EAEE-E4000 - Final Project:
Electricity Real-Time Locational Marginal Price Prediction

Saud Alghumayjan (saa2244), Noah Hartzfeld (nah2178)
Supervision: Prof. Pierre Gentine

January 25, 2025

1 Introduction

Electricity price prediction is one of the hot topics in the literature in recent years. However, according to [1], the research domain doesn't seem to be fully explored yet. Broadly, price forecasting is categorized into three main types: short-term, mid-term, and long-term. These correspond to predicting prices hours ahead, days to months ahead, and years ahead, respectively. The choice of forecasting horizon depends on the specific application. In our study, we concentrate on the short-term horizon, given its predominant impact on day-ahead and real-time market operations.

The structure of U.S. electricity markets is intricately designed, operating through a two-stage settlement framework: the day-ahead market (DAM) and the real-time market (RTM). In the first stage, the DAM sets the price for the 24 trading intervals of the following day. Subsequently, the RTM steps in to dynamically balance supply and demand, addressing deviations from the earlier day-ahead schedules in real-time.

Integration of renewable energy sources and energy storage systems has made real-time prices less predictable and more volatile compared to day-ahead prices. These prices are determined by various factors, such as: supply and demand, weather conditions, and market behavior. Predicting the real-time prices in wholesale electricity markets is crucial for maximizing social welfare and determining optimal trading strategies in the electricity market.

In this project, we proposed a machine learning-based solution to predict electricity real-time prices with a focus on the short-term horizon of 24 hours ahead, as most studies focus on shorter horizons. We collected load zone price data from the New York Independent System Operator (NYISO) market. We have experimented with three state-of-the-art models, Convolutional Neural Networks (CNN), Long Short-Time Memory (LSTM), and CNN-LSTM, considering variable look-back windows. We trained each model thoroughly to find the best structure and set of hyperparameters. We conducted an extensive evaluation of the best candidate models and analyzed hourly errors. Finally, we applied the idea of transfer learning to move a trained model for a certain zone and use it in another. The project repository can be found in GitHub https://github.com/Alghumayjan/EAEE_E4000_Final_Project_RTP_Prediction.

The remainder of the report is organized as follows: Section II describes the dataset. Section III introduces the methodology. Section IV presents the results. Section V concludes the report.

2 Dataset Overview

In this study, we use a thorough and thoughtfully assembled dataset to address our research question. Employing a machine learning approach, the dataset is fundamental to our project,

forming the empirical basis for our analyses and findings. This section aims to offer a detailed overview of the dataset.

2.1 Dataset Description

The dataset collected for this project spans five years from 2017-2021. It is composed of day-ahead price, real-time price, and load forecast data for each load zone in the NYISO market. Figure 1 shows a map of NYISO's load zones. All datasets can be found on the NYISO market website [2].

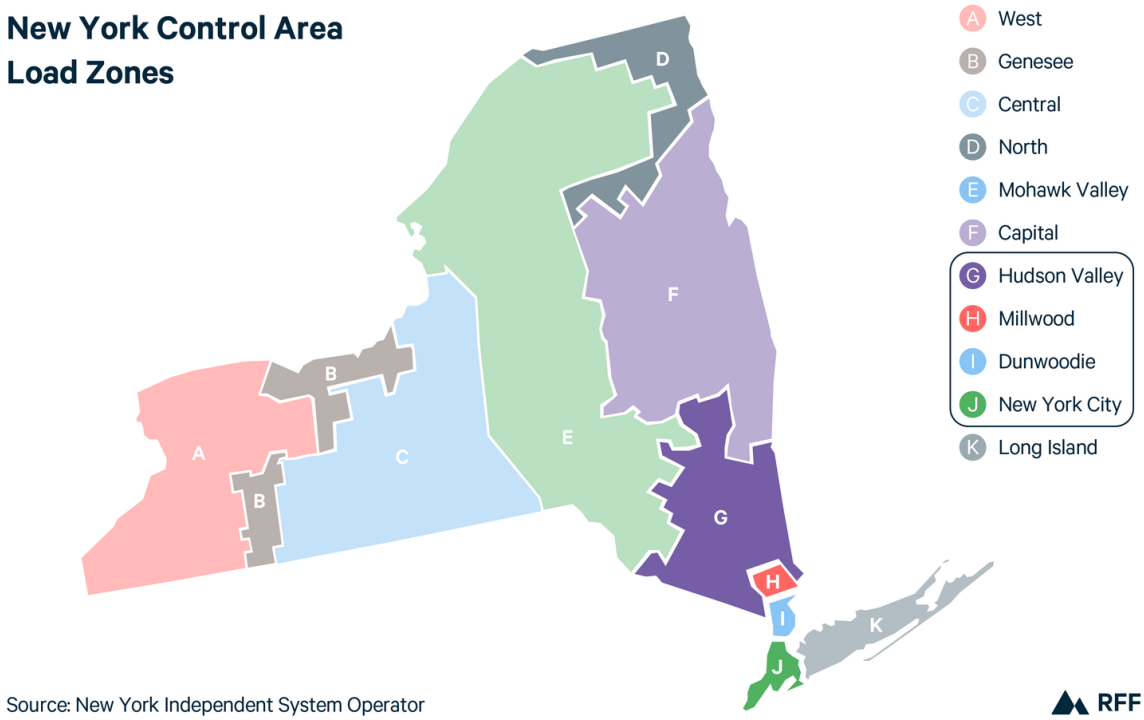

Figure 1: NYISO market load zones distribution in New York State.

- **Day-Ahead Price (DAP)**: This data is the day-ahead price for electricity in the wholesale market. It is the marginal price of electricity, which is the price of producing an additional 1 megawatt-hours (MWh) of electricity in dollars, so its unit is ($/MWh). It is also called the Locational Marginal Price (LMP), where the price is defined for each node in the system. The DAP is the aggregate price of all the nodes within each zone. This price is set by the system operator, which is in our case NYISO, based on the supply and demand. It gets published for the 24 trading intervals of the following day. The resolution of this data is hourly, which is the smallest available. This data was chosen based on its influence on the real-time price, our focus in this study. Figure 2 shows the price profile for this data for New York City (NYC) zone.

- **Real-Time Price (RTP)**: This data is the real-time price for electricity in the wholesale market. Similar to DAP, it is the marginal price of electricity and has the same units. The only difference is that this price represents the price of the electricity during the operation day, and it is set at each time step, hence the name. The resolution of this price in real-time is five minutes, but the collected data is hourly which is the average of the five-minute data. This price is set by the system operator and is influenced highly

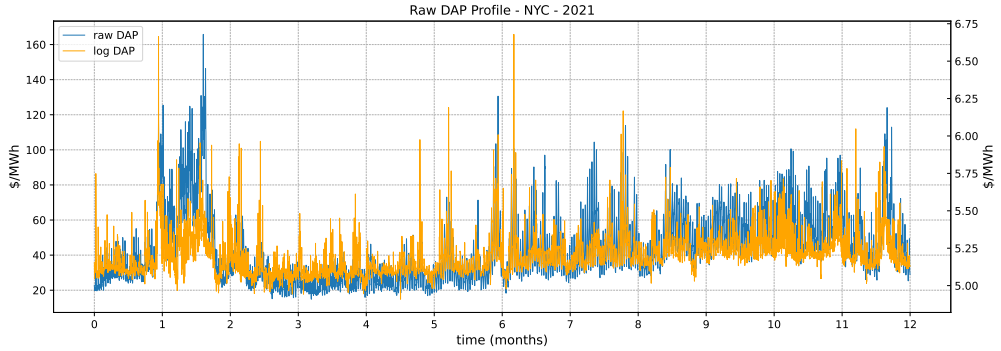

Figure 2: The price profile for the day-ahead price for NYC zone during 2021.

by the mismatch of supply and demand. Figure 3 shows the price profile for this data for the NYC zone.

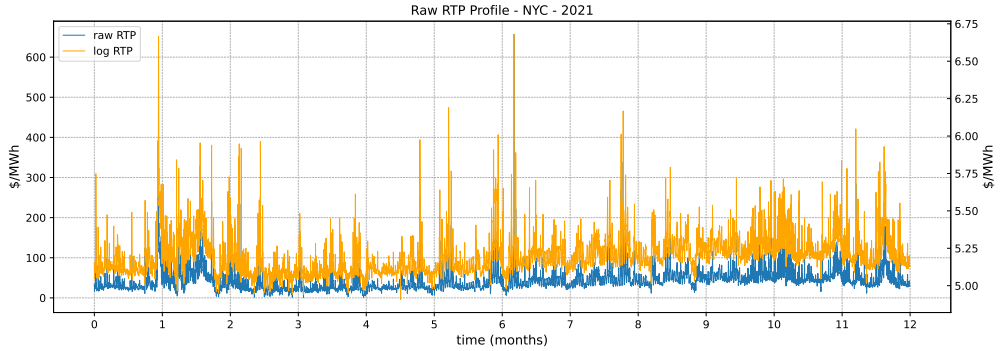

Figure 3: The price profile for the real-time price for NYC zone during 2021.

- **Load Forecast (LF)**: This data is the electricity load (demand) forecast made by the system operator. This data has a great influence on the RTP, as the load forecast can cause the mismatch between supply and demand. Hence, we included it in our project instead of the actual load. The unit of this data is megawatt-hour (MWh). The resolution is hourly. Figure 4 shows the price profile for this data for the NYC zone.

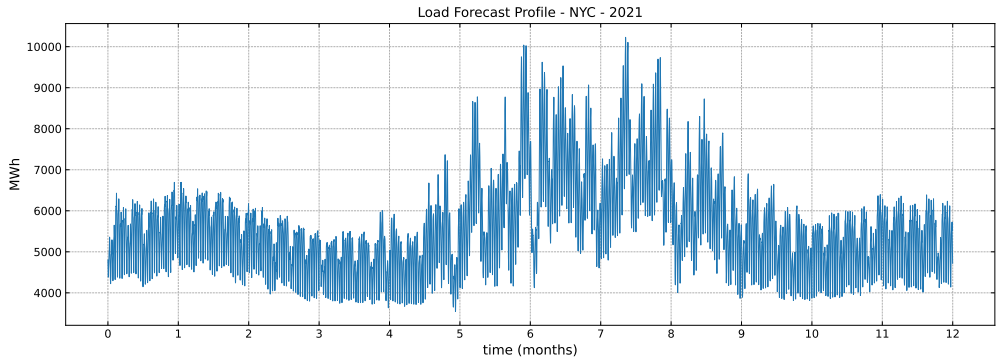

Figure 4: The load forecast profile for NYC zone during 2021.

2.2 Data Preprocessing

Collecting data from the main source sometimes needs data cleaning and preparation before feeding it into the model for training. So, we clean and merge the collected hourly data from

daily profiles collected from [2] to produce one file that includes all the data for each dataset. Also, as our price data can have random spikes, it can make the task much harder, so we do log transformation to lower the variance between the data. Figure 2-3 shows the log version of the prices alongside the raw version. It is clear that the log version has much lower spikes, but still hard to notice the difference in terms of the pattern or the trend. Figure 5 shows the distribution of the raw and log prices, we can see that the log version has a much better distribution. Moreover, figure 6 shows that the log version has a better shape, but is still not normally distributed, which can tell us that the prediction task will not be an easy one.

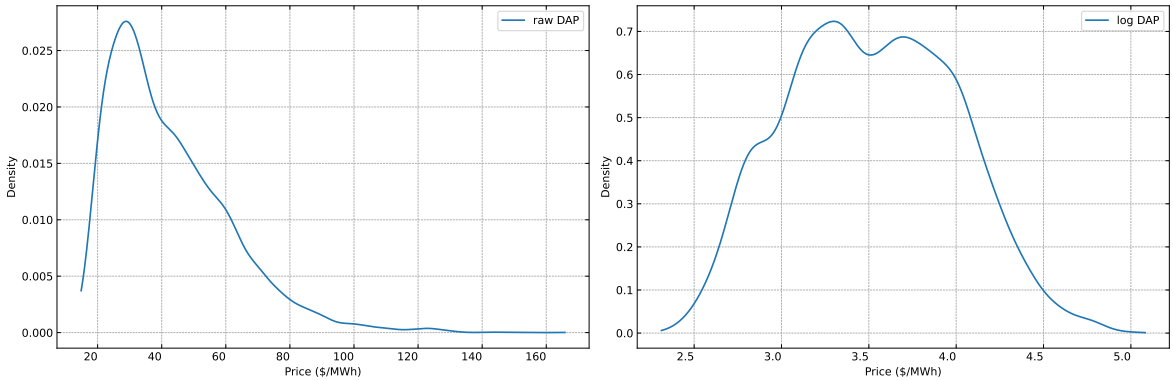

Figure 5: Density distribution of DAP for raw and log prices for 2021.

Table 1 summarizes statistics about DAP and RTP for all the zones. The price mean (μ) and standard deviation (σ) can help us understand and better judge the performance of the models.

Table 1: Zonal Price Data Statistics.

Zone	CAPITL	CENTRL	DUNWOD	GENESE	HUD VL	LONGIL	MHK VL	MILLWD	N.Y.C.	NORTH	WEST
DAP μ	44.60	29.90	41.65	28.79	40.74	54.06	30.19	41.63	42.52	22.68	31.03
DAP σ	18.93	16.30	18.11	15.92	17.27	31.37	16.68	18.21	18.92	16.05	19.44
RTP μ	44.16	29.57	41.44	28.55	40.09	54.78	30.08	41.75	42.46	23.53	30.89
RTP σ	29.60	22.69	25.96	22.54	23.34	63.64	23.39	29.62	26.86	30.23	27.05

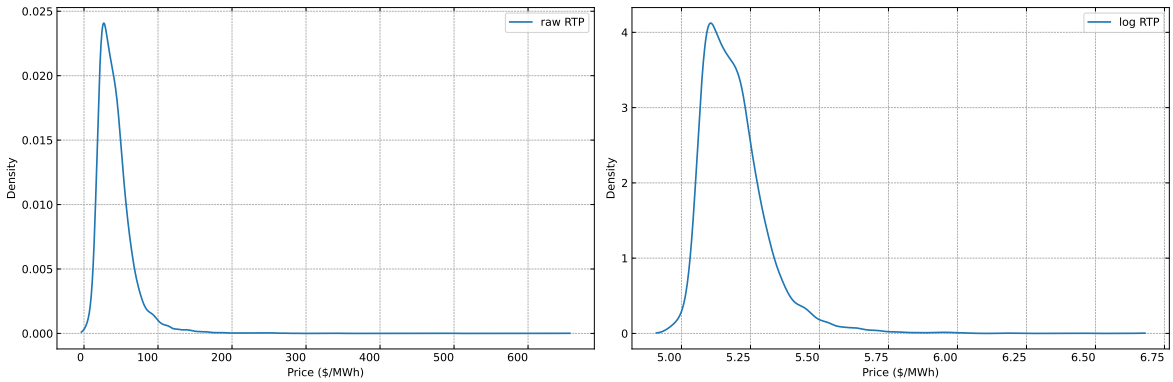

Figure 6: Density distribution of RTP for raw and log prices for 2021.

3 Methodology

In this section, we will explain in detail the workflow of this project to achieve the goal of this project, which is to build a machine learning-based solution to effectively forecast the real-time price of electricity in the wholesale market.

Figure 8 provides an overview of the project pipeline. We start from the collected data which serves as features to our forecasting model. These features are taken as time-series features with variable look-back windows. Then we conduct data preprocessing and log-transform the price data as explained in the aforementioned section. Then we use standardization to transform the data to zero mean and 1 standard deviation (0,1) distribution (The standard normal distribution), which helps in enhancing machine learning models' performance. Afterward, we build one of the state-of-the-art forecasting models that will be explained in the following subsections. Then, we rank the best models conduct further evaluations, and finally make the predictions.

Moreover, we added the Transfer Learning (TL) ability to our framework. When there is a new zone with limited data, we pass it to the best model, retrain it, and make predictions, such a process has shown better performance than training a new model from scratch on these limited data.

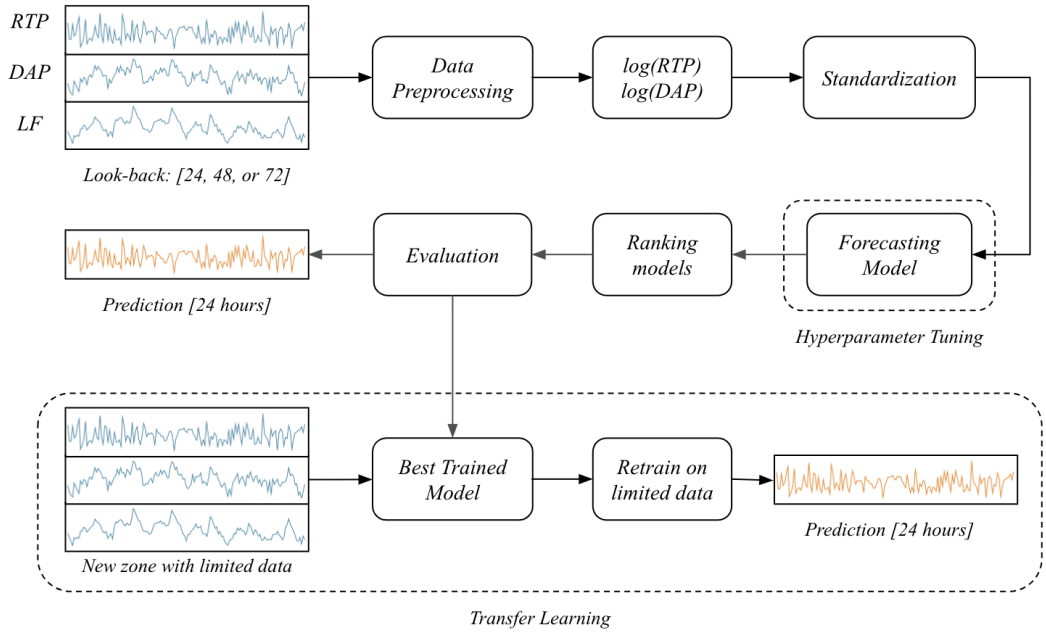

Figure 7: Project full pipeline.

3.1 CNN

Convolutional Neural Networks (CNN) are designed to process and analyze structured grid-like data and are well-suited for tasks involving images, videos, and other special data. They are especially good at learning spatial patterns and hierarchies of features across data. It is easy for a fully connected neural network to quickly grow to an unmanageable size, and CNN solves this problem by analyzing chunks of data and downscaling rather than continuously analyzing every data point. Furthermore, converting a 2D data structure like an image into a neural network loses the spatial relationships and information that was once in the data. By applying different filters to the data and downscaling CNN also tackles this problem.

To keep parameters low, CNN starts by considering the input data in patches, rather than by individual data points. The first step in the CNN model is to apply a convolution to each chunk of the data and assign a value. The value of each patch is determined by the max, sum, etc. whatever function is important for the task. These chunks of data are then connected to just a single neuron in the next layer with this value. This is downscaling, shrinking the dimensions of each next layer, without this there would be no dimensional reduction and no advantage to convolutions.

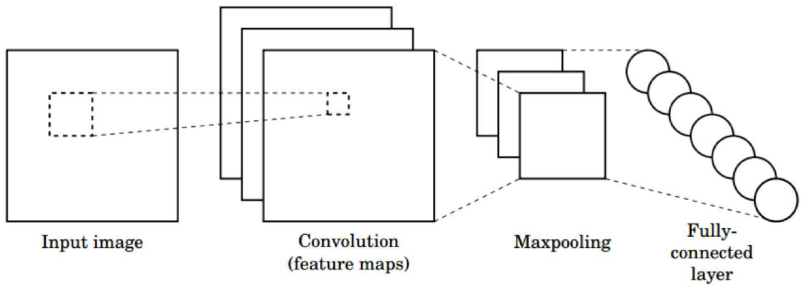

Figure 8: General CNN Structure.

It is this downscaling that helps CNN learn spatial patterns so well. The single value captures the most important values for an entire patch of the data, therefore capturing the spatial features of this block with one value. Multiple filters are applied to each section of the data, at each layer, searching for different patterns and features. The exact same parameters are then passed onto the next patch to be filtered identically. By maintaining the same weights for each filter across the data it helps to keep the number of individual parameters low. Since each filter is only looking for a specific feature in the data, weights do not need to be adjusted as the same features should be identified regardless of where it is located in the data. This same process of applying convolutions, downsizing, and connecting to the next level is repeated at each level. The early larger layers search for lower-level features in the data, and as they are downsized and combined each ensuing layer grows smaller and is filtered for higher-level features.

Since our data is a time series CNN may seem a less obvious candidate model for our project, however, CNN's ability to identify and leverage spatial locality to find patterns would be great to utilize in our data. By using 1D convolutional layers we can analyze and apply filters across our time series data. Similar to how patterns are found in 2D convolutions, 1D convolutions can extract temporal patterns in the data. Because CNNs identify low-level features in their lower levels they are excellent for capturing local patterns within the time series, regardless of their position in the data. Furthermore, because of how downscaling keeps only the most important values from each level, it is great at ignoring noise and variations, which would be particularly useful with our data set. Similarly, because weights are kept the same for individual filters, CNNs are not prone to fluctuations, which is another great attribute to leverage on our data. The CNN model should help us find key temporal features in our time series.

3.2 LSTM

Long Short-Term Memory (LSTM) [3], is a Recurrent Neural Network (RNN) model, designed to handle long-term dependencies from which standard RNNs struggle with. As is implied by its name, LSTM aims to get the best of both worlds by finding both long and short-term patterns. To do this, the LSTM model introduces several architectural improvements making it better suited for learning long-term dependencies in sequential data. The LSTM

model uses a more sophisticated memory block structure consisting of three gates: an input, output, and forget gate. The combined operations of these gates allow the current state to selectively suppress the previous state and update the next, giving much more control over the information output at each time step.

A standard RNN structure uses normal backpropagation, to update the weights of previous and current time steps. However, the structure is such that each time step is an identical Neural Network, so during backpropagation, the same computations of the same weights are repeated many times. This repetition is what causes standard RNNs to suffer from vanishing and exploding gradients. As gradients are multiplied in each layer during backpropagation, if the weights are close to zero, then the weights of the preceding layer receive incredibly small updates; this is the opposite for exploding gradients. LSTM solves this by attempting to minimize the number of computations that occur during backpropagation, by determining which values are important and which values are not. The main innovation that LSTM are series of memory blocks in which information is passed to, then through gates within, that compute and determine the usefulness of the data.

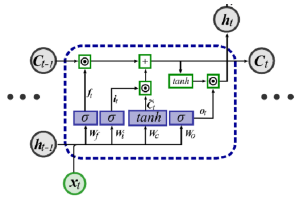

Figure 9: Memory Block Structure

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 10: Gate Logic

A single memory block has three distinct gates: the forget gate, the input gate, and the output gate. The leftmost gate relating to the first equation is the forget gate, which determines the relevancy of the information from the previous cell. Using output from the previous state, the current input, and current weights the forget gate reweighs the data from the previous time step. A sigmoid function is used for this calculation, as it keeps the new weights between 0 (forget the previous state entirely) and 1 (use the entirety of the previous state). The next two equations perform a similar task to determine the weight of the input data. The weights calculated in both the input and forget gates are then used in the fourth equation to determine the information to be carried forward in time and propagated to the next state. The last gate is the output gate, which once again performs a reweighting, but this time uses the last two equations to weigh the data that is propagated forward to determine what to pass to the output of this time step. Memory blocks are a powerful tool that helps identify the most useful data to influence the learning process. By suppressing information that is less useful, or has already been used, it helps LSTM avoid repeated identical calculations and makes it a prime model for evaluating long-term dependencies.

As our data is a time series the clear candidate to use is an RNN model. However, an LSTM model is best for finding long-term dependencies within time series data. We tested our models with up to 72 hours of previous data, with many data points contained within these hours. Using a standard RNN would most likely lead to errors in the gradient culminating in skewed results. By utilizing LSTM's powerful gating structure, we can consider data points further back in time patterns without overwhelming the model with repetitive data and calculations. The LSTM model should help us find long-term dependencies within the data to best predict the future.

3.3 CNN-LSTM

Both CNN and LSTM have powerful abilities that are useful to our problem, so rather than using only one model at a time, we can leverage both simultaneously. CNN-LSTM hybrid model is a combination of both models where the output of the CNN layers is fed as an input into the LSTM layers. This model combines the strength of both the LSTM and CNN models and is theoretically the strongest model for our data. Figure 11 shows a suggested architecture for the hybrid model.

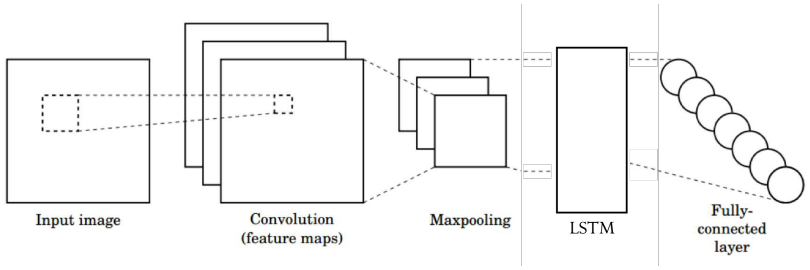

Figure 11: Suggested Structure for CNN-LSTM.

3.4 Hyper-Parameter Tuning

Each of the prescribed models has many hyperparameters that shape the model structure and influence the training process. To find the best-performing model, we need to extensively search through a wide range of hyperparameters. We utilize the keras-tuner tool [4] to help in this task. Table 2 summarizes the hyperparameters chosen for the search process. After training all these models, we rank the models based on the validation loss and save the best model for each look-back window. So, in the end, we will have nine candidate models, three versions per model.

Table 2: Hyperparameters for Different Models

Parameter	**CNN**	**LSTM**	**CNN-LSTM**
Batch Size	[32, 64]	[32, 64]	[32, 64]
Neurons	[16, 32, 64]	[16, 32, 64]	[16, 32, 64]
Filters	N/A	[16, 32, 64]	[16, 32, 64]
Kernel Size	N/A	[2, 3, 4]	[2, 3, 4]
Activation	[relu, tanh, sigmoid]	[relu, tanh, sigmoid]	[relu, tanh, sigmoid]
Pool Size	N/A	[1, 2]	[1, 2]
Learning Rate	[1e-2, 1e-3, 1e-4, 1e-5]	[1e-2, 1e-3, 1e-4, 1e-5]	[1e-2, 1e-3, 1e-4, 1e-5]
Loss	[mse, mae]	[mse, mae]	[mse, mae]

3.5 Transfer Learning

As our project tackles multiple zones with the same task, the transfer learning approach [5] can help us improve the performance of the predictions across different zones. It can help in saving training time, we can focus our efforts on enhancing one model for a representative zone and then apply it directly to other zones with minor retraining. Additionally, if we have

limited data for some zones, transfer learning can help enhance the model and overcome the problem of data shortage.

As shown in Figure 8, we will take the best model from the previous pipeline and retrain it after freezing all layers except the output layer. Additionally, for the sake of bench-marking, we will train each zone with full training from scratch and with limited data from scratch, then we will use transfer learning and retrain with full data and retrain with limited data. So, we will have four versions in total. More details will be provided in the results section 4

4 Results

In this section, we will present the results of the proposed framework. The study was done using the data described earlier in section 2. All models were implemented using the Python language [6] and TensorFlow version 2.12.1 [7] specifically Keras [8] and Keras-Tuner libraries. We split the dataset into 80% training and validation and 20% testing (4 years of training and 1 year of testing). The validation set is 20% of the training data.

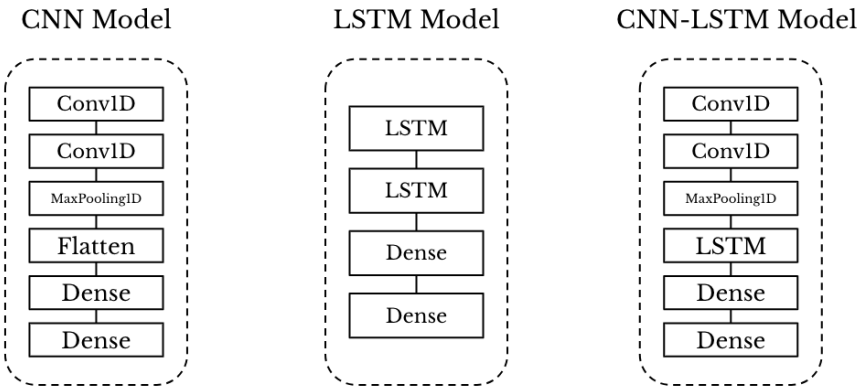

Figure 12: Models' proposed structures.

Figure 12 shows the model structures adapted for each model. These structures have been chosen after a careful experimentation phase, where we found that these structures have the best performance and are not too complex. CNN model will be made of two subsequent Conv1D layers followed by a MaxPooling1D layer, then a Flatten layer, and two subsequent Dense layers with the last one serving as the output layer with shape 24. The CNN model will be made of two subsequent Conv1D layers followed by a MaxPooling1D layer, then a Flatten layer, and two subsequent Dense layers with the last one serving as the output layer with shape 24. The LSTM model will be made of two subsequent LSTM layers followed by two subsequent Dense layers with the last one serving as the output layer with shape 24. The CNN-LSTM hybrid model will be made of two subsequent Conv1D layers followed by a MaxPooling1D layer, then an LSTM layer, and two subsequent Dense layers with the last one serving as the output layer with shape 24.

The following subsections will describe the results for each model, compare the best models, report the performance of the best model for all zones, and finally present transfer learning results.

4.1 CNN Model Results

After building the model with the structure outlines in Figure 12, we start training the model with keras-tuner to find the best models. The hyperparameters considered are shown in Table 2. Table 3 summarizes the top ten models after the hyperparameter search out of 60 models. We can see that the mean square error (mse) loss function achieves the best performance always, which makes sense as it is easier to converge. Also, we can see that 'relu' and 'tanh' both work fine as activation layers. The best number of filters is 32.

Table 3: CNN Model Top 10 hyperparameters based on validation loss.

Lag (Slider)	Neurons	Filters	Activation	Learning Rate	Kernel Size	Pool Size	Batch Size	Loss	Val Loss
48	64	32	relu	0.0001	3	1	32	mse	0.1918
24	64	32	relu	0.0001	3	1	32	mse	0.1922
72	32	32	tanh	0.001	4	2	32	mse	0.1931
72	64	32	relu	0.0001	3	1	32	mse	0.1941
24	64	32	relu	0.0001	3	1	64	mse	0.1942
24	32	32	tanh	0.001	4	2	64	mse	0.1945
48	32	32	tanh	0.001	4	2	32	mse	0.1957
48	64	32	relu	0.0001	3	1	64	mse	0.1963
24	32	32	tanh	0.001	4	2	32	mse	0.1964
72	64	32	relu	0.0001	3	1	64	mse	0.1966

Now, we take the best model for each lag (24, 48, and 72). These will serve as CNN candidate models. We test these models and report the testing performance in Figure 19.

In order to have a better sense of the predictions, Figures 13-14 show the predictions for CNN's candidate models for the NYC zone during July month in the testing year. As the predictions are made in a moving window fashion, the first figure shows the first hour for each prediction, and the second figure shows the 24-hour prediction for a fixed hour daily. We can see that all candidate models are doing a good job in the first hour, but the CNN-48 has better predictions when it comes to spikes. However, when we look at the 24-hour prediction, we can see that it is becoming much harder to get the spikes for all the models, but generally, CNN-48 still has the best overall performance.

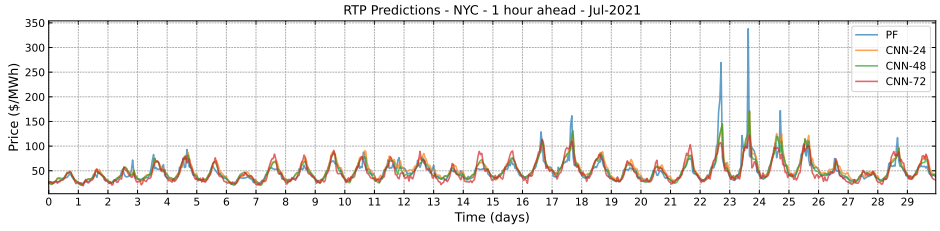

Figure 13: CNN model hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

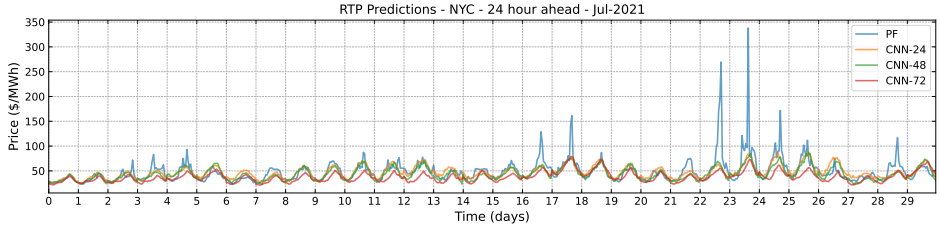

Figure 14: CNN model 24-hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

4.2 LSTM Model Results

With LSTM model, we followed the same process as the CNN model. Table 4 summarizes the top ten models after the hyperparameter search. We can see that the mean square error mse is the best loss function too. Also, we can see that 'tanh' works the best for the LSTM model and 32 neurons seem to be enough. Worth mentioning that LSTM benefits more from longer number of lags as the best was found with 72 lags.

Table 4: LSTM Model Top 10 hyperparameters based on validation loss.

Lag (Slider)	Neurons	Activation	Learning Rate	Batch Size	Loss	Val Loss
72	32	tanh	0.001	64	mse	0.191766
48	32	tanh	0.001	32	mse	0.192951
48	32	tanh	0.001	64	mse	0.193627
72	32	tanh	0.0001	64	mse	0.193819
72	32	tanh	0.001	32	mse	0.194875
48	32	tanh	0.0001	32	mse	0.195327
24	32	tanh	0.001	32	mse	0.195523
24	32	tanh	0.001	64	mse	0.196736
24	32	tanh	0.0001	32	mse	0.197818
72	32	sigmoid	0.001	64	mse	0.19983

Now, we take the best model for each lag (24, 48, and 72). These will serve as LSTM candidate models. We test these models and report the testing performance in Figure 19.

Figures 15-16 show the predictions for LSTM's candidate models for the NYC zone during July month in the testing year. We can see that all candidate models are doing a good job in the first hour, but the LSTM-72 is the closest model to the ground truth. However, when we look at the 24-hour prediction, we can see that LSTM-48 is under-predicting most of the time while LSTM-24 is over-predicting and LSTM-72 has been in-between.

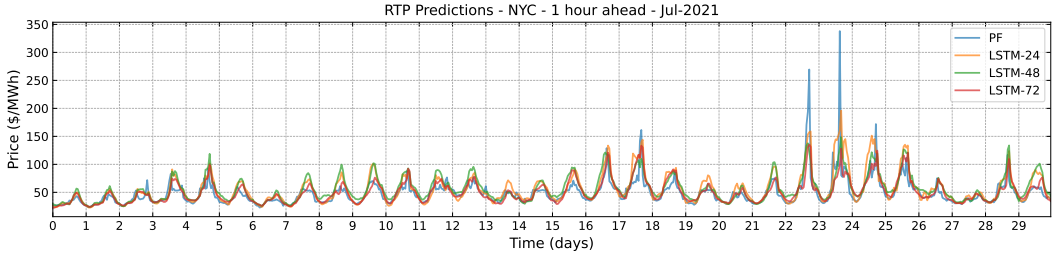

Figure 15: LSTM model hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

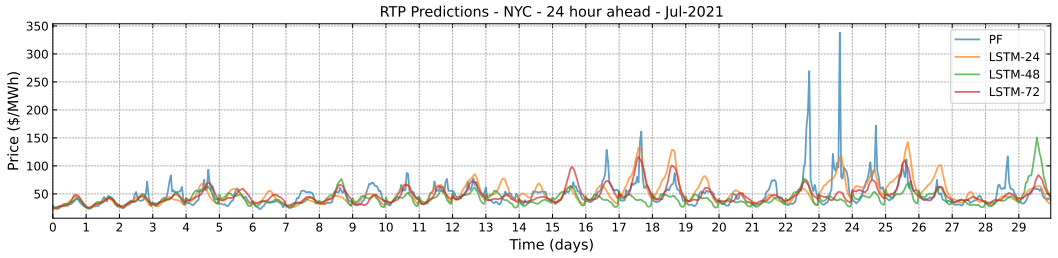

Figure 16: LSTM model 24-hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

4.3 CNN-LSTM Model Results

Similar to the previous models. Table 5 summarizes the top ten models after the hyperparameter search. We can see that the mean square error mse is the best loss function too. 'relu' and 'tanh' both work fine, and 40 filters seem to be perfect. Seem as LSTM, it benefits more from a longer number of lags as the best was found with 72 lags.

Table 5: CNN-LSTM Model Top 10 hyperparameters based on validation loss.

Lag (Slider)	Neurons	Filters	Activation	Learning Rate	Kernel Size	Pool Size	Batch Size	Loss	Val Loss
72	32	40	tanh	0.001	3	1	32	mse	0.1947
72	64	40	relu	0.001	3	1	64	mse	0.1954
48	32	40	tanh	0.001	3	1	64	mse	0.1962
72	32	40	tanh	0.001	3	1	64	mse	0.1968
72	64	40	relu	0.0001	4	2	32	mse	0.1973
24	32	40	tanh	0.001	3	1	64	mse	0.1974
72	64	40	relu	0.001	3	1	32	mse	0.1977
24	64	40	relu	0.001	3	1	64	mse	0.1984
24	64	40	relu	0.001	3	1	32	mse	0.1985
48	64	40	relu	0.0001	4	2	32	mse	0.1995

Now, we take the best model for each lag (24, 48, and 72). These will serve as CNN-LSTM candidate models. We test these models and report the testing performance in Figure 19.

Figures 17-18 show the predictions for CNN-LSTM's candidate models for the NYC zone during July month in the testing year. We can see that all candidate models are doing a good job in the first hour, but the CNN-LSTM-24 is the closest model to the ground truth. However, when we look at the 24-hour prediction, we can see that all models are mostly under-predicting.

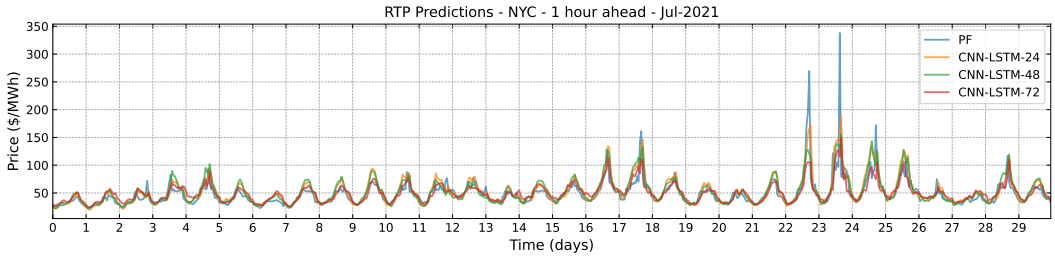

Figure 17: CNN-LSTM model hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

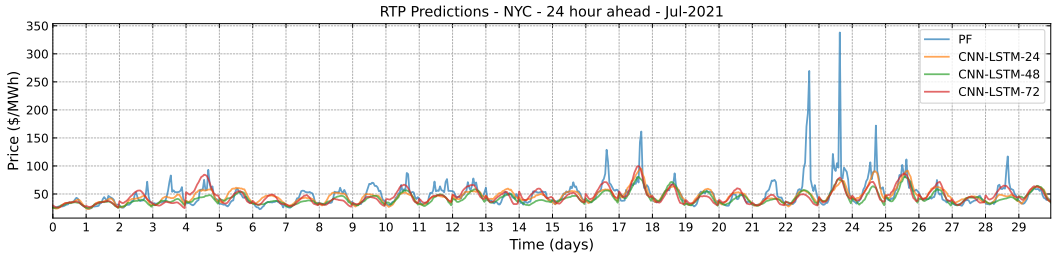

Figure 18: CNN-LSTM model 24-hour ahead predictions from the three candidate models for NYC zone during Jul-2021.

4.4 Overall Comparison

In the previous subsection, we saw how each model is performing under training and we got nine candidate models in total. In this subsection, we will compare these models, analyze their errors, and elect the best overall model.

Figure 19 shows the performance of the candidate models on the testing set. We can see that CNN is the best-performing model in the three versions. Notably, LSTM seems to be underperforming in the testing set, suggesting that it was over-fitted to the training set, as we can see that it had a better training model than the CNN in the 72 version, but then in the testing, the CNN became the best. The CNN-LSTM model has been too an over-fitting victim.

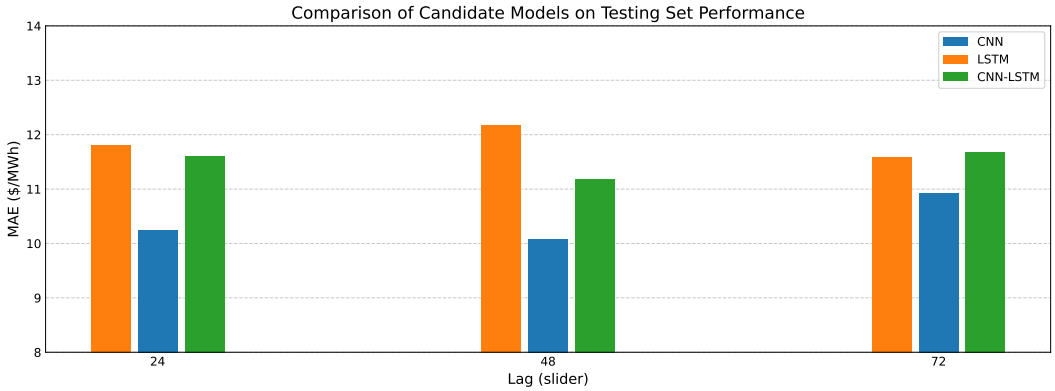

Figure 19: Testing performance comparison of the nine candidate models.

Usually, LSTM can be better than CNN in time-series tasks, but we saw in this demonstration that CNN was better. One explanation for this is that our dataset seems to be much harder than any traditional time-series dataset, where we can see that the price profile in figure 3 is almost random with hardly any pattern. CNN model was able to find more complex patterns and associations in the features making it the best model in this task. the CNN-LSTM model was in-between these two models.

To better understand the influence of the errors and to see where to improve in the model. Figure 20 shows the hourly mismatch between the forecast and the ground truth when we only consider 24-hour lags. We can see that the first hour of the predictions is the most accurate one, and gradually the task becomes harder, and after the fifth hour, all the remaining hours become similarly harder. Clearly, the CNN model has quite a gap from the other models.

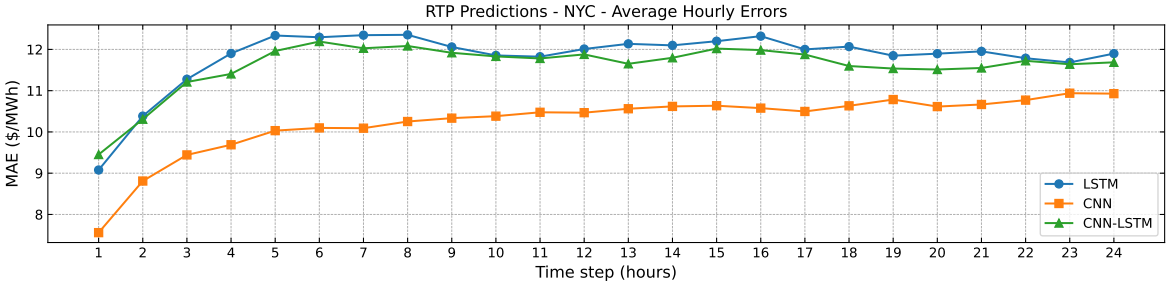

Figure 20: Hourly errors of the three models for the 24 lags version.

Figure 21 shows a similar trend as the case with 24-hour lags with improvement for the CNN and the CNN-LSTM models. The CNN model was able to keep the error under $10/MWh until the 8th hour.

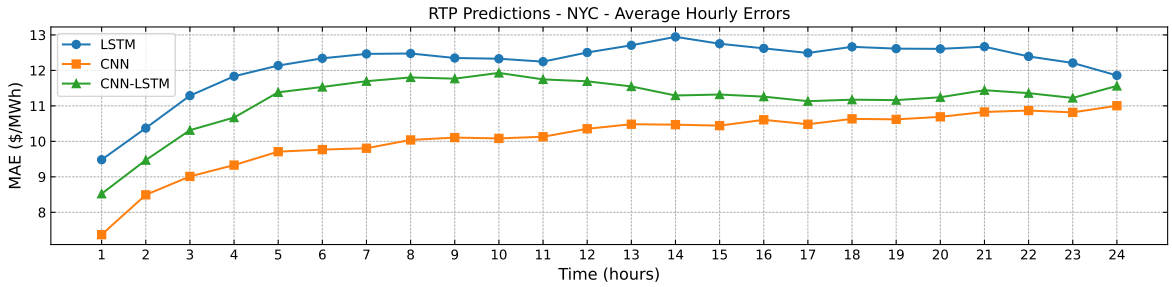

Figure 21: Hourly errors of the three models for the 48 lags version.

Figure 22 shows the hourly mismatch when we only consider 72-hour lags. Now, we can see that CNN-LSTM became the best model in the first hour, but as the horizon gets larger, the error becomes larger quickly, LSTM model shares the same behaviour. On the other hand, the CNN model is the most resilient, as the performance degradation is slower than the other two models, and on average it is the best-performing model.

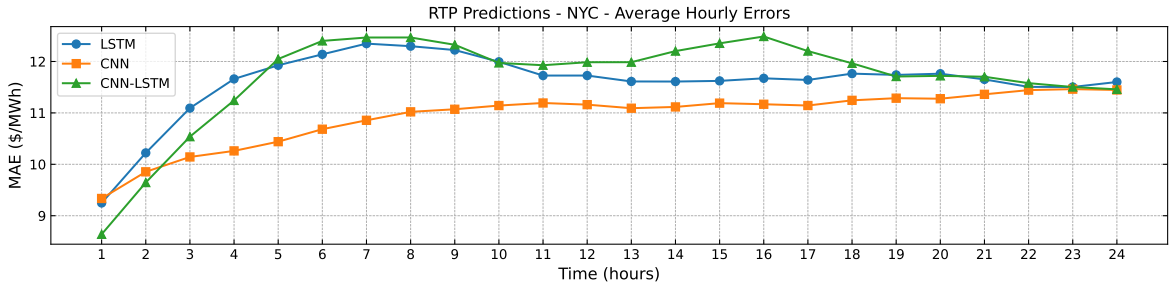

Figure 22: Hourly errors of the three models for the 72 lags version.

4.5 Transfer Learning Results

In this subsection, we present the result of implementing the idea of transfer learning explained in 3.5. In order to evaluate the effectiveness of this idea, we made four different cases: two cases using the full training set (4 years) and two cases using the limited training set (1 year). One is training all eleven zones on the CNN-48 model from scratch, the other is retraining a pre-trained model.

Figure 23 shows the prediction accuracy for all NYISO load zones after training it on the CNN-48 model from scratch using 4 years and 1 year of training. We can clearly see that training on more data will lead to a better prediction model. Some zones suffer greatly when face a shortage in data such as the GENESE zone its error increased by almost 50%. Most zones are within errors of $10/MWh except the LONGIL zone its error is more than double, this is due to its high prices and extreme spikes as we can see in 1 its standard deviation is more than $60/MWh and it is higher than its mean. The error can be understood when we compare it with the mean of the prices. We can see most zones are achieving an error that is less than half to one-fourth of the mean, which is a reasonable achievement.

We saw in the previous figure the performance of training each zone from scratch. Now, we will use transfer learning and see how much improvement percentage we can make for each case. We used the CNN-48 model that is trained on the NYC zone with a full training set, and we will apply it to all the zones and retrain it by freezing all layers except the last one which is the output layer.

Figure 24 shows the prediction improvement percentage after transfer learning. We can

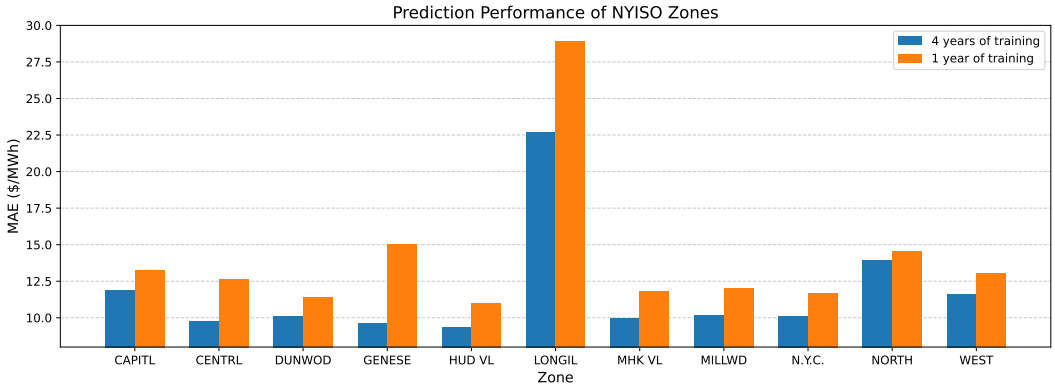

Figure 23: Prediction accuracy for CNN-48 model trained on each zone considering different training sets.

see that all zones have improved by at least 6% to 23% when it has limited training data. This shows that transfer learning is very effective in such tasks. On the other hand, with the case of no training data shortage, the transfer learning can find it hard to improve the performance but can get very close results with minor performance loss. This can be helpful to save training time, as transfer learning takes much less time in retraining (converge in less than 5 epochs).

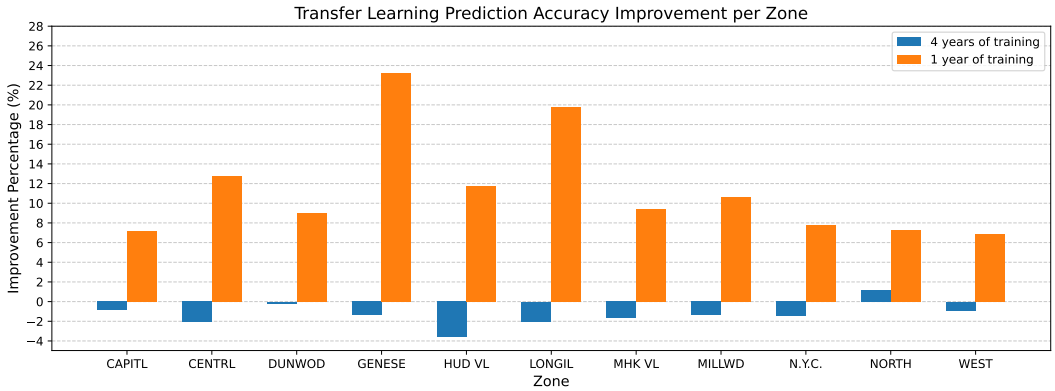

Figure 24: Prediction accuracy for CNN-48 model trained on each zone considering different training sets.

5 Conclusion

In this report, we have suggested a full machine-learning framework for electricity RTP prediction. We have utilized several methods described earlier. We have shown that the CNN model is the best overall model and having two historical days are enough to predict the next 24-hour prices with reasonable accuracy. We also saw that hourly prediction errors increase logarithmically with time, with the first hour as the most accurate. Moreover, we investigated the effectiveness of the transfer learning approach. We saw that transfer learning can help in overcoming data shortage problems with accuracy improvement reaching 23% from not using it and it can save more time as it can train on new data rapidly.

For the next steps, we would like to investigate the idea of a combined classification and regression framework by adding an anomaly detection part, where we try to detect spikes occurrence and then feed these to the regression part of the framework. Such an approach can help minimize the mismatch in prediction as the price patterns are heavily affected by spikes throughout the day.

References

[1] J. Lago *et al.*, "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark," *Applied Energy*, vol. 293, p. 116983, 2021.

[2] NYISO, "Energy market & operational data." https://www.nyiso.com/energy-market-operational-data.

[3] S. Hochreiter *et al.*, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, nov 1997.

[4] T. O'Malley *et al.*, "Keras Tuner." https://github.com/keras-team/keras-tuner, 2019.

[5] K. Weiss *et al.*, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[6] G. Van Rossum *et al.*, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[7] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[8] F. Chollet *et al.*, "Keras." https://keras.io, 2015.