CS412 Final Report: Power Load Prediction Model

Junway Lin* and Ibrahim Mammadov*
3220300390 and 3220300385
junway.22@intl.zju.edu.cn, ibrahim.22@intl.zju.edu.cn
Spring 2025
*Equal contribution

***Abstract*—This report presents a comprehensive exploration of power load forecasting using a variety of machine learning and deep learning models. We evaluated statistical methods (ARIMA, Holt–Winters, Prophet), ensemble models (XGBoost, LightGBM, Random Forest), and deep learning architectures (RNN, LSTM) across a full year of 15-minute interval power usage data. Data preprocessing techniques were carefully designed to address missing data, anomalies, and seasonal variations. Among all models tested, LSTM with engineered lag features delivered the most accurate predictions, significantly outperforming baseline approaches. The results emphasize the importance of robust data preparation, thoughtful feature design, and the selection of architectures suited to temporal modeling. Finally, we propose future directions including transformer models, model ensembling, and external data integration to further enhance forecasting performance.**

I. Introduction

A. Background

Load data prediction is crucial for energy management and optimization. In this project, the goal is to design and train a prediction model using load data from a household in Jianshan Town, Haining City, covering the period from January 1, 2022, to December 1, 2024. During the project, we will train a predictive model using historical data and evaluate its accuracy in forecasting future load (PV, Battery) patterns. The final load prediction model is expected to predict daily load patterns based on historical data (predicting the next day's load data based on the previous day) and then have its performance evaluated.

B. Problem Introduction

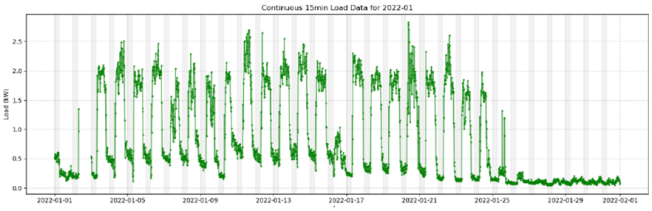

Figure 1: Sample Graph of Monthly Plot Showing Missing Values and Anomalies

In this project, we were provided with three datasets: train data, validation data, and check data which contained the load data from a household in Jianshan Town, Haining City. The training dataset had values which ranged from January 1, 2022, to December 30, 2023. The validation dataset ranged from January 1, 2024, to May 31, 2024 and the checking dataset ranged from June 1, 2024 to December 1, 2024. Each dataset was structured to hold 96 entries of load data per day, where each of the 96 entries represented 15-minute intervals. Upon initial observation of the training data set, we noticed that there were missing values, abnormal spikes and drops in load, and negative values which needed to be handled. Hence, the first step would be to perform data preprocessing to handle the cases of missing values and determining how to handle the detected anomalies, normalizing the data, and extracting features. Then, appropriate models will be explored and examined on their accuracy.

II. Data Preprocessing

A. Importance of Data Processing

Data preprocessing is a crucial step in model training because raw data often contains noise, missing values, and inconsistencies that can negatively impact the performance of machine learning models [1]. By cleaning, transforming, and normalizing the data, preprocessing ensures that the models receive high-quality, relevant input, leading to more accurate and reliable predictions. It also helps in reducing computational complexity and improving training efficiency by eliminating redundant or irrelevant features.

B. Missing Values

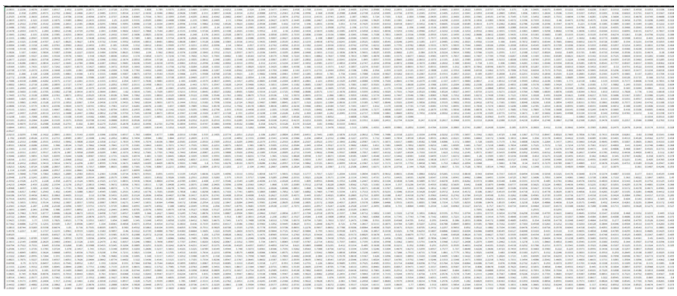

Figure 2: Sample Snapshot of Missing Data in Training Dataset

Handling missing values is a critical preprocessing step in time series forecasting, especially when working with real-world energy consumption data. Several methods can be employed depending on the extent and pattern of missingness. Forward fill (propagating the last known value) and backward fill (using the next known value) are simple and effective for

time series where temporal continuity matters. For more statistical robustness, interpolation techniques such as linear, spline, or polynomial interpolation can estimate missing values based on surrounding data points. In cases of random missingness, mean or median imputation may be used, though these can oversimplify the signal. More advanced methods include K-nearest neighbors (KNN) imputation, which fills in values based on similarity between data windows, or regression imputation, which uses correlated features to predict missing values.

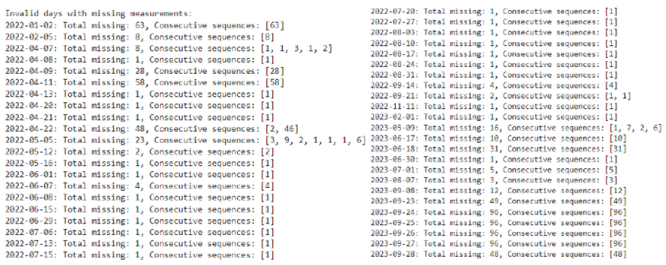

Figure 3: List of Missing Values

To decide what method(s) are ideal, we first identified the locations and sequences of missing data entries. The goal was to find a balance of avoiding creating too much artificial data and preserving as much data as possible. Hence, we settled on a combination of methods to handle missing data, where the removal of an entire day would be used if there are instances where more than 38 out of 96 data entries are missing. By allowing up to 60% of data to remain, we retain many partially complete days instead of discarding them entirely which helps to preserve the temporal structure and reduces unnecessary data loss. Additionally, days with 40% or more missing data are likely too incomplete for reliable imputation, and keeping them may introduce noise or bias during training. Hence, this threshold aims to strike a compromise between keeping enough data for robust model training and ensuring that the retained data are still informative and not overly sparse.

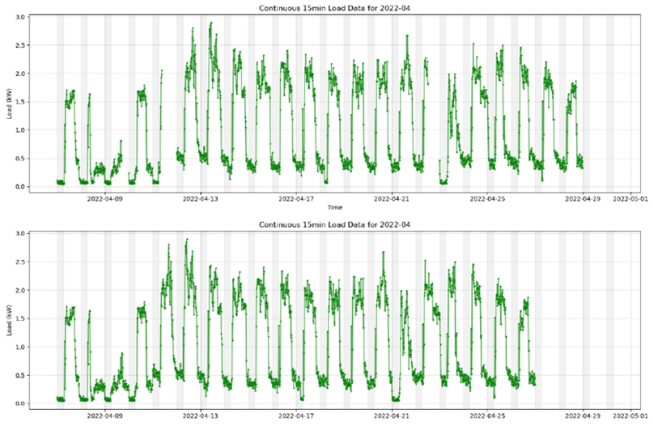

Figure 4: Sample Monthly Comparison of Original VS Cleaned Dataset

After the initial pass to remove dates that contained far too many missing data entries, a second pass was run on the intermediate dataset. In this stage, the main challenge was determining the appropriate impute method to handle the missing values and how we can use the information of the consecutive sequences to choose the best handling method. Initially, we settled on using a simple time-series linear impute method to fill in the missing data. However, there may be improvements to be made by exploring other methods such as KNN or model-based imputation, which would also allow us to have a much higher missing-data tolerance.

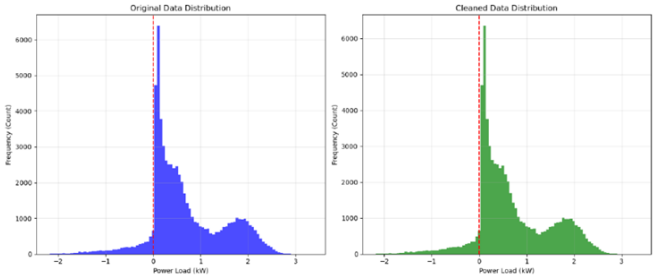

Figure 5: Comparison of Data Distribution Between Original and Cleaned Dataset

C. Abnormal Values

A dataset is highly unlikely to follow a set pattern, especially in real-world applications like power consumption, where unexpected events, holidays, or sensor errors can introduce anomalies. To address sudden spikes or drops in power load, we examined the temporal context and identified that many of these fluctuations coincided with national holidays or special occasions when usage patterns diverge sharply from typical behavior. To account for this, we sourced a list of Chinese holidays and introduced an is_Holiday feature, which would allow for the model to be able to flag these days and predict values accordingly. As a result, the model can recognize and adapt to these context-specific deviations, improving its ability to forecast similar anomalies in the future.

Negative values represent battery load instead of the usual PV load. Hence, the usual imputation strategies such as zero-filling, interpolation, local averaging, and implementing as a feature would not be effective as doing so severely hindered the accuracy of the model. The reason for this effect seems to stem from the fact that the model will try to average the value with other load data and results in a model that is unable to predict peak load. After extensive testing, we found that the most effective solution was to flag these values as anomalies rather than include them as features or impute them. By treating these anomalies as structural exceptions rather than learning targets, we preserve data integrity and improve the robustness and generalization of the model.

Furthermore, we also explored using an intermediate model which would detect and predict days which are anomalies. However, after trying multiple MAD multipliers, the model would always be sensitive to any changes to peak load. As a result, this method proved ineffective for our goals and was discarded.

```
MSTL periods = [96, 672]
Residual median = -0.008033, MAD = 0.150026
Threshold = 6.0×MAD = 0.900157
Flagged 2745 / 66720 points (4.11%)
Anomaly Detection Results (MAD multiplier = 6.0)
```

timestamp	Date	PowerSlot	Value	Slot	residual	threshold
2022-01-23 16:15:00	2022-01-23	Power66	1.7246	66	0.949420	0.900157
2022-01-23 16:30:00	2022-01-23	Power67	1.6300	67	0.973627	0.900157
2022-01-23 16:45:00	2022-01-23	Power68	1.5860	68	0.900559	0.900157
2022-01-23 17:00:00	2022-01-23	Power69	1.5920	69	0.992467	0.900157
2022-01-23 17:30:00	2022-01-23	Power71	1.5654	71	0.934624	0.900157
...	...	...	...	...	...	...
2023-12-29 15:45:00	2023-12-29	Power64	2.2231	64	1.025799	0.900157
2023-12-29 16:00:00	2023-12-29	Power65	2.1716	65	0.932768	0.900157
2023-12-29 16:15:00	2023-12-29	Power66	2.1927	66	0.979591	0.900157
2023-12-29 17:30:00	2023-12-29	Power71	2.1885	71	0.927217	0.900157
2023-12-29 17:45:00	2023-12-29	Power72	2.2361	72	0.904498	0.900157

2745 rows × 6 columns

Figure 6: Detected Anomalies Using a MAD Multiplier of 6.0

III. Model Selection

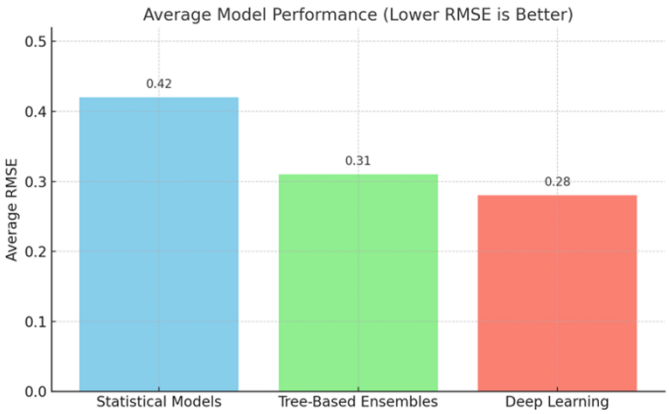

Figure 7: Average Expected RMSE Amongst the Different Classes of Models

A. Statistical Models

Several classes of models were systematically evaluated to determine the most suitable model for forecasting daily power consumption. Our evaluation began with traditional statistical models such as ARIMA, Holt–Winters, and Prophet. These models offer high interpretability and are well suited to capture trends and seasonality in time series data [2]. However, they are limited in their ability to incorporate external features (e.g. holidays, weather), and they often struggle with handling multiple or irregular seasonal patterns, making them less adaptable for complex real-world scenarios. As a result of being the poorest in performance, these methods were immediately ruled out.

B. Tree-Based Ensembles

We then explored tree-based ensembles which include models such as Random Forest, LightGBM, and XGBoost. These models demonstrated strong performance due to their ability to model non-linear relationships and integrate a variety of feature types, such as lagged power values, time-of-day indicators, and binary holiday flags. Additionally, they are relatively robust to missing data and outliers. However, they require careful hyperparameter tuning, and their decision-making process is less transparent compared to linear models, which can pose challenges for interpret-ability. Hence, with average reported accuracy that rivals deep learning model as well its profile fitting the parameters of the project, this class of models was selected for testing and evaluation.

C. Deep Learning Models

Finally, we assessed deep learning architectures such as RNNs (Recurrent Neural Networks), GRUs (Gated Recurrent Unit), LSTM (Long Short-Term Memory) networks, and Transformers. These models are highly effective at capturing long-term dependencies and complex temporal patterns, making them powerful tools for sequential modeling tasks [3]. However, their performance comes at the cost of requiring substantial amounts of training data, computational resources, and expertise in model architecture and tuning [4]. This class of models is able to pick up on the accuracy of the load values where tree-based ensembles may fall short. Hence, deep learning models were also selected to be tested and evaluated.

IV. Performance Evaluation

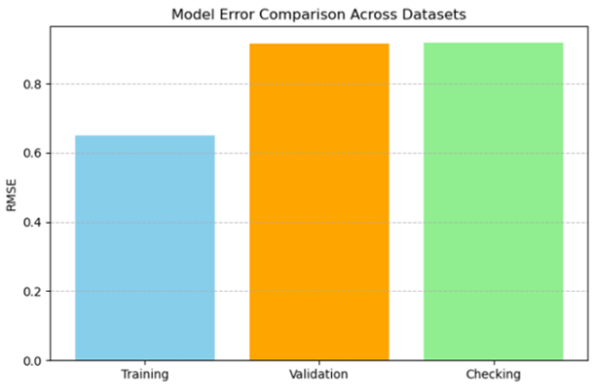

Figure 8: RMSE of XGBoost on Different Datasets

A. XGBoost Performance

We initially selected XGBoost as our primary model due to its strong reputation for handling structured data and its ability to model complex, nonlinear relationships. However, the results were ultimately suboptimal, as reflected in the high RMSE values across all datasets: Training RMSE = 0.649, Validation RMSE = 0.916, and Checking RMSE = 0.919.

A notable issue we observed was that model performance on the validation set began to degrade as training progressed,

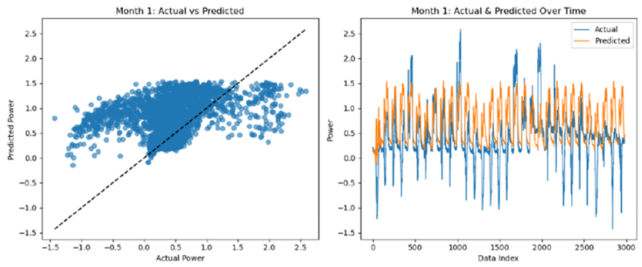

Figure 9: Scatterplot and Overlay Line Plot to Evaluate Model Performance VS Validation Data

suggesting overfitting. Specifically, after approximately 25 boosting rounds, the model started fitting the training data too closely, capturing noise and dataset-specific quirks that failed to generalize to unseen data. This overfitting was likely exacerbated by several factors. First, the feature set used for training was relatively limited, restricting the model's ability to learn meaningful and generalizable patterns. Without sufficient input signals, such as weather, time-based granular features, or domain-specific variables, XGBoost's complexity may have caused it to model spurious correlations rather than robust trends. Second, potential issues in normalization or feature scaling may have introduced inconsistencies, especially if scaling was not performed consistently across the training, validation, and testing sets. Such inconsistencies could distort feature relationships and reduce model reliability. Although XGBoost is a powerful tool, its effectiveness is heavily dependent on well-engineered features and careful regularization. In our case, the lack of experience resulted in a combination of limited feature representation, possible preprocessing issues, and insufficient regularization which largely contributed to its poor generalization performance.

B. RNN Performance

Due to the unforeseen poor performance of XGBoost, we believe it to be best to explore deep learning models since they are expected to perform better in peak load accuracy. However, this comes at the cost of having a high computation time and significantly more use in resources. Hence, model training was switched from CPU to CUDA GPU-training.

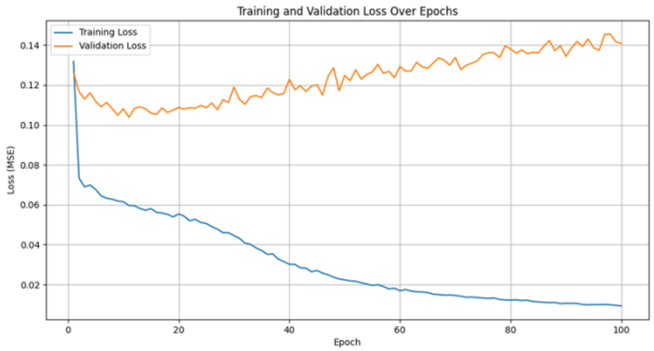

Figure 10: Training and Validation Loss of RNN

In our evaluation of Recurrent Neural Networks, the model demonstrated notably poor performance, with RMSE = 0.9266, MSE = 0.8585, MAE = 0.6558, and an exceptionally high MAPE of 115.29%. These metrics indicate that the RNN struggled to produce accurate predictions, particularly in capturing the scale and variability of the actual power usage. During training, we observed that after around 20 epochs, the training loss continued to decrease rapidly while the validation loss began to rise, a classic sign of overfitting. This suggests that the model began memorizing patterns from the training data without learning generalizable trends, likely due to the RNN's inability to effectively manage long-term dependencies or seasonal patterns within the sequence data. The RNN's architecture, which lacks dedicated mechanisms for remembering distant time steps (unlike LSTM or GRU), likely contributed to its poor performance in modeling complex, temporally extended behaviors in our dataset. Furthermore, the high MAPE indicates the model performed particularly poorly on days with lower absolute consumption, where even small prediction errors can lead to disproportionately high percentage errors. In addition, limited feature engineering and potential inconsistencies in input scaling may have made it difficult for the RNN to distinguish between meaningful patterns and noise. Overall, the basic RNN architecture proved insufficient for this task, highlighting the need for more advanced models that are better suited to capturing long-range dependencies and dynamic temporal structures in power consumption data.

C. GRU Performance

Gated Recurrent Units (GRUs) were also considered as a potential model due to their simplified structure and computational efficiency compared to Long Short-Term Memory (LSTM) networks. GRUs use fewer parameters by combining the forget and input gates into a single update gate, making them faster to train and less prone to overfitting on smaller datasets [4]. However, in practice, this architectural simplicity often comes at the cost of a reduced capacity to model long-term dependencies in complex sequences. In our experiments and supported by findings in the literature, GRUs generally underperform compared to LSTMs on tasks that involve intricate temporal relationships or require modeling subtle seasonal trends, both of which are critical in power load forecasting. Given that LSTMs offer better control over information flow through separate memory and gating mechanisms, they tend to capture more nuanced patterns over time. As a result, we chose to focus our efforts on LSTM-based models and other more advanced architectures, skipping GRUs to prioritize depth and accuracy over training speed and model simplicity.

D. LSTM Performance

The LSTM model showed a substantial improvement over previous architectures in our power load forecasting task. The model was built as a bidirectional LSTM with sequence_length = 5, batch_size = 16, number_epochs = 150, learning_rate = 0.001, LSTM_hidden_size = 128, number_layers = 2, and

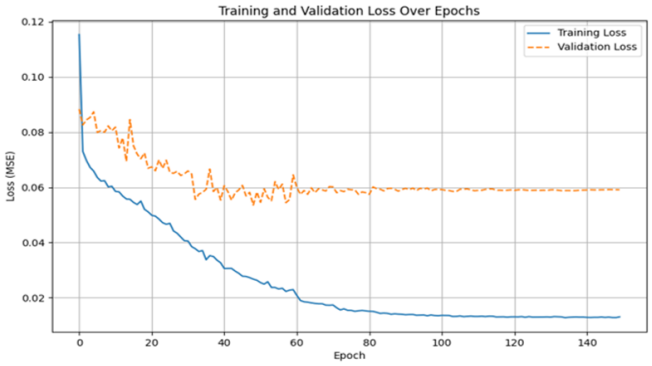

Figure 11: Training and Validation Loss of LSTM with Lag Features

dropout = 0.25. Most notably, the sequence length was selected to be 5 since a length of 7 would cause the model to generalize the whole week, while a length of 3 causes the model to be heavily affected by anomalies, both of which contributes to an overall poorer performance. Our baseline LSTM, without additional feature engineering, achieved RMSE = 0.7343, MSE = 0.5961, MAE = 0.4794, and MAPE = 96.25% on the checking dataset.

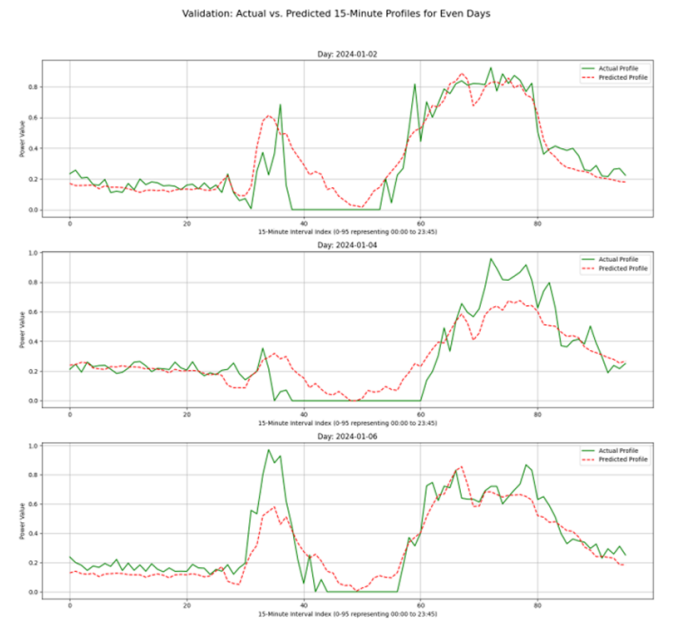

Figure 12: Sample Graphs of Model Prediction

These results showed that LSTM was much more capable of capturing temporal patterns and general consumption trends compared to the simpler RNNs or statistical models. However, it still struggled with specific challenges, particularly sudden seasonal shifts, anomalous days (like holidays), and peak load values. The base model tended to smooth out the predictions, underestimating the sharp variations that often characterize real-world load behavior.

To address these shortcomings, we introduced lag features, previous time steps as additional inputs, which provided the

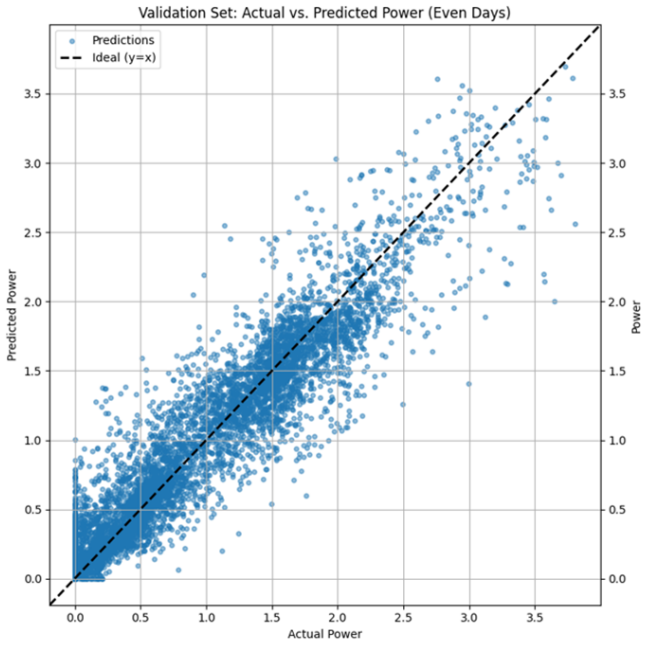

Figure 13: Scatterplot of Actual VS Predicted Load

model with more historical context and temporal cues. This enhancement significantly boosted performance when predicting on an unseen dataset, lowering RMSE to 0.475, MSE to 0.3591, MAE to 0.311, and MAPE to 48.72%. By allowing the LSTM to anchor its predictions to past observations more explicitly, the model became better at anticipating peaks and recognizing short-term shifts, which ultimately led to much more accurate and reliable forecasts.

V. Future Improvements

Looking ahead, there are several promising directions to enhance the accuracy and robustness of our forecasting pipeline. First, exploring transformer-based architectures, such as Temporal Fusion Transformers or Informer, could provide a major leap in performance due to their ability to model long-range dependencies and attend to important time steps, outperforming traditional recurrent structures in many sequence modeling tasks [5]. Additionally, revisiting the data preprocessing stage may offer valuable improvements; experimenting with advanced imputation techniques (e.g., KNN imputation, interpolation with seasonal decomposition, or denoising autoencoders) could help recover useful information from incomplete days rather than discarding them. Another crucial step would be to go back to earlier models like XGBoost or RNN and refine their hyperparameters, regularization, and feature sets, since their current results fall significantly short of the average performance across model classes. Model ensembling is another strategy worth pursuing, combining predictions from statistical, tree-based, and neural models could leverage the strengths of each and improve overall reliability. Finally, integrating external context like weather forecasts through

NVIDIA's FourCastNet could add an additional predictive signal, especially for accurately estimating peak load magnitudes, which are often influenced by temperature and atmospheric conditions [6]. These combined improvements could push our forecasting system toward state-of-the-art accuracy and resilience.

VI. Conclusion

We developed and tested multiple forecasting models for short-term power load prediction using a real-world, fine-grained dataset. Our findings showed that deep learning models, particularly LSTM enhanced with lag features, provided the best predictive performance. However, challenges remained in capturing anomalies, holiday effects, and sharp seasonal shifts. The performance of traditional and tree-based models, while interpretable and efficient, was generally lower than that of the LSTM-based approach. We conclude that further improvements can be achieved by exploring transformer architectures, refining preprocessing and imputation strategies, tuning underperforming models, leveraging ensemble methods, and incorporating external predictors like weather forecasts through tools such as NVIDIA FourCastNet. We believe these steps would help build a more accurate and adaptable forecasting system suitable for real-world deployment.

References

[1] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1–2, pp. 1–39, 2010.

[2] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

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

[4] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[6] K. Pathak, K. Saurabh, Z. Li, R. Yu, and A. Anandkumar, "Fourcastnet: Global medium-range weather forecasting with graph neural networks," *arXiv preprint arXiv:2202.11214*, 2022.