Justus Purat and Alexander Kammeyer

Software Project Distributed Systems

Consumption Data Forecast for HPC Systems

Muneeb Ch, Ahmet Er, Aron Huth
Yasin Kaya, Mandeep Karn, Michael Zent

Freie Universität Berlin
Institute for Computer Science

SS 2025

Table of Content

1. Introduction

Computing systems continuously become more powerful. While bringing tremendous use, that is accompanied by monetary and ecological costs inter alia in the form of energy prices and CO_2 emissions. Nowadays, High-Performance Computing (HPC) systems have risen to one of the largest cost factors in today's high-technology industry. Hence, it is necessary to balance the striving for higher performance with a minimization of the costs.

Therefore it is desirable to be able to forecast energy prices and the CO_2 emission intensity of the energy production, and such be capacitated to make more sustainable decisions. For instance one could schedule resource-intensive tasks for periods when energy prices are lower, or reduce system performance when high CO_2 emissions are expected.

This work aims to achieve a viable 24h-forecast of the energy prices in Germany, and at that to establish the necessary basis for later refinement and extension.

2. Approach

The basic idea is to forecast the energy price development using a machine learning model trained on corresponding time series data. Approaching a problem like this involves several layers of complexity, including data collection, data analysis, preprocessing, model selection, training, and validation.

First of all, a reliable dataset is required on which the model shall be trained. For this purpose, access to an InfluxDB time series database was provided which contains various HPC consumption datasets to be queried via an API using Flux, a functional data scripting language supported by programming libraries compatible with the InfluxDB client.

The database contains various time-series datasets regarding energy production, energy prices, CO_2 emissions and weather. Although the main focus of this project is to forecast energy prices, and in perspective CO_2 emissions, we considered the other data sources as well. Further analysis revealed that there are correlations between the target data and the provided supplemental information which could be used to improve the accuracy of the forecasts.

Another key issue we encountered early on was the need for data pre-processing. Some records in the database were missing, while others showed abnormal spikes

or noise that could distort the training process. Cleaning and transforming the data was therefore a necessary step before model training could begin.

Finally, an appropriate machine learning model had to be selected. With the rapid advancement of machine learning and data science, we were faced with a wide range of model architectures. To make an informed choice, we first had to familiarize ourselves with different forecasting techniques and evaluate their performance on the provided dataset. This required running the cleaned data through multiple candidate models, comparing the forecasted outputs with actual values, and validating the results to identify the most accurate and suitable model for our use case.

3. Related Work

Due to the growing importance of data science and forecasting in areas like energy systems and sustainability, there is a wide range of research related to our topic. Although various sources influenced our work, two in particular were especially helpful during the course of this project.

The first is the set of papers published by our project stakeholder Alexander Kammeyer in 2023 and 2024. Although his work is not directly focused on forecasting models, it provides valuable insight into the optimization of HPC systems. His papers propose working with a digital twin of a HPC cluster, i.e. a virtual real-time representation that integrates both internal system metrics and external data sources to simulate and optimize resource usage. Concepts such as power-aware scheduling and dynamic system reconfiguration contribute to a broader understanding of energy efficiency and sustainability in HPC environments. These ideas gave us a deeper perspective on the role of predictive models in adaptive system operation.

The second major influence was the work done by the student team that worked on this project during the software project's last iteration in 2023, providing experience to build on. A forecast on the German energy production's CO_2 emissions was made based on energy mix data, using a Random Forest Regressor trained on the Exponential Moving Average, yielding a mean absolute prediction error (MAE) of 51% over 7 days. Another forecast considered the production of each energy carrier, using a Long Short-Term Memory Neural Network and yielding a MAE of 9-505% over 7 days, depending on the regarded carrier. The energy price forecast

was made with Random Forest Approaches, the best of which reaching a MAE of 25% over 4 days.

The main take-aways were:

- The Random Forest Tree produced correct trends, but often exaggerated.
- The used Long Short-Term Memory Neural Networks reacted badly on any short- and mid-term deviation from long-term trends.
- Bad reaction on unexpected events, e.g. the shut-down of nuclear energy.
- No use of correlations made.
- No aggregation of closely related features.
- MAE is not meaningful when not normalized by the data's value range.
- Inconvenient collection of python notebooks, repeating basic data handling and pre-processing tasks again and again.

Hence we decided to establish an application programming interface which allows to manage and prepare the data for forecasting in an unified way for console Python programs. That should include methods for distance correlation analysis and the creation of artificial features like lagged values, and enable trend predictions. Further, we decided to try to find an own modeling approach as the earlier results didn't seem clear and would require a complete rewrite anyway.

4. Deep Learning Model Setup

Instead of relying on already trained and pre-existing forecasting frameworks and machine learning algorithms, we decided to implement our own deep neural network. The reasons for doing so will be discussed in detail in the following, however, simply put, our experiments with the Random Forest Regressor, Prophet, and Lag LLama didn't yield good performance results on our data.

4.1. Data Preparation

We were retrieving the data from multiple databases stored in InfluxDB. The entries in the databases were 15 minutes apart. However, we aggregated for an hour and considered the mean hourly values for all the fields in all the databases during the retrieval of the data to our working memory as the energy prices, the data of interest, are provided with a resolution of 1 hour. Our main goal was to be able to make forecasts at least a few hours, ideally for a whole day.

To go about doing this, we worked on two different approaches. Our first approach was a multivariate one. The thought process behind this was the reasoning that the

price values must have a correlation with other factors such as weather and energy production. However, we later abandoned this strategy because of two reasons:

- The uncertainty of the availability of some correlated features during the time of inference.
- The price correlated to a single other feature in an almost 1:1 relation which however contained no information which could be used to improve the model. Other features were weakly correlated, which maybe reflects a highly regulated market. Such the price basically was an univariate feature anyway.
- Our research showed that in similar scenarios all state-of-the-art forecasting models follow an univariate approach instead of a multivariate one.

4.1.1. Univariate Approach

The univariate approach is the type of data engineering methodology where the new features are engineered solely from the targeted variable in the form of lagged and rolling-mean features. To figure out which of them would be the most helpful in forecasting, we employed distance correlation analysis and considered those with a dCor value of larger 0.8 to be significant.

Regarding lagged features, those were of use which accorded to multiples of 24h of lag, including the values ±1h. For the rolling-mean features the same periodicity of 24h could be observed, however the multiples themselves naturally were of no use but the +1h and the -1 and -2h values.

4.1.2. Time based features

In addition to engineered features discussed above, we also created time based features such as hour, year, day of the week, and day of the year. Simply applying sine and cosine transformations on them to cater for their cyclic nature generated only features of weak correlation, so we relied only on the “raw” time values to contain the periodicity information. However, it might be an aspect for future research to explore how to modulate the sinusoid such that it becomes useful.

4.1.3. Creation of Data Sequences

To be able to use the data we had in memory for model training and forecasting, we had to reshape it as a 3D array. Each index of a 3D array was a sequence of hourly data consisting of time, lagged, and rolling mean features represented as a 2D array. This phase also required a lot of research and experimentation. In the end, we figured out that the sequence of 48 hours of data to forecast price 1 hour in the

future works the best. We also realized that we could then potentially use that hour's forecast to do recursive forecasting for up to one week into the future.

4.1.4. Preprocessing

After the feature engineering and sequence creation, the next step was to normalize the data. We did the normalization by applying the Min-Max-Scaler from the Sklearn library. We also divided the data into training and validation sets. We used 90% of the data for training and 10% for validation.

4.2. Model Architecture

There has been a lot of research recently on what type of deep learning architectures yield the best results on forecasting tasks. The **LSTM** architecture has been available for a long time, while the **Transformers** architecture has revolutionized the solutions to forecasting problems recently. We experimented with both these architectures but ultimately decided to choose LSTM architecture for this project because not only did it give us comparatively better results on our data, but also more control with what we wanted to achieve.

A LSTM deep learning architecture is based on either a single LSTM layer or multiple layers stacked on top of one another. These **Long Short-Term Memory (LSTM)** layers form a type of recurrent neural network (RNN) specifically designed to handle sequential data and learn long-term dependencies between data points. The main difference between the LSTM and Transformers is that the LSTM layers process the sequential temporal data one element at a time using gated memory cells, while the Transformers process the entire input sequence simultaneously using self-attention mechanisms.

4.2.1. Convolution 1D and Max Pooling

Instead of directly feeding the raw temporal sequences to LSTM layers, we decided to pass the data through a couple of 1D convolution layers first. 1D convolution layers apply filters on temporal sequences that work across a single dimension – the temporal one – as opposed to 2D convolution filters that work across two dimensions of 2D images (vertically and horizontally). Filters are basically sliding windows that detect specific "shapes" in data (e.g., a sudden spike, a dip, and a short-term trend reversal). These filters learn local patterns and features across different time steps (time step is a data point) within a sequence. After each convolution layer, we also added a Max pooling layer which served the purpose of

reducing the size of resulting learned sequences after convolution while retaining all the important information.

4.2.2. LSTM Layers

After the initial application of 1D-convolution, the middle of our deep neural network consisted of a couple of LSTM layers. The implementation of such a structure served two purposes:

- The convolution layers extracted the robust, high-level local features, and then the LSTM layers processed these learned features sequentially to understand long-term dependencies between data points.
- The combination allows the LSTM layers to receive a richer, more abstract representation of the sequence, rather than just raw time series values.

4.2.3. Artificial Neural Network

The tail of our deep neural network was a simple artificial neural network with one input layer, one hidden layer, and one output layer. The output layer only had one neuron which gave us the value of the forecasted price one hour in the future.

4.3. Loss Function

The default go to loss function for us at the start was the mean squared error. However, after the initial runs, we realized that the price values fluctuated a lot during several hours of the day which was depicted in the form of peaks and dips on the graph. These fluctuations weren't captured with great accuracy by our deep neural network. After changing the hyper-parameters many times without seeing much improvement, we pin-pointed the problem to the selection of the loss function.

4.3.1. Drawbacks of Mean Squared Error

The biggest drawback of MSE is that it applies quadratic penalties for all errors. That makes the model more conservative as it tries to minimize the squared error across all data points by finding a middle ground. Thus, the fear of a large penalty discourages the model to go for the peaks and the dips. Instead, it tends to stay near the middle where the overall error is minimum. Another drawback of MSE lies in its symmetrical nature. It tends to treat all errors the same which means no domain knowledge can be embedded in the model.

4.3.2. Asymmetric Custom Loss Function

To deal with these drawbacks, we decided to implement our own custom loss function which had three crucial control layers. For the first one we introduced a variable “delta” which controls the behavior of the loss function. For errors less than delta, it behaved quadratically, and linearly for those bigger than delta. At that, a large delta means that the loss function behaves again like the MSE, and a small delta means that the loss function behaves as the mean absolute error (MAE). The applied value of 0.01 was found experimentally.

The second control layer was introduced using a couple of more variables. These variables allowed us to make the loss function asymmetric in nature, allowing us to customize the function according to the domain needs. In the current state of our loss function, overpredictions incur 30% more penalty than underpredictions.

The third layer gave us more control through adjustment of penalties for peaks and dips. Penalty increases for underprediction when the actual normalized price gets higher for peaks. Similarly, penalty increases for overpredictions when it gets lower for dips. This was done to encourage the model to go for the sudden peaks and dips in the price values.

4.4. Training

The training process with our customized loss function went on for about 68 epochs. The best training metrics were achieved during the 53rd epoch. We had implemented the early stopping with a patience of 15 to avoid overfitting, as depicted in Fig.1. After the 53rd epoch, the model performance stagnated and didn’t improve on the validation data.

The results achieved at epoch 53 were as follows:

- Training Loss 1.0905e-04
- Training MAE 0.0060
- Training MSE 1.0004e-04
- Validation Loss 2.0292e-04
- Validation MAE 0.0087
- Validation MSE 3.1625e-04

The metrics were calculated on the normalized data rather than the actual data, that’s why the values are so small.

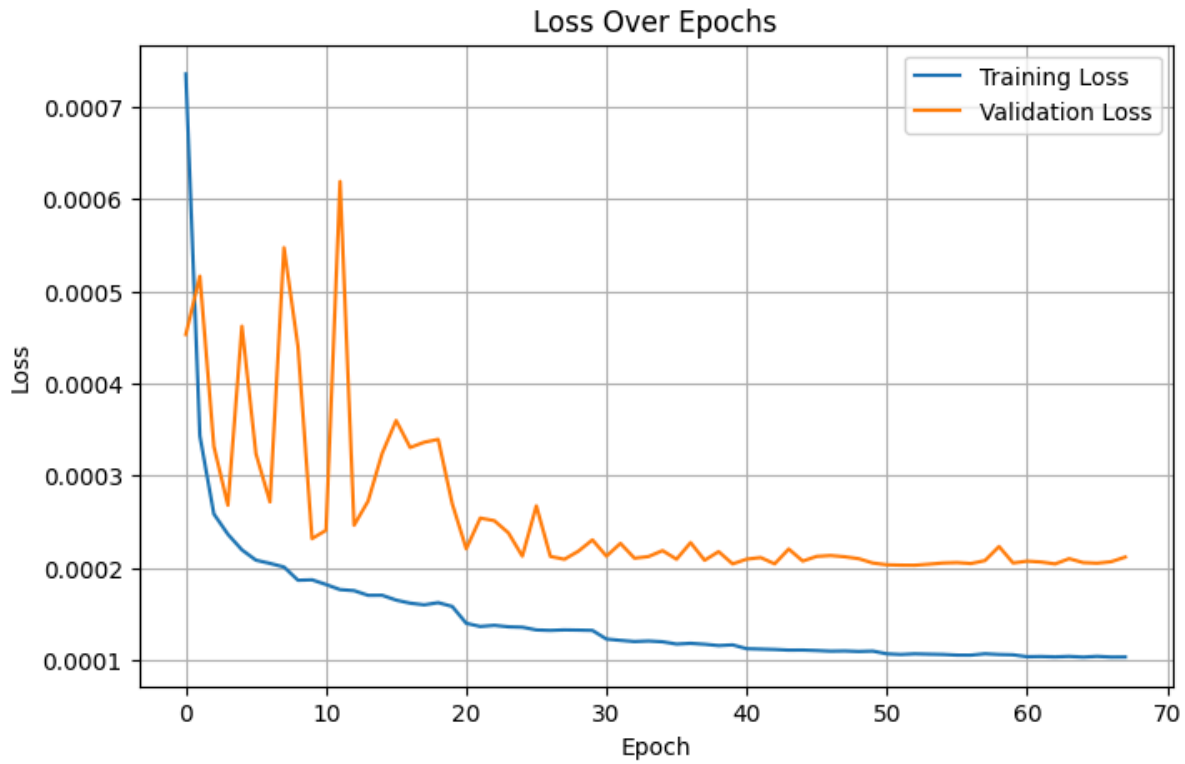

Fig.1: Loss development during training of the LSTM neural network.

4.5 Recursive Forecasting

After the training, we used our model for inference on the unseen data. This was done recursively. We started off with an initial sequence of 48 hours of data. We used that data to make a forecast of the price value for 1 hour into the future using our model. Then the forecasted price value was hypothetically appended at the end of the sequence in the form of its lagged, rolling, and time based features and the top row of the sequence was removed. Eventually, each forecasted price value served the purpose as the lag or rolling mean feature for another price entry in the future. We used this methodology to recursively forecast as far ahead in the future as we wanted using a base model that had the capability of only predicting 1 hour in the future.

4.6 Note on Foundation Model

We explored whether a **Foundation Model** for time-series data could act like a large language model (LLM) – making accurate forecasts for new tasks right out of the box. Our study focused on Lag Llama, an open-source model pre-trained on a diverse corpus of nearly 8,000 time series from sectors like energy, transportation and economics.

We tested Lag Llama in two ways – as a zero-shot model, i.e. without new training, and after fine-tuning to our specific data. We measured its performance using standard metrics like MAE and RMSE, along with probabilistic forecasts.

4.6.1. Pattern Recognition

Right away, we noticed the model was fantastic at identifying the underlying patterns in the data. Even with no task-specific training, its forecasts visually captured the correct direction of trends, cycles, and seasonal shifts. This shows that its extensive pre-training taught it the fundamental rhythms of time-series data. In short, the model was great at predicting when changes would occur.

4.6.2. Magnitude Prediction

Despite its strength in pattern recognition, the model struggles to predict the size of those changes. The quantitative results revealed significant weaknesses. A very high Mean Absolute Percentage Error (MAPE) of 60% pointed to large errors, while a more moderate Symmetric MAPE (sMAPE) of 39.9% suggested the issue was related to scaling its predictions, not a systematic bias. It knew which way the data was heading but could not accurately guess by how much.

4.6.3. Fine Tuning Paradox

Surprisingly, attempting to fine-tune the model on our data made its performance worse, not better. We suspect this could be due to a few factors – small epochs for effective fine-tuning due to low computation resources, non-optimal training settings, or the fine-tuning process accidentally overwriting the valuable general knowledge the model had learned during pre-training.

On a better note, its probabilistic forecasts, which measure uncertainty, were reasonably well calibrated. For example, its median (50th percentile) predictions covered the actual value 51.0% of the time.

4.6.4. Key Insight

Our results highlight that successful time-series forecasting involves mastering two separated skills:

- Pattern Recognition: Understanding when changes will happen.
- Magnitude Prediction: Predicting how much they will change by.

Foundation models like Lag Llama, thanks to their broad pre-training, already excel at the first task. However, they struggle with the second, failing to calibrate the scale of their predictions to a specific domain.

Furthermore, the model's sensitivity to fine-tuning suggests these powerful tools can be fragile. A straightforward attempt to adapt them can backfire, disrupting their valuable pre-trained knowledge without successfully teaching them the specific scaling information needed for a new task.

5. Progress of Work

Sprint 1

We established the connection to the provided InfluxDB database via experimental Jupyter Notebooks and queried it to become familiar with the structure and shape of the data. Then we started to analyse the data with Python and the Pandas library. We also created charts to get a visualisation and a better understanding of what we are dealing with. In addition, we examined the work from the last project iteration of 2023.

Sprint 2

Continuing to analyse the data, we focused on finding correlations in the energy production bucket but also between the CO_2 and the energy price bucket. Further, we started with the engineering of artificial features and discussed which modeling technique shall be used. At this point we worked on Random Forest Regression, LSTM, Prophet and Lag Llama. Also decisions were made considering pre-processing, especially in favor of the distance correlation which replaced the till then used Pearson correlation.

Sprint 3

Experiments showed potential for LSTM and LagLlama, so we focused on them. Implementation of the data management and pre-processing interface as well as the models for production purposes started.

Sprint 4

Finished the data pre-processing methods for usage in modeling. Also, we concluded that we only use the LSTM Model because the results of Lag Lama turned out to be poor. For LSTM we included Max Pooling Layers.

Sprint 5

Adjustments in the employed LSTM model, like tweaking the selection of the generated artificial features and improving the loss function. Modeling was united with the pre-processing into an overall framework.

6. Interface Documentation

6.1. Class ConsumptionData

Fetching and pre-processing of HPC consumption data. Further referred to as `cd`.

ConsumptionData cd.ConsumptionData(void)
Constructs and returns a new `ConsumptionData` object.

void cd.setConnection(string serverURL, string token, string org)
Configures a connection to an InfluxDB database server, containing the consumption data of a HPC system in a Kammeyer-compliant structure.

`DataFrame fetch(string startTime, string stopTime, float nanThreshold=0.03)`
Fetches via the configured connection to an InfluxDB server the raw data from `startTime` to `stopTime`, and creates an overall table. Columns with NaN values in a share of `nanThreshold` are dropped.

Creates a table, stored within the class objects and also returned by this function. For now, the most important columns are the following ones:

_time	Timestamp, aggregation of per-hour data [datetime]
carbonIntensity	Calculated carbon intensity of measured energy production [float, kg/MWh]
fossilFuelPercentage	Share of fossil energy in the carbonIntensity [float, %]
fossilEnergy	Energy production from Braunkohle, Steinkohle, Erdgas, 'Sonstige Konventionelle' [float, MWh]
renewableEnergy	Energy production from Biomasse, 'Wind Onshore', 'Wind Offshore', Photovoltaik, Wasserkraft, 'Sonstige Erneuerbare' [float, MWh]
price	Monetary energy costs [float, €/MWh]

Further weather data columns, without elaborating here:
cloud_cover, dew_point, precipitation, pressure_msl, relative_humidity, solar, sunshine, temperature, visibility, wind_speed
Additional time columns, a breakdown of the _time column:
month, day, dayofweek, dayofyear, hour

Energy carriers Kernenergie and Pumpspeicher are ignored, as well as the entries for wind_direction and wind_gust_direction.

ConsumptionData cd.deepCopy(void)
Returns a new `ConsumptionData` object, being a deep copy of the calling one.

DataFrame cd.get(void)
Returns a deep copy of the stored data.

void cd.set(DataFrame data)
Sets a deep copy of the provided data.

void cd.toCSV(string filepath)
Writes the stored data to a CSV file under the given path.

void cd.fromCSV(string filepath)
Stores the data read from a CSV file under the given path.

DataFrame cd.dropCols(list columns)
Removes the specified columns in-place from the stored data, and returns a deep copy of the new data set.

DataFrame cd.keepCols(Index columns)
Keeps only the specified data columns, as well as the `_time` column, in the stored data, and returns a deep copy of the new data set.
TODO: Change such that it takes a `list` instead of an `Index`.

integer cd.nanCount(void)
Returns the number of NaN values per data column.

void cd.dropNAN(void)
Drops rows with NaN values in-place in the stored data.

void cd.dropHeadRows(integer number)
Drops given number of head rows in-place in the stored data.

void cd.appendRows(DataFrame rows)
Appends the given data rows in-place to the stored data.

tuple cd.shape(void)
Returns the shape of the stored data as a tuple of integers.

DataFrame cd.toGradient(void)
Transforms absolute measurement values into gradient values.

The gradients between two rows at times t1 and t2 are calculated by

gradients(t2) = values(t2) - values(t1)

There's no division by the time difference as a provision of hourly non-NaN data is assumed, i.e. it always is t2 - t1 = 1.

Changes stored data in-place, and returns a deep copy of the gradient-turned data. Number of rows will be one less, as the first row will become NaN and is dropped.
TODO: Generalization to consider time differences > 1.

DataFrame cd.toAbsolute(DataFrame startValues)
Transforms gradient values into absolute ones, by adding the gradient values cumulatively row by row upon the given start row to begin with.

Changes the stored data in-place, and returns a deep copy of the absolutized data.
TODO: Generalization to consider time differences > 1

DataFrame cd.removeOutliers(float threshold=3, string method='linear')
Removes outliers via Z-Score beyond a `threshold` number of standard deviations by which a value of a measurement has to be away from the measurement's mean value to be considered being an outlier. Interpolates the emerging data gaps via the specified `method`, see pandas.pydata.org for details.

Changes the stored data in-place, and returns a deep copy of the cleaned data.

DataFrame cd.dCorr(string feature, string freq='hour', float threshold=0.4)
Calculates and returns the distance correlation for a given feature to all other features, for their mean sampled with given time frequency, and above the given threshold value.

DataFrame cd.addLag(list columns, list periods)
Generates new columns as lagged features for a given list of columns with a given list of periods. Adds them into the stored data in-place, and returns a deep copy of the new data set.

DataFrame cd.addRollingMean(list columns, list windows, integer shift=0)
Generates new columns as rolling means for a given list of columns with a given list of windows at a given shift. Adds them into the stored data in-place, and returns a deep copy of the new data set.

6.2. Class modelLSTM

Training and forecasting on HPC consumption data. Further referred to as `ml`.

modelLSTM ml.modelLSTM(string modelName)
Constructs and returns a new modelLSTM object with the given name to be used for saving and loading the generated model.

void ml.train(ConsumptionData cd, list featureColumns, string targetColumn, float trainRatio=0.9)
Trains the model on the `featureColumns` for the `targetColumn` of the provided consumption data `cd`, using a `trainRatio` share of the data for training. Creates the following files which describe the generated model:
`<modelName>.keras`,
`<modelName>_minmax_X_scaler.gz`
`<modelName>_minmax_Y_scaler.gz`

DataFrame ml.forecast(ConsumptionData cd, string targetColumn, integer horizon=24)
Returns a forecast on the next `horizon` hours for the `targetColumn` based on the given consumption data and the model which has been created with `train()`.

6.3. Provided Files

For details refer to the following files, provided in the folder `final/` of the supplemental archive file `SWP-DS2025-Consumption-Data-Forecast.zip`.

Requirements

Install the required python packages by executing on the provided requirements file
`pip install -r requirements.txt`

Implementation

Fetching and preprocessing is implemented in `ConsumptionData.py`, model training and the actual forecasting in `TwoSidedAsymmetricHuberLoss.py` and `modelLSTM.py`.

Application

The `app/` subdirectory contains files which illustrate the usage of the provided interface. The files `training.py` and `forecast.py` train and forecast based on

absolute data values, `trainingGrad.py` and `forecastGrad.py` based on gradient-turned values.

Data Files

Further, a CSV file is provided in `app/` with pre-fetched consumption data for three complete years, namely from 2022 till 2024.

7. Results

Forecasts for 24h achieve a normalized MAE, i.e. a MAE divided by the target feature's value range, of 9-13% at yearly or monthly retraining. The closer the training data's end-date to the date for which the forecast has to be done, the worse the normalized MAE is in total, but can become better on certain time sections.

Fig.2 depicts the results after training on full three years from 01.01.2022 to 31.12.2024 using the absolute data values, and forecasting the 27.06.2025. No outliers were removed, as the threshold of three standard deviations would have erased up to 16% of the data which has been considered as too much. 47 created lagged and rolling-mean features were considered significant with a distance correlation value of more than 0.8 and therefore have been used for training and forecasting. A normalized MAE of 9.2% has been achieved, at that following the trends and yielding very close results at the beginning and the end of the day.

Repeating the experiment, but using gradient values instead of absolute ones tends to smooth the curve as depicted in Fig.3. It also allows to remove outliers, as their number drops to less than 5% as only trend-relevant outliers are covered now. Furthermore, the number of lagged and rolling-mean features which remained relevant and had to be considered could be reduced by one sixth to 39 which increases the efficiency of the training and forecast processes. However, this comes at the cost of an increased normalized MAE of 11.9%.

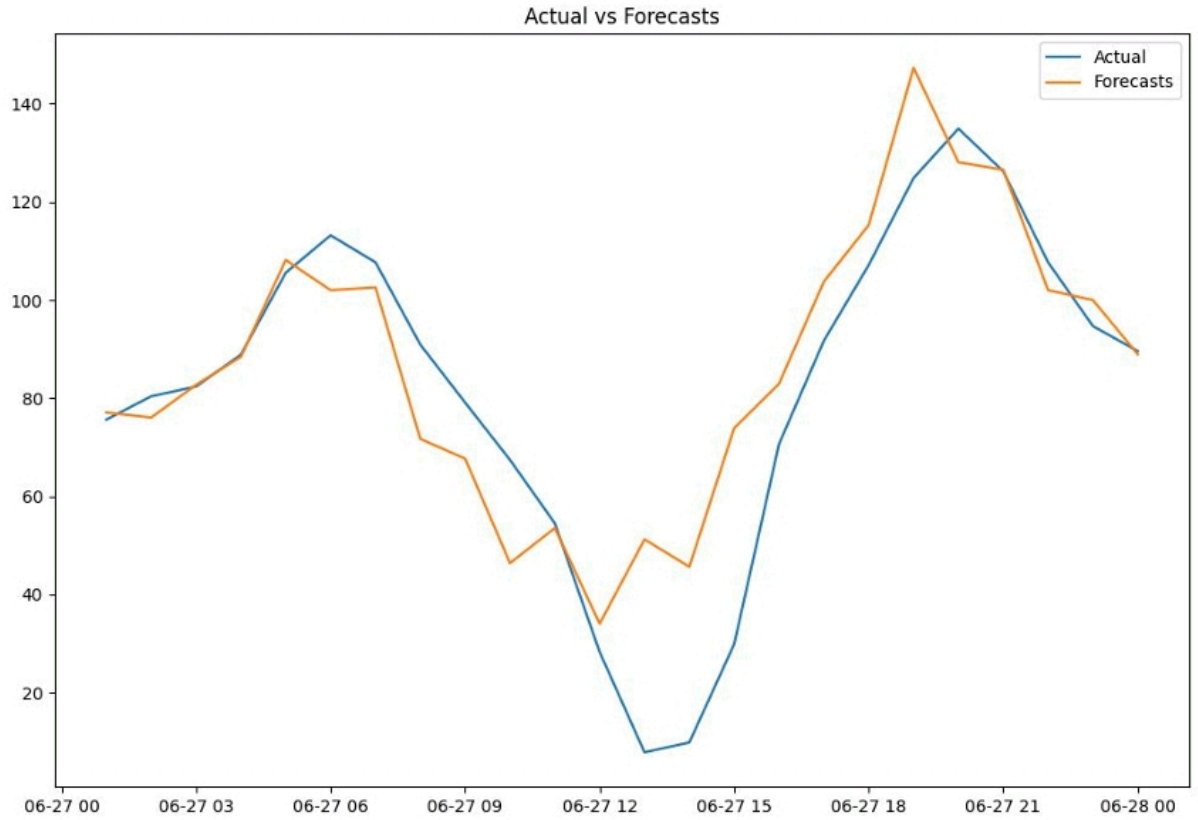

Fig.2: Energy price forecast for 27.06.2025 after training on absolute values from 2022-2024.

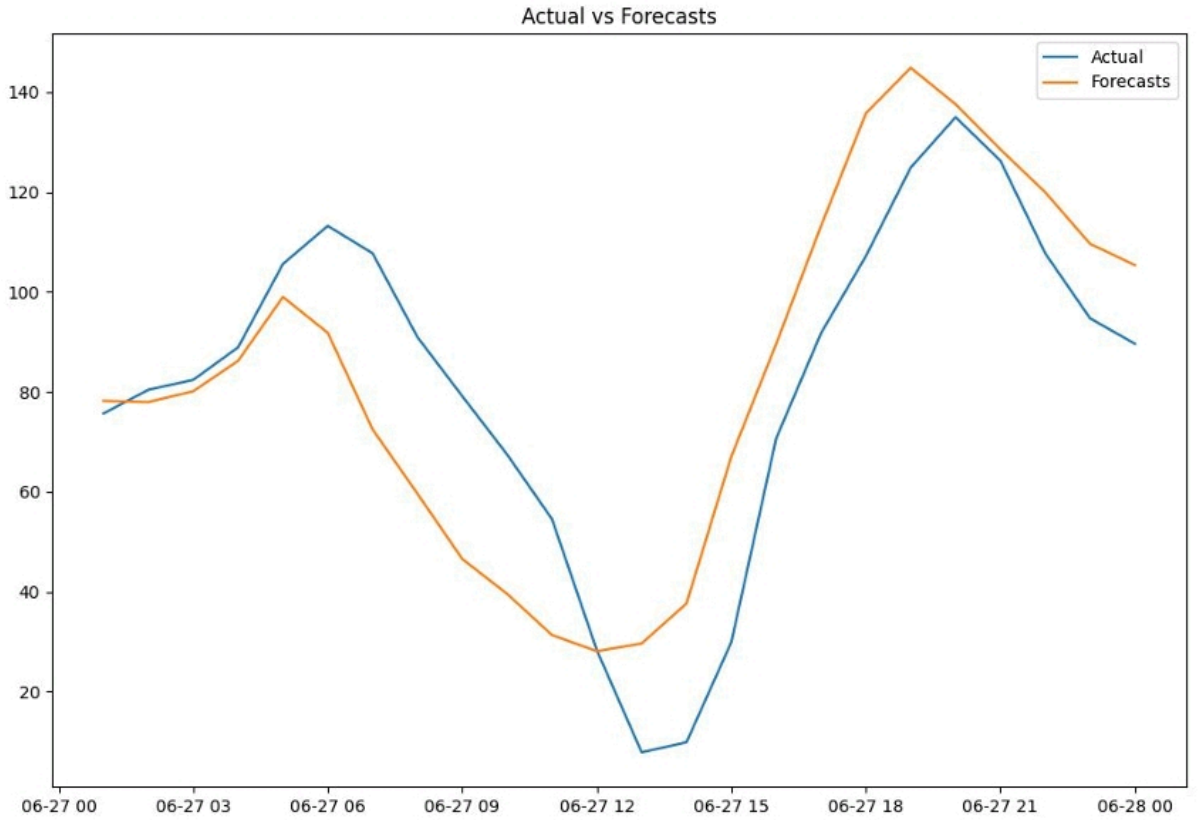

Fig.3: Energy price forecast for 27.06.2025 after training on gradient values from 2022-2024.

8. Conclusion and Future Work

This work established a basic framework for handling and pre-processing HPC consumption data from the Physikalisch-Technische Bundesanstalt, providing methods e.g. for querying and aggregating the data, correlation analysis, creation of artificial values and conversion to and from gradient data. Further, a model was found and implemented which viably represents the available historic data and can be used for forecasting with decent results.

In the future one can refine the framework e.g. by generalizing some aspects like the model parametrization and by adding further data analysis methods. The model itself could be tuned by purging unused neurons, fine-tuning its parameters and searching for further patterns. Also it is desired to forecast the CO_2 emission intensity of the German energy production as well, which probably would have to be made employing a multivariate model, as there are plenty of correlations to other database features.

9. Literature

- A.Kammeyer et al. (2023), *Optimization of Energy Efficiency of an HPC Cluster*, SMSI 2023, pp. 378-379
- A.Kammeyer et al. (2023), *Towards an HPC cluster digital twin and scheduling framework for improved energy efficiency*, ACSIS 35:265-268
- A.Kammeyer et al. (2024), *HPC operation with time-dependent cluster-wide power capping*, ACSIS 39:385-393
- A.Kammeyer et al. (2024), *Developing a digital twin to measure and optimise HPC efficiency*, Measurement: Sensors, 2024.101481
- K.Rasul et al. (2023), *Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting*, arXiv:2310.08278
- G.Székely et al. (2007), *Measuring and testing dependence by correlation of distances*, The Annals of Statistics, 35(6):2769-2794
- C.A.Mertler & R.V.Reinhart (2017), *Advanced and Multivariate Statistical Methods*, 6th ed., Routledge
- A.C.Elliott et al. (2017), *Applied Time Series Analysis*, 2nd ed., CRC
- R.E.Chandler & E.M.Scott (2011), *Statistical Methods for Trend Detection and Analysis in the Environmental Sciences*, Wiley
- W. Palma (2016), *Time Series Analysis*, Wiley