the adjacency matrix for this set of training data. The 1's on the diagonal represent each time step connected to itself, while the 1's on the top represent the edges between different time steps. The data and the adjacency matrix are simultaneously sent to the two interfaces of the deep learning framework respectively, and its GCN function library automatically constructs them into graph data containing nodes and edges. Without additional location or topological information, the time series is constructed as graph data by this. It is worth mentioning that because of the irreversibility of the time series relationship, the edges between time nodes are one-way edges, that is, the adjacency information only aggregates from the previous time nodes to the later time nodes. Although the node represented by the last time step is connected to all other nodes, it only receives new information from other nodes and does not pass on its own feature information.

The constructed graph data is then sent to the GCN as input, $N$ represents the number of GCN blocks. These graph convolution blocks extract features from the data by directly passing information about the current node and indirectly passing information about the node of the preceding blocks further away, which makes the trends that exist between the data more obvious, without changing the original graph data structure. And in the last block, we increase the number of output channels of it to boost the dimensionality of feature data in each time step. The output of the GCN is fed into the GRU in steps, the feature data of the leaning forward time step is processed first by the GRU cell, and the obtained hidden state is passed to the next GRU cell, which then operates with the next time step features to obtain a new hidden state. With such iterative processing, the hidden node of the last time step aggregates the effective information of all previous time steps, by which the GRU successfully captures and passes on the long-term dependencies between the time series. The underlying layer of the model is DNM, which takes the data as the synaptic input of each branch. After the branch data is multiplied, the membrane layer adds it up and finally, through the nonlinear operation of the soma layer, the data is deeply integrated. The strong connectivity of the synapses, the stability and the strong plasticity brought about by the changes in neuronal excitability greatly enhance the model's ability to fit nonlinear features, integrate the data and output the final predicted regression value.

## III. Experiments and Analysis

### A. Preliminary

*1) Datasets:* Thanks to the worldwide expansion of the Smart Meter Infrastructure (SMI), many researchers have access to datasets with complete, high-quality and sufficiently large

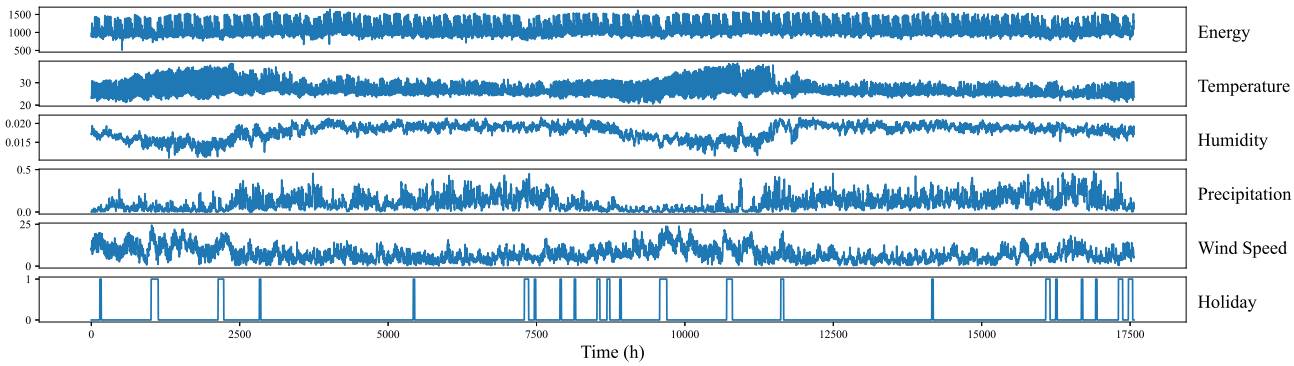


Fig. 5. Panama load dataset. The horizontal axis represents time with hourly resolution, and the vertical axis is the value of each feature. Each subplot represents a feature whose label is to the right of the subplot, refer to Table I for more details.

TABLE I
DESCRIPTION OF DATASET VARIABLES

| Variable | Description | Unit |
|---|---|---|
| Energy | National electricity load | MWh |
| Temperature | Temperature at 2 meters in Tocumen, Panama city | ℃ |
| Humidity | Relative humidity at 2 meters in Tocumen, Panama city | % |
| Precipitation | Liquid precipitation in Tocumen, Panama city | liters/$m^2$ |
| Wind Speed | Wind Speed at 2 meters in Tocumen, Panama city | m/s |
| Holiday | Holiday binary indicator | 1 = holiday, 0 = regular day |

electrical parameters, laying the foundation for the development of traditional power systems into smart grids. To demonstrate the validity of the proposed GGDNM learning model, the Panama national load dataset [32] and Tetouan regional load dataset [33] used in this topic are introduced below.

The Panama load dataset is a national aggregation level dataset, it records the energy demand and related multivariate variables of the country from January 3, 2015 to January 3, 2017, with a recording interval of one hour. Since Panama is located in the south of Central America, its strong seasonal changes have a significant impact on the characteristics of electricity load, making the dataset suitable as a case study for short-term load forecasting. The Panama load dataset contains 17,564 samples, each of which contains six features: energy (MWh), temperature (°C), humidity (%), precipitation (liters/$m^2$), wind speed (m/s), and whether the day is a holiday. These six features show the typical characteristics of load data, including obvious seasonality and trend, and have a strong correlation with daily human activities and environmental factors. They are divided into a training set and a test set in a ratio of 6:4, i.e. there are 10,538 training set samples and 7,016 test set samples. The data is visualized in Fig. 5, and the features are described in detail in Table I.

However, at lower levels of aggregation, such as regional and community, the variability of load data is much higher than at the national level, which means that it is much more difficult to do load forecasting at these levels. Therefore, a load dataset at regional aggregation level of Tetouan is also selected for this study to further illustrate the validity of the GGDNM as well to test its generalization ability. Tetouan is a city located in the north of Morocco, and this dataset records the energy demand(KWh), temperature(°C), humidity(%), and wind speed(m/s) for a slice of the city's main electricity-using region in 2017 at an hourly resolution, which contains 8736 samples. Data is organized similarly to the Panama load dataset and will not be repeated here.

*2) Data Pre-Processing:* (1) Removal of missing values and outliers. The purpose of removing missing values and outliers is to improve data quality and make the experimental data more accurate and reliable. First, the mean value of each column of features is calculated, and the outliers that are greater than three times the mean value are removed using a Boolean index filter. Then, all missing values are filled in with the column mean value, completing the removal of missing values and outliers. (2) Normalization. The purpose of normalization is to unify data of different features into the same scale range, so that the model can assign a more balanced weight to each feature, avoiding instability and convergence difficulties in the model training process caused by different feature scales, while improving the model's performance and generalization ability. This study uses maximum-minimum value normalization, and the calculation formula is as follows:

$$X = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{13}$$

where $X$ is the normalized value, $x$ is the original data, $x_{\max}$ and $x_{\min}$ respectively represent the maximum and minimum values in the original feature data column.

*3) Evaluation Metrics:* This research selects the mean absolute error ($MAE$), mean absolute percentage error ($MAPE$), root mean square error ($RMSE$) and coefficient of determination ($R^2$) as the evaluation criteria for model performance. Among them, $MAE$, $MAPE$ and $RMSE$ are evaluation criteria used to reflect the deviation between the predicted value and the true value. The closer the value is to 0, the smaller the deviation is. $R^2$ is a statistical measure used to evaluate the goodness of fit of a regression model, reflecting the degree to which the model explains the variance of the observed data. The maximum value is 1, indicating a perfect fit. The formulas for the above evaluation criteria are shown below:

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|y_t - y'_t|, \tag{14}$$

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{y_t - y'_t}{y'_t}\right| \times 100\%, \tag{15}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - y'_t)^2}, \tag{16}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_t - y'_t)^2}{\sum_{i=1}^{N}(\bar{y}_t - y_t)^2}, \tag{17}$$

where $N$ is the length of the time series, $y_t$ is the actual data, $y'_t$ is the resulting forecast series, and $\bar{y}_t$ is the average of the actual data.

## B. Benchmark Models

In order to validate the effectiveness of the proposed models, a series of benchmark models are selected in this study to carry out experiments on the same dataset, which are used to compare with the performance of GGDNM. These models are either very representative classical models in the history of deep learning or cutting-edge models that have gained much attention in recent years. They have all achieved very good results on time-series prediction tasks, including BPNN [34], [35], which is the cornerstone of deep learning and is applicable to a wide range of domains with strong generalization capabilities; LSTM [6], the most representative time-series processing model, which is able to capture long-term dependencies in sequences and allow the model to have a memory function; CNN_LSTM [36], using the convolutional layer to extract features, which strengthens the learning effect of LSTM and significantly improves the prediction accuracy; TCN [37], which uses dilated convolution and a large receptive field, has stronger long-range dependency processing capability than RNN, and has no loop structure and is more stable in gradient propagation; Transformer [38], whose core self-attention mechanism enables communication between global features, allowing the models to process sequences without being restricted to a fixed window or distance, and can effectively capture long-distance dependencies, thus greatly improving sequence modelling capabilities; and KAN [39], [40], a recently proposed fully-connected architecture that replaces the weights in each node with a parameterized univariate function, which is capable of approximating a complex function with fewer parameters when dealing with high-dimensional data, and thus has a higher computational efficiency.

TABLE II
THE STRUCTURE SETTINGS FOR EACH MODEL

| Method | Batchsize | Learning rate | Learning epoch | Structure |
|---|---|---|---|---|
| GGDNM | 32 | 0.001 | 100 | GCN1(out_channels=6)<br>GCN2(out_channels=32)<br>GRU(nodes=128)<br>DNM(M=10) |
| BPNN | 32 | 0.001 | 100 | FC1(nodes=128)<br>FC2(nodes=64)<br>FC3 |
| LSTM | 32 | 0.001 | 100 | LSTM(nodes=128)<br>FC1(nodes=64)<br>FC2 |
| CNN_LSTM | 32 | 0.001 | 100 | CNN(out_channels=64)<br>LSTM(nodes=128)<br>FC |
| TCN | 32 | 0.001 | 100 | TCN(nodes=128,<br>dilation_size=2)<br>FC |
| Transformer | 32 | 0.001 | 100 | PositionalEncoder<br>Encoder(nodes=128)<br>FC |
| KAN | 32 | 0.001 | 100 | KAN_FC1(nodes=128)<br>KAN_FC2(nodes=64)<br>KAN_FC3 |

## C. Implementation Details

In order to ensure the fairness of the experiment, the GGDNM model components proposed in this study and the benchmark models used for comparison refer to existing relevant literature and experiments, and adopt the most representative and competitive model architectures and parameters. On this basis, fine-tuning is carried out to enable all methods to achieve optimal performance on the same task. For example, the BPNN uses two hidden layers, plus a dropout layer to reduce overfitting, and finally an output layer to obtain the predicted value [41], [42]. Repeated experiments are conducted using the trial and error method [43], and the final number of hidden layer nodes is determined to be 128 and 64, respectively. The best test performance is obtained on the Panama load dataset. Considering that the structure of KAN is similar, the number of nodes selected is the same as that of BPNN; Similarly, the LSTM selects one hidden layer, and the optimal performance is achieved when the number of hidden nodes is 128. The same is true for the GRU structure. When CNN is used for time series feature extraction, the size of the convolution kernel covers all the features of a single time step, and the number of output channels is selected as 64. The number of hidden layer nodes of TCN is set to 128, and the expansion factor of the convolution kernel is set to 2. Since long sequence prediction is not involved, only the encoder of Transformer is used [44], and the number of hidden layer nodes is set to 128 after experimentation. The specific parameter settings are summarized in Table II.

The hyperparameters for training all models are kept consistent, i.e., the learning rate is 0.001, the sample batch size is 32, and the models look back at the time steps of the past 3 hours to predict the load value of the next hour. The number of training

TABLE III
COMPARISON OF RESULTS ON PANAMA DATASET

| Model | $MAE$ | $MAPE$ (%) | $RMSE$ | $R^2$ |
|---|---|---|---|---|
| BPNN | 36.576 | 3.14 | 49.272 | 0.932 |
| LSTM | 30.102 | 2.60 | 38.733 | 0.958 |
| CNN_LSTM | 26.456 | 2.33 | 35.192 | 0.965 |
| TCN | 26.499 | 2.31 | 36.155 | 0.964 |
| Transformer | 29.151 | 2.57 | 38.161 | 0.959 |
| KAN | 30.069 | 2.50 | 40.903 | 0.953 |
| GGDNM | **19.022** | **1.62** | **26.288** | **0.983** |

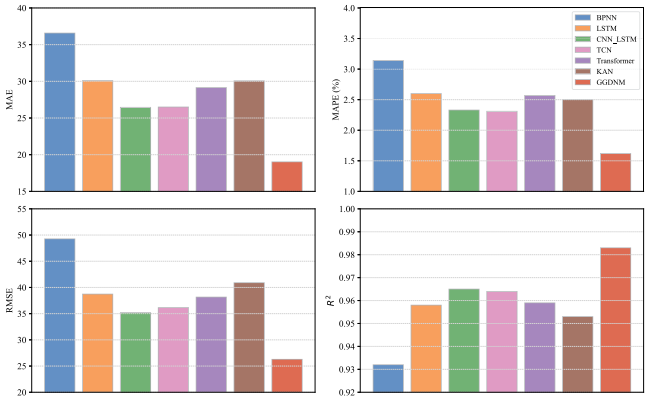


Fig. 6. Histograms of the performance of different models.

epochs is 100, and the $MAE$ loss function and Adam optimizer are selected. All model codes are written in the deep learning framework PyTorch 2.1.1+cu121 and python 3.8.18. The results of the experiments are the average of 10 independent runs. All experiments are run on the Windows 10 operating system, a 12th Gen Intel(R) Core(TM) i5-12490F 3.00 GHz CPU and an NVIDIA GeForce RTX 4060 GPU.

### D. Comparisons of Prediction Performance

The performance of different models on the Panama test dataset is shown in Table III. The GGDNM model proposed in this study achieves the best results in the four indicators of $MAE$, $MAPE$, $RMSE$ and $R^2$, reaching 19.022, 1.62%, 26.288 and 0.983 respectively. Compared with the most classic BPNN, GGDNM reduces its $MAE$ and $MAPE$ by nearly half. KAN's performance surpasses this traditional fully connection structure, but GGDNM still reduces its $MAE$ and $MAPE$ by 36.8% and 37.7% respectively. Compared to LSTM, which is widely used in time series tasks, GGDNM improves the $RMSE$ by 32.1%. The addition of the convolutional layer significantly enhances the ability of LSTM to capture temporal relationships, increasing the $R^2$ of the CNN_LSTM model to 0.965, and the cooperation of the graph convolutional layer, GRU and dendritic output cooperation improves it by a further 0.018. TCN, which is specifically designed to perform temporal convolutions on sequence data, achieves good results, with GGDNM reducing its $MAE$ and $MAPE$ by 28.2% and 29.9% respectively. The performance of the Transformer is not as good as expected, considering its number of parameters and computational complexity. In comparison, GGDNM has a 31.1% reduction in $RMSE$ and increases its $R^2$ by 0.024. The histograms of the performance of different models are shown in Fig. 6.

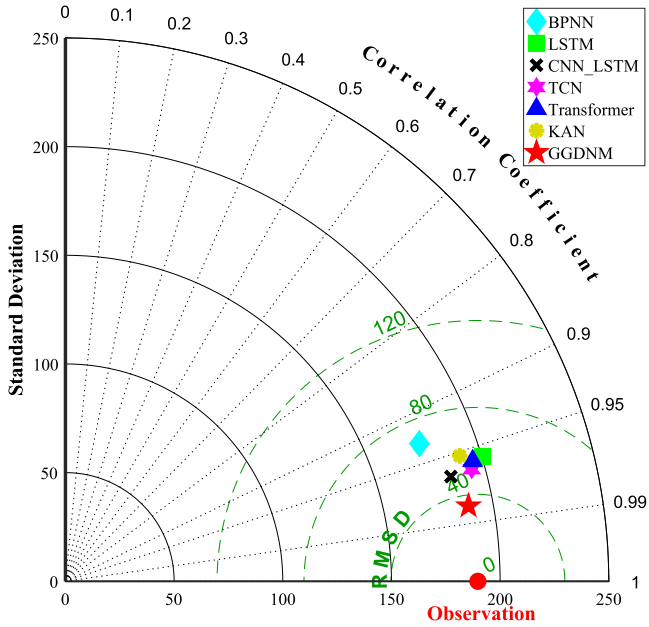


Fig. 7. Taylor diagram of different models.

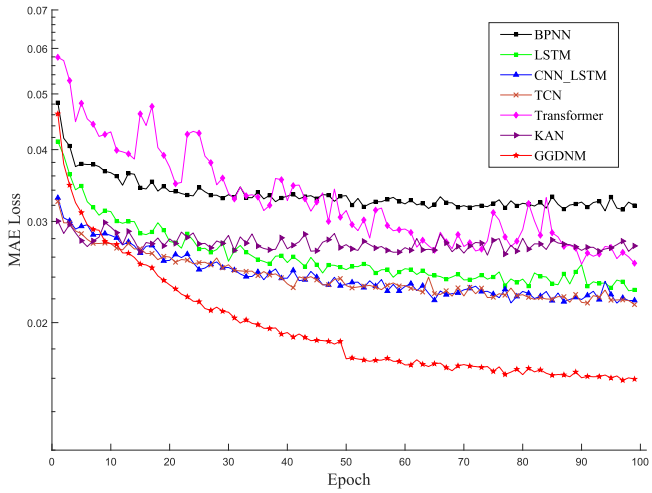


Fig. 8. Loss trace of different models.

Fig. 7 represents taylor diagram of different models. The horizontal and vertical coordinates indicate the standard deviation of the data, the radial line from the origin indicates the correlation coefficient, and the green concentric circles indicate the root mean square deviation. Compared with the other models, the scatter of GGDNM is obviously closest to the observed value point, the root mean square error of the predicted data is the smallest, the correlation coefficient is the highest, and the value of the standard deviation is the closest to the value of the standard deviation of the real data, which implies that the fitting effect of GGDNM to it is the best among these models.

The loss function trace of different models during training is given in Fig. 8. At the beginning of training, the GGDNM loss function decreases slightly more slowly than that of

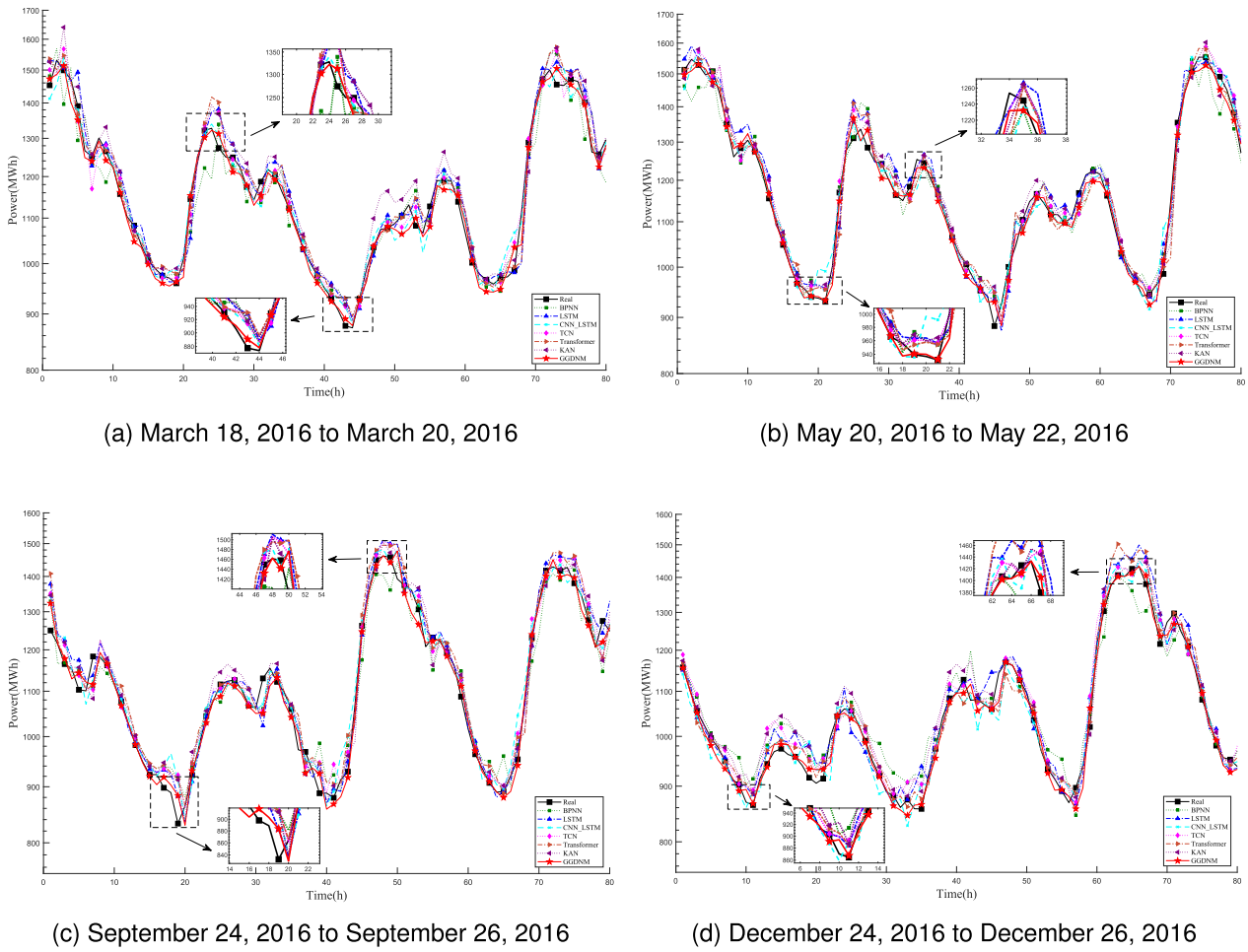


Fig. 9. Comparing load forecasting results of different models.

CNN_LSTM, TCN and KAN, is on a par with that of BPNN and LSTM, and is much faster than that of Transformer. However, as training progresses, when the epoch reaches around 50, the loss functions of the other benchmark models quickly converge to their minimum values, while the loss function of GGDNM continues to steadily decrease, indicating that the model is still learning new features from the data iteration and will not converge until near the end of training. This robustness is brought about by close co-operation between the hybrid model modules.

Fig. 9 shows the fitting curves of different models for the predicted load, with data from 12 days randomly selected in four different seasons. It can be seen that the GGDNM prediction curve can follow the trend of the real load curve very well, and can quickly adjust when the load fluctuates. In particular, when the real load data increases or decreases sharply, GGDNM can perceive this change very sensitively, accurately tracking the steep curve and reducing the lag of the prediction curve. As can be seen in the partially enlarged portion in the figure, when the load reaches peaks or troughs, the proposed model can still fit the curve better than other models. When the real load value reaches a peak and suddenly drops, most of the other models' prediction curves will continue to rise beyond this peak, while GGDNM can keenly sense this trend and fit the load curve down. When the real load value decreases to a certain level and suddenly starts to rise, GGDNM can still follow it closely. It must be admitted that other benchmark models can sometimes do this, but this is more of a coincidence and random phenomenon; and GGDNM sometimes fails to achieve superior performance at all peaks, but it is clear from the figure that it is up to the task in most cases, which is very important in real-world power resource decision-making and scheduling, and we try to explore and explain the origin of this superior performance in Section B of Chapter IV. In addition, from a holistic perspective, the GGDNM's prediction curve appears smoother and more stable, with few strange spikes and violent jitters, which are common and unreasonable in the curves of BPNN and KAN. This shows that GGDNM can excel at this multi-variable load prediction task, improving accuracy while also becoming more reliable.

### *E. Generalizability Test*

To better illustrate the validity of GGDNM and to verify its generalization ability on load dataset at different levels of aggregation, experiments are carried out on the Tetouan regional dataset using a similar experimental methodology and setup. Performance results are shown in Table IV.

TABLE IV
COMPARISON OF RESULTS ON TETOUAN DATASET

| Model | $MAE$ | $MAPE$ (%) | $RMSE$ | $R^2$ |
|---|---|---|---|---|
| BPNN | 1768.725 | 5.83 | 2422.206 | 0.863 |
| LSTM | 1468.635 | 4.90 | 2023.359 | 0.904 |
| CNN_LSTM | 1366.192 | 4.48 | 2021.510 | 0.907 |
| TCN | 1331.082 | 4.41 | 1976.193 | 0.910 |
| Transformer | 1380.321 | 4.66 | 1932.535 | 0.914 |
| KAN | 1450.478 | 4.73 | 2178.941 | 0.890 |
| GGDNM | **1220.427** | **3.98** | **1803.074** | **0.928** |

TABLE V
RESULTS OF ABLATION EXPERIMENT

| Model | $MAE$ | $MAPE$ (%) | $RMSE$ | $R^2$ |
|---|---|---|---|---|
| GGDNM | 19.022 | 1.62 | 26.288 | 0.983 |
| w/o GCN | 24.381 | 2.13 | 32.643 | 0.969 |
| w/o GRU | 26.850 | 2.38 | 38.121 | 0.960 |
| w/o DNM | 23.779 | 2.10 | 32.272 | 0.971 |
| GRU | 27.684 | 2.46 | 37.408 | 0.961 |

It is clear from the experimental results that due to the increased variability of data, all the models show a decrease in performance on the dataset at a lower load aggregation level, but GGDNM still achieves the best results of 1220.427, 3.98, 1803.074, 0.928 on $MAE$, $MAPE$, $RMSE$, $R^2$ respectively. Fully connected models with relatively single structure such as BPNN and KAN are obviously not able to fulfill the prediction task well, but the mapping ability brought by the non-fixed weights of KAN slightly improves its expressive ability. Unlike on Panama dataset, Transformer outperforms CNN_LSTM and TCN here, perhaps because of its intrinsic complex structure that better extracts features from the volatility of data. While GGDNM improves on its performance by about one-seventh, proving the effectiveness of its hybrid module with certain generalization.

## IV. DISCUSSION

### A. Ablation Experiment

Ablation experiments systematically remove or modify parts of the model to assess the importance of its components and understand their contribution to overall performance [45]. In order to verify the effectiveness of the proposed model components in the GGDNM, this study conductes an ablation experiment with it together with the w/o GCN (where the GCN layer is removed and the data is directly input into the GRU), w/o GRU (where the GRU layer is removed and the raw data is input into the GCN), w/o DNM (where the DNM is replaced with a traditional fully connected layer as the output layer), and GRU models. The results are shown in Table V.

As can be seen, the GGDNM still achieves the best predictive performance. After the GCN layer is removed, the $MAE$, $MAPE$ and $RMSE$ increase by 28.2%, 31.5% and 24.2% respectively due to the lack of feature extraction by graph convolution, which means that the error becomes larger. When the GRU is removed, the performance drops even more severely, with $R^2$ decreasing to 0.960, which highlights the importance of a temporal processing module in the entire model. The disappearance of the DNM layer weakens the model's nonlinear expression ability, causing the $MAE$ and $MAPE$ of the model to increase by 25.0% and 29.6% respectively. When only GRU is left in the model, its performance becomes almost similar to that of the LSTM model, which fully demonstrates the role and importance of graph convolution and dendritic learning in GGDNM. Fig. 10 shows the visualized form of the prediction results of several models in the ablation experiment.

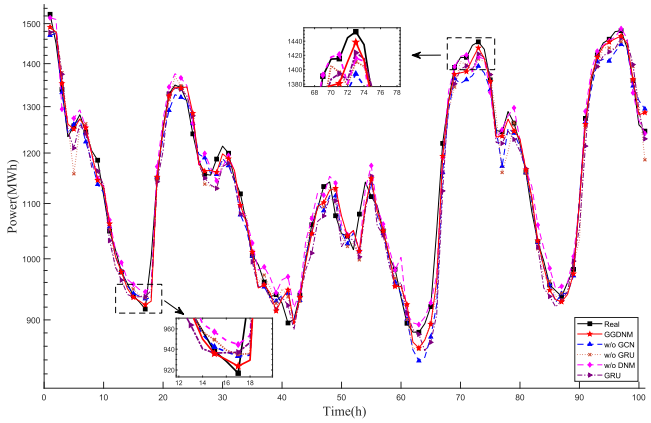


Fig. 10. Comparing ablation experiment results.

### B. Function of the Graph Convolution

To further explain the role played by GCN in the model, this study also explores the changes that occur during the internal data operation of the model. We want to know what the graph convolutional layer does with these low-dimensional representations when data is first fed into the model.

To accomplish this, a well-trained GGDNM and another GGDNM model that has only completed random initialization and remains untrained are first saved locally. Next, we manually select four training samples with typical trend characteristics from the Panama dataset and focus only on the target predictive feature, i.e., the load value. They are samples with an obvious upward trend, an obvious downward trend, peak and trough respectively, as shown in the black samples in Fig. 11. It is worth noting that they are pre-processed normalized values ranging from 0 to 1, and each sample contains three time steps. Then, the two saved models are loaded, and the four samples are used as input for feed-forward calculation respectively. The corresponding output features after the GCN1 layer (seen in Table II, where the number of input and output features remains the same) are obtained, and we still focus only on the load value and print it, as is shown in the blue and red samples in Fig. 11, representing the untrained and trained, respectively.

Comparing the three, the blue dashed lines look haphazard and irregular because the model has not been trained. When the original sample has an upward trend, the upward trend of the red line, i.e., the slope between the points, becomes larger

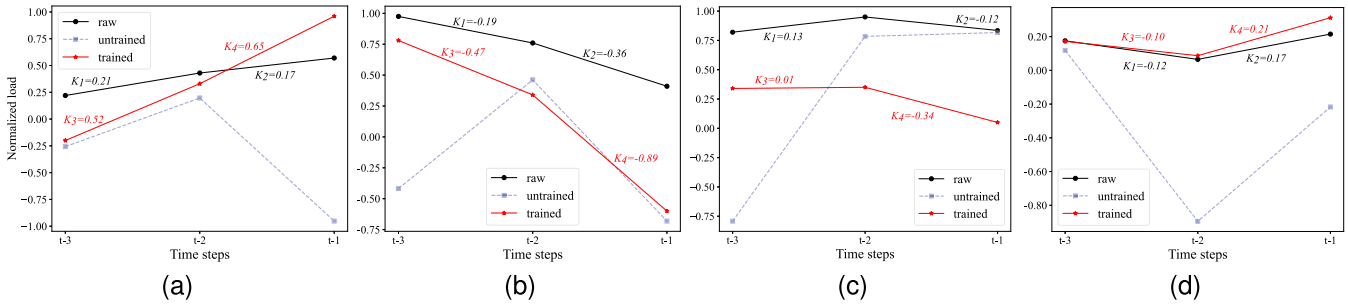


Fig. 11. Variations in the internal features of models after graph convolution. The horizontal axis represents the time steps and the vertical axis represents the normalized load values. The black lines represent the raw values, the blue dashed lines and the red lines indicate the load values processed by the untrained and trained GCN layer, respectively. $K$ is the slope between the corresponding two points, indicating an upward or downward trend. In all four typical feature trend data, the GCN learns and enhances the original trend.

than the black line; when the original sample reaches a peak, the GCN-processed red line smartly stops continuing to grow, mitigating the overshooting phenomenon. The same is true when the sample is falling or at the trough. It is clear that the GCN not only learns the trend and sudden changes contained in the features over time, but also they are even enhanced after the feature aggregation operation of the GCN. This enhancement effect becomes particularly obvious when the load features have an upward or downward trend, indicated by the before and after slopes. This may explain why the GGDNM can quickly respond and follow up the real load data.

The aggregation function of the graph convolutional layer is similar to the attention mechanism in Transformer, which allows features from different nodes to communicate and pass information. The unidirectional edges used in graph convolution to represent the logical relationship between time nodes can be used to correspond to the mask operation in the attention mechanism calculation process. They are all designed to prevent future information from being leaked and ensure that the temporal relationship is passed forward. However, Transformer's complex mechanism seems a bit redundant when used on relatively simple short time load data, whereas GCN is equally capable of accomplishing similar functions, and even does it better and faster.

However, after trying to print out the number of parameters for the encoder part of the Transformer containing the attention mechanism and the GCN part of the GGDNM respectively, we find that the number of parameters for the former is 33,472, while for the latter is only 656. It is clear that their purposes of manipulating the data are similar, and that the GCN accomplishes such a purpose, but the amount of arithmetic energy consumed and the time spent are quite different, proving the efficiency of GCN in GGDNM.

### C. Comparison of Fully Connected Layer, KAN and DNM

Experiments are also carried out on the effect of different output layers on the model performance, including the fully connected layer (FC), KAN and DNM. Keeping the structure and parameters of the feature extraction part of the model, i.e., the GCN and GRU, unchanged, the data is integrated using the fully connected layer, KAN and DNM as the output layers respectively, and the experimental performance results obtained are shown in Table VI, and their visualisations are shown in Fig. 12. It can be seen from the graphs that there is almost no difference in the model performance when FC and KAN are used as output layers, and KAN slightly reduces the RMSE of the fitted data. In addition, we also print out the total number of parameters for each of the three output layers, which are 129, 1280, and 203, respectively, indicating that the DNM accomplishes its task excellently with a relatively small number of parameters.

TABLE VI
COMPARISON OF DIFFERENT OUTPUT LAYERS

| Output Layer | $MAE$ | $MAPE$ (%) | $RMSE$ | $R^2$ | Parameters |
|---|---|---|---|---|---|
| FC | 23.779 | 2.10 | 32.272 | 0.971 | 129 |
| KAN | 23.864 | 2.10 | 30.898 | 0.973 | 1280 |
| DNM | 19.022 | 1.62 | 26.288 | 0.983 | 203 |

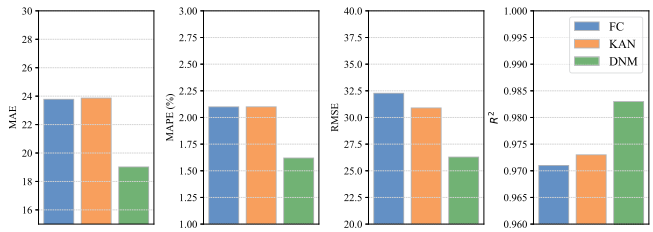


Fig. 12. Histograms of the performance of different output layers.

## V. CONCLUSION

This study proposes a hybrid deep learning model, GGDNM, for short-term load forecasting based on graph convolutional neural network, gated recurrent unit and dendritic learning network. First, the original load data are constructed into graph structured data based on time points as the input of the graph convolutional neural network. The changing trends in the data are enhanced by the graph convolution operation and feature extraction is completed. Then, the GRU computes the output of the GCN to capture the enhanced time-series dependencies. Finally, a traditional fully connected layer is replaced with a dendritic learning layer to use its powerful integration ability

to process non-linear data to output the final prediction results. GGDNM achieves a $MAE$ of 19.022 on the Panama load dataset at the national aggregation level, a $MAPE$ of 1.62%, a $RMSE$ of 26.288, and a $R^2$ of 0.983. On another regional aggregation level dataset, GGDNM also achieves improvement in these four evaluation metrics compared to the other benchmark models, reducing $MAPE$ to less than 4% and achieving close to 0.93 on $R^2$, showing that GGDNM stands out among a number of benchmark models and contributes to short-term load forecasting studies. In addition, ablation experiments are carried out in this study to observe the way in which data features change within components and to compare the effects of different output layers to further explain the effectiveness of GCN and DNM.

However, the GGDNM still has limitations. The combination of three different networks increases the computational complexity of the model, which reduces the efficiency of learning. In the future, we will explore more concise and efficient networks to simplify the computational speed of the model, and try to improve the GCN so that it can adjust its feature extraction process in advance of data trends in different regions.

## REFERENCES


[1] J. G. Jetcheva, M. Majidpour, and W.-P. Chen, "Neural network model ensembles for building-level electricity load forecasts," *Energy Buildings*, vol. 84, pp. 214–223, 2014.

[2] Y. Huang et al., "Multi-objective optimization of campus microgrid system considering electric vehicle charging load integrated to power grid," *Sustain. Cities Soc.*, vol. 98, 2023, Art. no. 104778.

[3] A. Jahani, K. Zare, and L. M. Khanli, "Short-term load forecasting for microgrid energy management system using hybrid SPM-LSTM," *Sustain. Cities Soc.*, vol. 98, 2023, Art. no. 104775.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, Jan. 2019.

[7] R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, and J. Wang, "Machine health monitoring using local feature-based gated recurrent unit networks," *IEEE Trans. Ind. Electron.*, vol. 65, no. 2, pp. 1539–1548, Feb. 2018.

[8] P. B. Weerakody, K. W. Wong, G. Wang, and W. Ela, "A review of irregular time series data handling with gated recurrent neural networks," *Neurocomputing*, vol. 441, pp. 161–178, 2021.

[9] K. Wang et al., "Multiple convolutional neural networks for multivariate time series prediction," *Neurocomputing*, vol. 360, pp. 107–119, 2019.

[10] H. J. Sadaei, P. C. D. L. e Silva, F. G. Guimaraes, and M. H. Lee, "Short-term load forecasting by using a combined method of convolutional neural networks and fuzzy time series," *Energy*, vol. 175, pp. 365–377, 2019.

[11] M. Imani, "Electrical load-temperature CNN for residential load forecasting," *Energy*, vol. 227, 2021, Art. no. 120480.

[12] X. Zhou et al., "Graph convolutional network hashing," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1460–1472, Apr. 2020.

[13] I. Ullah, M. Manzo, M. Shah, and M. G. Madden, "Graph convolutional networks: Analysis, improvements and results," *Appl. Intell.*, vol. 52, no. 8, pp. 9033–9044, 2022.

[14] Y. Hu, X. Cheng, S. Wang, J. Chen, T. Zhao, and E. Dai, "Times series forecasting for urban building energy consumption based on graph convolutional network," *Appl. Energy*, vol. 307, 2022, Art. no. 118231.

[15] H. Mansoor et al., "Graph convolutional networks based short-term load forecasting: Leveraging spatial information for improved accuracy," *Electric Power Syst. Res.*, vol. 230, 2024, Art. no. 110263.

[16] J. Song, L. Zhang, G. Xue, Y. Ma, S. Gao, and Q. Jiang, "Predicting hourly heating load in a district heating system based on a hybrid CNN-LSTM model," *Energy Buildings*, vol. 243, 2021, Art. no. 110998.

[17] W. H. Chung, Y. H. Gu, and S. J. Yoo, "District heater load forecasting based on machine learning and parallel CNN-LSTM attention," *Energy*, vol. 246, 2022, Art. no. 123350.

[18] C. Li, G. Li, K. Wang, and B. Han, "A multi-energy load forecasting method based on parallel architecture CNN-GRU and transfer learning for data deficient integrated energy systems," *Energy*, vol. 259, 2022, Art. no. 124967.

[19] D. Niu, M. Yu, L. Sun, T. Gao, and K. Wang, "Short-term multi-energy load forecasting for integrated energy systems based on CNN-BiGRU optimized by attention mechanism," *Appl. Energy*, vol. 313, 2022, Art. no. 118801.

[20] S. Arastehfar, M. Matinkia, and M. R. Jabbarpour, "Short-term residential load forecasting using graph convolutional recurrent neural networks," *Eng. Appl. Artif. Intell.*, vol. 116, 2022, Art. no. 105358.

[21] J. Ji, C. Tang, J. Zhao, Z. Tang, and Y. Todo, "A survey on dendritic neuron model: Mechanisms, algorithms and practical applications," *Neurocomputing*, vol. 489, pp. 390–406, 2022.

[22] W. Chen, J. Sun, S. Gao, J.-J. Cheng, J. Wang, and Y. Todo, "Using a single dendritic neuron to forecast tourist arrivals to Japan," *IEICE Trans. Inf. Syst.*, vol. 100, no. 1, pp. 190–202, 2017.

[23] T. Zhou, S. Gao, J. Wang, C. Chu, Y. Todo, and Z. Tang, "Financial time series prediction using a dendritic neuron model," *Knowl.-Based Syst.*, vol. 105, pp. 214–224, 2016.

[24] A. Gidon et al., "Dendritic action potentials and computation in human layer 2/3 cortical neurons," *Science*, vol. 367, no. 6473, pp. 83–87, 2020.

[25] J. Ji, S. Gao, J. Cheng, Z. Tang, and Y. Todo, "An approximate logic neuron model with a dendritic structure," *Neurocomputing*, vol. 173, pp. 1775–1783, 2016.

[26] T. Zhang, C. Lv, F. Ma, K. Zhao, H. Wang, and G. M. O'Hare, "A photovoltaic power forecasting model based on dendritic neuron networks with the aid of wavelet transform," *Neurocomputing*, vol. 397, pp. 438–446, 2020.

[27] J. Ji, M. Dong, Q. Lin, and K. C. Tan, "Forecasting wind speed time series via dendritic neural regression," *IEEE Comput. Intell. Mag.*, vol. 16, no. 3, pp. 50–66, Aug. 2021.

[28] K. Song et al., "Short-term load forecasting based on CEEMDAN and dendritic deep learning," *Knowl.-Based Syst.*, vol. 294, 2024, Art. no. 111729.

[29] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.

[30] Y. Yu, Z. Lei, Y. Wang, T. Zhang, C. Peng, and S. Gao, "Improving dendritic neuron model with dynamic scale-free network-based differential evolution," *IEEE/CAA J. Automatica Sinica*, vol. 9, no. 1, pp. 99–110, Jan. 2022.

[31] E. Bas, E. Egrioglu, and T. Cansu, "Robust training of median dendritic artificial neural networks for time series forecasting," *Expert Syst. Appl.*, vol. 238, 2024, Art. no. 122080.

[32] A. Madrid, "Short-term electricity load forecasting (Panama case study)," Mendeley Data, vol. 1, 2021. [Online]. Available: https://data.mendeley.com/datasets/byx7sztj59/1

[33] Fedesoriano, "Electric power consumption," Aug. 2022. [Online]. Available: https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption

[34] S.-T. Chen, D. C. Yu, and A. R. Moghaddamjo, "Weather sensitive short-term load forecasting using nonfully connected artificial neural network," *IEEE Trans. Power Syst.*, vol. 7, no. 3, pp. 1098–1105, Aug. 1992.

[35] D. Srinivasan, A. Liew, and C. Chang, "A neural network short-term load forecaster," *Electric Power Syst. Res.*, vol. 28, no. 3, pp. 227–234, 1994.

[36] T.-Y. Kim and S.-B. Cho, "Predicting residential energy consumption using CNN-LSTM neural networks," *Energy*, vol. 182, pp. 72–81, 2019.

[37] H. Wang, Y. Zhao, and S. Tan, "Short-term load forecasting of power system based on time convolutional network," in *Proc. 8th Int. Symp. Next Gener. Electron.*, 2019, pp. 1–3.

[38] C. Wang, Y. Wang, Z. Ding, T. Zheng, J. Hu, and K. Zhang, "A transformer-based method of multienergy load forecasting in integrated energy system," *IEEE Trans. Smart Grid*, vol. 13, no. 4, pp. 2703–2714, Jul. 2022.

[39] M. H. Sulaiman, Z. Mustaffa, M. S. Saealal, M. M. Saari, and A. Z. Ahmad, "Utilizing the Kolmogorov-Arnold Networks for chiller energy consumption prediction in commercial building," *J. Building Eng.*, vol. 96, 2024, Art. no. 110475.

[40] T. Quanwei, X. Guijun, and X. Wenju, "MGMI: A novel deep learning model based on short-term thermal load prediction," *Appl. Energy*, vol. 376, 2024, Art. no. 124209.

[41] S. Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture," *Int. J. Eng. Trends Technol.*, vol. 3, no. 6, pp. 714–717, 2012.
[42] S. S. Reddy and J. A. Momoh, "Short term electrical load forecasting using back propagation neural networks," in *Proc. 2014 North Amer. Power Symp. (NAPS)*, IEEE, 2014, pp. 1–6.
[43] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.
[44] Q. Yan, Z. Lu, H. Liu, X. He, X. Zhang, and J. Guo, "An improved feature-time Transformer encoder-Bi-LSTM for short-term forecasting of user-level integrated energy loads," *Energy Buildings*, vol. 297, 2023, Art. no. 113396.
[45] X. Sun, P. Wu, and S. C. Hoi, "Face detection using deep learning: An improved faster RCNN approach," *Neurocomputing*, vol. 299, pp. 42–50, 2018.

**Chunyang Zhang** received the B.S. degree in 2024 from the Nanjing University of Posts and Telecommunications, Nanjing, China, where he is currently working toward the M.S. degree. His research interests include load forecasting and cross-regional load demand transfer learning and focuses on the application of artificial intelligence techniques in power systems.

**Yang Yu** received the M.S. and Ph.D. degrees from University of Toyama, Toyama, Japan, in 2017 and 2020, respectively. He is currently a Lecturer within the College of Automation & College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing, China. His research interests mainly include evolutionary computing, optimization problems, and artificial neural networks.

**Tengfei Zhang** (Senior Member, IEEE) received the B.S. degree from Henan University, Kaifeng, China, in 2002, and the M.S. and Ph.D. degrees from Shanghai Maritime University, Shanghai, China, in 2004 and 2007, respectively. From 2014 to 2015, he was a visiting Researcher with University College Dublin, Dublin, Ireland. He is currently a Professor within the College of Automation & College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing, China. He has authored or co-authored more than 100 publications and more than 30 inventions. His research interests include intelligent information processing, micro-grid modeling and control.

**Keyu Song** received the B.S. degree from Anhui Normal University, Wuhu, China, in 2022. He is currently working toward the M.S. degree with the College of Automation & College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing, China. His research interests include photovoltaic power forecasting, multi-objective optimization, and feature selection.

**Yirui Wang** received the Ph.D. degree from the Faculty of Engineering, University of Toyama, Toyama, Japan, in 2020. He is currently an Associate Professor with the Faculty of Electrical Engineering and Computer Science, Ningbo University, Zhejiang, China. His research interests include computational intelligence, swarm intelligent algorithms, combinatorial optimizations, and computer vision.

**Shangce Gao** (Senior Member, IEEE) received the Ph.D. degree in innovative life science from the University of Toyama, Toyama, Japan, in 2011. He is currently a Professor with the Faculty of Engineering, University of Toyama, Japan. His research interests include nature-inspired technologies, machine learning, and neural networks for real-world applications. He is also an Associate Editor for many international journals such as IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, and IEEE/CAA JOURNAL OF AUTOMATICA SINICA.