The University of South Bohemia in České Budějovice
Faculty of Science

PV Power Forecasting using Distributed Machine Learning for Smart Energy Grid (An extension to Federated Learning)

Master´s thesis

Muhammad Ammar Zafar

Advisor: Prof. Dr. Andreas Kassler

České Budějovice 2024

Bibliographical details:
Zafar, M.A., 2024: PV Power Forecasting using Distributed Machine Learning for Smart Energy Grid (An extension to Federated Learning). Mgr. Thesis, in English. – 97 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Annotation:
This thesis investigates the application of federated learning and tree-based models, such as LightGBM and Catboost, in Photovoltaic (PV) power forecasting. Addressing challenges in accuracy, uncertainty, and scalability, the study designs a robust federated learning architecture tailored for tree-based forecasting models. The novel aggregation strategy efficiently combines updates from multiple nodes, enhancing forecast accuracy.

Declaration:
I declare that I am the author of this qualification thesis and that in writing it I have used the sources and literature displayed in the list of used sources only.

Place, date.
Student's signature

Ceske Budejovice, 26 January 2024 

Abstract

The intermittent nature of renewable energy resources like Photovoltaic (PV) poses a challenge to energy forecasting, grid balancing, and scheduling. Accurate and reliable forecasts of energy generation and consumption are needed to optimize the performance of the smart energy grid. Machine Learning (ML) has shown great potential in addressing these challenges. However, existing approaches have limitations in terms of accuracy, uncertainty, and scalability. Federated learning is a technique that allows machine learning models to be trained and applied on distributed data sources without the need to centralize the data. In the context of PV production and consumption forecasting models, federated learning can be used to train and apply these models on data from different prosumers, such as different households or buildings, while preserving the privacy of the data owners. This thesis investigates PV power forecasting using federated learning and tree-based models, such as LightGBM and Catboost. The importance of accurate PV power forecasting for renewable energy integration is highlighted, along with the challenges for data aggregation for tree-based models. The research objectives are outlined: to design and create a robust federated learning architecture specifically tailored for tree-based forecasting models, such as LightGBM and Catboost, to design and implement a novel aggregation strategy that efficiently combines updates from these models obtained from multiple nodes in the federated learning architecture, and to evaluate those models in a distributed test-bed on data-set coming from the smart grid community and compare the performance in terms of forecast accuracy to non-distributed approaches. The evaluation shows that the developed approach reduces the Mean Absolute Error (MAE) for Catboost Production by 88.1%, LightGBM Consumption by 14.93%, and Catboost Consumption by 79.5%. For LightGBM Production it remained almost same. After using federated learning, Mean Quantile Loss (MQL) and Mean Prediction Interval Range (MPIR) also significantly decreased.

Keywords

Machine Learning (ML), Photovoltaic(PV), A Friendly Federated Learning Framework (FLWR), Mean Prediction Interval Range (MPIR), and Mean Quantile Loss (MQL), Random Forests (RF), LightGBM, CatBoost

Acknowledgements

I am immensely grateful to all those who have contributed to the completion of this thesis. Without their unwavering support, valuable insights, and encouragement, this journey would not have been possible.

First and foremost, I extend my deepest gratitude to my thesis advisor, Prof. Dr. Andreas Kassler, for his constant guidance and mentorship. His expertise and dedication have been instrumental in shaping the direction of this research and improving the quality of my work. I am indebted to him for his patience, valuable feedback, and belief in my abilities.

I also extend my heartfelt appreciation to the faculty members of the M.Sc Artificial Intelligence and Data Science for their encouragement, stimulating discussions, and constructive criticism throughout my academic journey. Their commitment to excellence in teaching and research has been a constant source of inspiration.

My sincere thanks go to the Mr. Phil Aupke who generously dedicated their time and shared their insights, making this research possible. Their contributions are invaluable and have enriched the findings of this thesis.

I am thankful of my family and friends' constant encouragement and patience throughout this trying time. My endurance has been motivated by their support, love, and confidence in me.

Finally, I want to extend my sincere gratitude to all the researchers, writers, and scholars whose work served as the basis for my thesis. Our understanding remains shaped and inspired by their contributions to the discipline.

Contents

Chapter 1

Introduction

The use of renewable energy sources, especially photovoltaic (PV) systems, has grown significantly in recent years as a crucial element of sustainable energy policies. With its eco-friendly qualities and ability to lower greenhouse gas emissions, photovoltaic power generating has become a prospective replacement for conventional fossil fuel-based electricity generation. However, the intermittent nature of solar energy poses considerable difficulties for grid managers and energy planners in preserving grid stability and guaranteeing resource efficiency.

One critical aspect in effectively integrating PV systems into the power grid is accurate and reliable forecasting of PV power production and consumption. Accurate forecasts enable grid operators to anticipate fluctuations in power input and output, thereby optimizing grid management, minimizing imbalances, and making informed decisions about backup power sources.

Moreover, while accurate PV power forecasting is essential for efficient grid operations, there is a growing concern about preserving data privacy, especially with the increasing use of smart meters and data-intensive monitoring systems. Energy consumption data contains sensitive information about households and businesses, and its unauthorized access or misuse could compromise individuals' privacy and security.

To address the challenges of both accurate PV power forecasting and data privacy preservation, novel approaches are required that can leverage the power of data while ensuring privacy protection. One promising avenue is the application of federated learning that is decentralized machine learning framework that enables multiple parties to train models collaboratively while keeping their data locally stored and without sharing raw data. Federated learning can enable effective model training across numerous distributed sources without compromising the privacy of individual data contributors, which has the potential to resolve the conflicting goals of accurate forecasting and data privacy.

1.1 Background

In the realm of PV power forecasting, machine learning (ML) based approaches have gained significant traction. Advanced techniques, such as deep learning models like LSTM (Long Short-Term Memory) networks and CNN (Convolutional Neural Networks), have been actively explored by researchers. These deep learning models have demonstrated promising results in capturing complex temporal patterns inherent in PV power production and consumption data, leading to improved forecast accuracy. The ability to learn from historical patterns and make accurate predictions in dynamic environments makes these models particularly suitable for PV power forecasting.

However, Forest-based models like LightGBM [23] and Catboost [40]have shown considerable success in Multi-Variate Time Series Forecasting [20]. These ensemble methods leverage decision trees to capture interactions between variables and provide robust predictions. In the context of PV power forecasting, the application of LightGBM and Catboost has shown significant potential for capturing complex relationships between meteorological and operational factors, leading to accurate and reliable predictions of PV power generation.

The combination of federated learning with advanced machine learning techniques, such as tree-based models, presents a promising avenue for enhancing PV power forecasting [54]. By leveraging the strengths of federated learning's privacy-preserving capabilities and the predictive power of these advanced ML models, the accuracy and efficiency of PV power forecasting can be further improved. The integration of federated learning with tree-based models has the potential to revolutionize the field of PV power forecasting, contributing to more efficient renewable energy integration and sustainable energy management.

1.2 Problem Description

In this thesis, the main challenge is of developing a PV power forecasting model using federated learning, with a specific emphasis on forest-based regression models such as LightGBM and CatBoost. The main hurdle lies in creating a framework that facilitates efficient aggregation of updates from these forest models, all while preserving the key advantages of federated learning, namely privacy and data decentralization.

One of the core issues addressed is that existing federated learning frameworks, like FLWR (A Friendly Federated Learning Framework), lack built-in support for aggregating weights specifically tailored to forest-based models. This limitation poses a significant obstacle in effectively employing forest-based models within a federated learning setup for PV power forecasting.

Another aspect of the challenge is related to the nature of forest-based models themselves. These models often consist of multiple trees, and each tree is typically trained independently. As such, accommodating multiple models training per client within the federated learning framework becomes a challenging task that requires innovative solutions.

The central focus of this thesis is to devise a novel aggregation mechanism that can overcome these challenges effectively. This mechanism should enable seamless integration of forest-based models' updates from multiple nodes while retaining data privacy and ensuring efficient communication among the participating devices. By addressing these critical issues, the research aims to unlock the potential of federated learning for PV power forecasting, leveraging the strengths of forest-based models in capturing complex temporal patterns inherent in solar energy generation data.

The outcome of this research is expected to contribute to the field of federated learning and its applicability to PV power forecasting. By devising a robust aggregation approach tailored to forest-based models, the thesis aims to broaden the scope of federated learning applications, providing privacy-preserving, accurate, and efficient forecasting solutions for the renewable energy sector. The research outcomes have the potential to advance the state-of-the-art in decentralized machine learning approaches, impacting various domains beyond PV power forecasting.

1.3 Thesis Objective

The first objective of this thesis is to design and create a robust federated learning architecture specifically tailored for multivariate time series forecasting, with a focus on PV power prediction. This architecture will enable distributed devices or nodes to collaborate and train predictive models collectively without sharing raw data. The goal is to ensure data privacy while harnessing the collective knowledge of all nodes to enhance forecasting accuracy.

The second objective is to implement and evaluate tree-based regression models, such as LightGBM and CatBoost, within the federated learning framework. These ensemble models have shown promise in handling complex interactions within time series data and are well-suited for PV power forecasting. The goal is to assess the performance of these models in terms of accuracy, robustness, and efficiency within the federated setting.

The third objective is to design and implement a novel aggregation strategy that efficiently combines updates from tree-based regression models obtained from multiple nodes in the federated learning architecture. The objective is to develop an aggregation mechanism that preserves data privacy, minimizes communication overhead, and ensures accurate and reliable PV power forecasts.

The fourth objective is to evaluate those models in a distributed test-bed on data-set coming from the smart grid community and compare the performance in terms of forecast accuracy to non-distributed approaches.

1.4 Research Questions

1. How does Federated learning impact forecast accuracy of Multi-Variate Time Series Forecasting in the context of Renewable Energy Systems?

2. How can Federated Learning be implemented on tree-based models?

1.5 Methodology

This thesis follows a traditional methodology for a thesis that aims to develop forecasting models using Machine Learning. Performing federated inference for PV production and consumption models requires careful consideration of data preparation, model selection, federated learning setup, federated inference, evaluation, and deployment. By following these steps, it is possible to train accurate and robust PV production and consumption forecasting models on distributed data sources while preserving the privacy of the data owners.

Firstly the data collected by [3] from Swedish households having PV installations and battery storage will be used. The data will include the variety of features, such as the power generation of PV, household consumption, weather data, and time of day. The data will be preprocessed to remove outliers, missing values, and redundant features. The data will be scaled to normalize the range of values.

Secondly suitable model architectures for the PV production and consumption forecasting task are to be chosen. The models should be designed to handle the variability in the data from different sources and to be robust to changes in the data distribution over time. Considering all this proposed models will be[15] GBQR – Catboost, GBQR - LightGBM

Thirdly, Set up a federated learning framework consisting of a central server and multiple clients, each of which has its own local data. The server sends the initial model parameters to each client, and the clients train the model on their local data.

Lastly, the performance of the proposed framework will be evaluated using various metrics like Mean Absolute Error (MAE), Mean Prediction Interval Range (MPIR), and Mean Quantile Loss (MQL). The results will be compared with existing ML techniques.

1.6 Outline

Chapter 2 provides a comprehensive background on time series forecasting and its application in PV power forecasting. Traditional forecasting methods for multi-variate time series are explored, along with the unique challenges faced when predicting PV power production and consumption. The chapter also delves into federated and distributed learning, presenting an overview of federated learning, and its advantages over traditional centralized machine learning. Privacy and security concerns in federated learning are discussed, along with emerging research trends in this field.

Chapter 3 outlines the research design adopted for this study. The data collection and preprocessing procedures for PV power data are detailed, focusing on handling missing data and outliers. The design of the federated learning architecture for PV power forecasting is presented, elucidating the selection of participating nodes and their respective roles. Algorithms for aggregation approaches for FedAvg are presented

Chapter 4 explains the challenges associated with aggregating forest-based model updates in federated learning and implementations. The proposed novel aggregation mechanism is introduced, offering an efficient means of combining model updates from various nodes while ensuring data privacy. The chapter presents the technical details of the aggregation approach and how it addresses the unique requirements of federated PV power forecasting.

Chapter 5 presents the results of the research. A comprehensive performance comparison is conducted, evaluating the forecasting accuracy of federated tree-based models against traditional centralized models. The predictive capabilities of LightGBM and Catboost in the federated learning setting are compared. In this chapter, the implications of the obtained results are discussed in-depth. The forecasting performance of federated forest models is interpreted, highlighting their strengths and limitations. The impact of the novel aggregation approach on forecast accuracy and efficiency is analyzed.

Chapter 6 presents a concise summary of the key findings and contributions of the thesis. The research objectives are revisited, and their fulfillment is affirmed. The significance of the novel aggregation approach in the context of federated PV power forecasting is emphasized, highlighting its potential impact on the field of renewable energy forecasting.

Chapter 2

Background and Literature Review

In this chapter the background and related work is introduced that is necessary for understanding the concepts of this thesis. This will range from research on time series forecasting techniques to underlying concepts about federated learning.

2.1 Background

This section provides detailed technical background of all the concepts used in this thesis, such as time series forecasting, evaluation metrics, tree-based models, and federated learning.

2.1.1 Time Series Forecasting and Evaluation Metrics for Interval Ranges

Time series forecasting and multivariate time series forecasting [20] are key tasks in predictive analytics and machine learning, and they play an important role in gaining valuable insights and generating informed decisions from temporal data. Time series forecasting is concerned with predicting the future values of a particular variable over time by examining its previous measurements. This analytical method is very useful when dealing with data that has a distinct temporal pattern, such as stock prices in financial markets, temperature measurements in climate research, or monthly sales numbers in retail analytics. In time series forecasting, several well-established approaches are used, each customized to particular data qualities and trends. Classical techniques, such as Autoregressive Integrated Moving Average (ARIMA), are appropriate for stable time series, whereas sophisticated models, such as Long Short-Term Memory (LSTM) neural networks, are well-suited for capturing long-term relationships in sequences.

Multivariate time series forecasting, on the other hand, broadens the scope to include

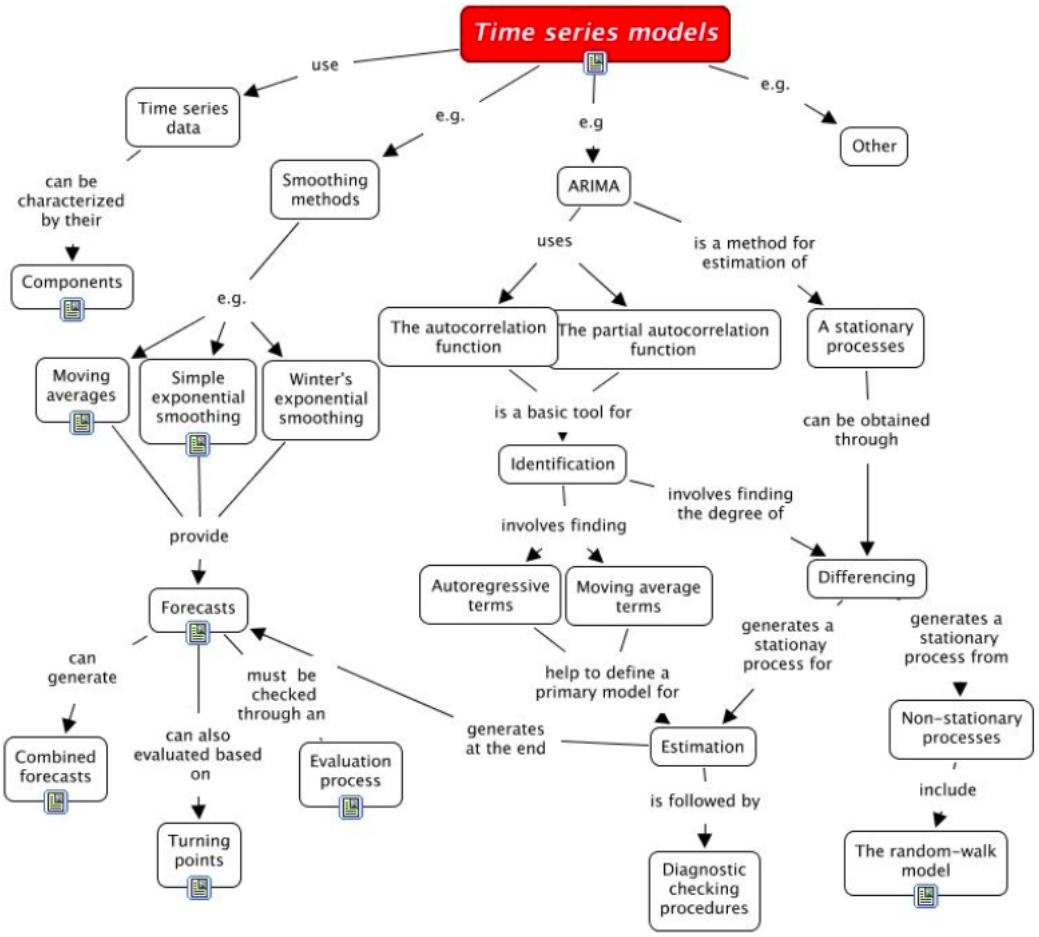

Figure 2.1.1: Time series models [47]

the prediction of future values for many connected variables that change over time. The behavior of these variables is interrelated in many real-world circumstances, and their interactions can have a major impact on each other's dynamics. In an e-commerce scenario, for example, projecting the sales of multiple products at the same time becomes critical, taking into account not just their past sales patterns but also the influence of marketing efforts, consumer preferences, and external circumstances such as holidays and special events. This interdependence presents modeling challenges and complications since it necessitates capturing detailed linkages and dependencies between variables.

To address these challenges, numerous specialized algorithms for multivariate time series forecasting have been developed. Vector Autoregression (VAR) models, for example, use lagged values from many time steps to model variable interdependence. State space models provide a versatile framework for modeling temporal dynamics and adding external variables. Deep learning methods, such as Multivariate Long Short-Term Memory (MLSTM) networks, have showed significant promise in dealing with complex multivariate time series data, efficiently capturing both short-term and long-term dependencies in recent years.

Both time series forecasting and multivariate time series forecasting are essential tools for comprehending and predicting temporal data patterns. This expansion enables decision-makers to get deeper insights, find complicated relationships, and generate more accurate forecasts in a variety of complex scenarios ranging from finance and economics to healthcare and beyond. The significance of these forecasting techniques will continue to be at the forefront of data science and predictive modeling as the value of temporal data analysis develops in an increasingly data-driven world.

Mean Absolute Error (MAE)

Photovoltaic (PV) power forecasting is a crucial aspect of efficient energy management, enabling grid operators and consumers to optimize energy consumption from renewable sources. It is becoming increasingly important to accurately forecast the amount of power generated by photovoltaic panels (PV), particularly solar power, in order to maintain the stability of the power grid, reduce the amount of energy that is wasted, and facilitate the smooth incorporation of renewable energy sources into the power grid. The use of machine learning techniques has emerged as a powerful tool for improving the accuracy and reliability of PV power forecasting. This has enabled more precise and timely predictions to be made, even in the face of complex and dynamic weather conditions.

The Mean Absolute Error, or MAE, is a metric that is frequently utilized for the purpose of evaluating the efficacy of various forecasting models. The MAE is a measure of the accuracy of a model's prediction that is both comprehensive and easy to understand. This measure enables direct comparisons to be made between various methods of forecasting.

The MAE quantifies the extent to which the model's predictions diverge from the actual observations by calculating the average absolute difference between the predicted values and the true values. A smaller MAE value indicates a higher level of accuracy, which indicates that the model's predictions are closer to the real world situation.

Within the framework of the study, the MAE was selected to serve as the primary evaluation metric for the purpose of assessing the efficacy of a number of different PV power forecasting models. The MAE served as a reliable and objective indicator of prediction accuracy, which made it easier to make comparisons between the various methods of forecasting in order to find the model that was the most successful.

Finding the average absolute difference between the values that were predicted and those that were actually observed across all of the test samples was required in order to compute the MAE. The MAE can be expressed in mathematical terms as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.1}$$

Here, n denotes the number of test samples, y_i represents the true value of the target variable for the i^{th} sample, and $\hat{y}_i$ is the predicted value of the target variable for the i^{th} sample.

The mean absolute error (MAE) is a straightforward and easily interpretable metric that provides insights into the overall accuracy of the model's predictions. The mean absolute error (MAE) is a measure that is used to determine how far, on average, the model's predictions stray from the actual values. This is done by evaluating the magnitude of the absolute errors.

Using the MAE as the evaluation metric enabled quantitative comparisons of the performance of various forecasting models, including the federated learning approach, with conventional methods such as LightGBM and Catboost. These comparisons were made possible by the fact that the MAE was used as the evaluation metric. The superiority of the federated learning approach as a method for producing more accurate predictions for PV power forecasting was demonstrated by the observed reductions in MAE associated with that method.

In addition to this, the MAE delivered valuable information that helped identify areas of the forecasting models that could use some improvement. For instance, if the MAE was higher for certain weather conditions or time periods, this indicated that there may be potential difficulties in capturing the variations in photovoltaic power generation or consumption under those conditions. Having an understanding of such patterns made it easier to fine-tune the model and improve its overall performance.

In conclusion, the Mean Absolute Error (MAE) was able to function as an effective

and objective evaluation metric for the PV power forecasting models. The adoption of this approach allowed for the validation of the federated learning approach's superiority over conventional methods and provided valuable insights for further research and improvements in sustainable energy management. Its adoption also allowed for the validation of the superiority of federated learning over conventional methods. In spite of the continued rise in popularity of renewable sources of energy, PV power forecasting that is both accurate and reliable will continue to play a critical role in maximizing the benefits that can be derived from these sources and fostering a more sustainable energy future. The application of MAE in this study exemplifies the significance of employing reliable evaluation metrics in advancing the field of photovoltaic power forecasting and improving energy management practices for the purpose of making the world a greener and more sustainable place.

Mean Quantile Loss (MQL)

When evaluating quantile regression models, particularly those used in PV power forecasting, the Mean Quantile Loss (MQL) is an important metric that is used. Quantile regression, as opposed to traditional regression, which focuses on predicting the mean of the target variable, aims to predict different quantiles of the target variable, providing a more comprehensive view of the data distribution. Traditional regression is focused on predicting the mean of the target variable.

$$\text{MQL} = \frac{1}{N} \sum_{i=1}^{N} L_{\Sigma}(y_i, \hat{y}_i) \tag{2.2}$$

Here, N denotes the number of test samples, y_i represents the true value of the target variable for the i^{th} sample, and $\hat{y}_i$ is the predicted value of the target variable for the i^{th} sample.

The MQL is a measurement that determines how accurate the quantile predictions are, and it reflects the degree to which the model's estimated quantiles coincide with the actual quantiles of the target variable. It is computed as the average absolute difference between the predicted quantiles and the true quantiles across a set of test samples. This difference is taken across all of the samples. If the MQL is lower, this indicates that the model's estimated quantiles are closer to the true values. This, in turn, indicates that the prediction accuracy is higher.

Because it enables a more in-depth comprehension of the uncertainty connected to the predictions, the MQL is particularly useful in PV power forecasting. We are able to gain insights into the model's ability to estimate the variability of PV power production and consumption by evaluating the performance of the model at various quantiles.

The Mean Quantile Loss is a crucial metric for assessing the accuracy of quantile regression

models in PV power forecasting. These models are used to predict the amount of power that will be generated from photovoltaic cells. Utilizing it yields useful information about the performance of the model at various quantiles, which enables a more in-depth evaluation of prediction intervals and uncertainty estimation. We are able to improve the reliability of PV power predictions and enhance sustainable energy management practices if we incorporate the MQL into the process of evaluation.

Mean Prediction Interval Range (MPIR)

When evaluating prediction intervals in forecasting models, including PV power forecasting, the Mean Prediction Interval Range (MPIR) is an essential metric to use. The use of prediction intervals allows one to get an idea of the degree of uncertainty that is associated with the model's predictions by providing a range of values within which actual observations are likely to fall with a certain probability.

The mean width of the prediction intervals across a group of test samples is what the MPIR is designed to measure. If the MPIR is lower, this indicates that the prediction intervals are more precise and narrower, which suggests that the uncertainty estimation is improved.

$$\text{MP IR} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_{u,i} - \hat{y}_{l,i}) \tag{2.3}$$

Here, N denotes the number of test samples, y_i represents the true value of the target variable for the i^{th} sample, and $\hat{y}_i$ is the predicted value of the target variable for the i^{th} sample.

When it comes to the forecasting of PV power, the MPIR is an extremely important component in determining the accuracy of the prediction intervals. Accurate prediction intervals provide grid operators and consumers with valuable information, which aids in the making of informed decisions regarding energy consumption and the management of the grid.

It is possible to conduct a thorough analysis of the performance of the forecasting model by including the MPIR in the evaluation process alongside other metrics such as the Mean Absolute Error (MAE) and the Mean Quantile Loss (MQL). This will allow for the achievement of a comprehensive evaluation. These metrics are improved with the addition of the MPIR, which supplements them by providing insights into the model's ability to estimate uncertainty and overall reliability in PV power predictions.

When it comes to assessing the accuracy of prediction intervals in PV power forecasting, the Mean Prediction Interval Range is an important metric to take into consideration. Its application contributes to more informed decision-making in energy management

practices by providing valuable information about the uncertainty associated with the model's predictions. We are able to improve the precision and dependability of PV power forecasting if we take into account the MPIR during the evaluation process. This will help to contribute to energy utilization that is more efficient and sustainable.

2.1.2 Tree-Based Models

Tree-based models are a popular class of machine learning algorithms due to their ease of use, interpretability, and versatility in a variety of tasks [49]. These models are built on the decision tree concept, in which data is recursively partitioned into subsets based on specific feature values. Each internal node in the tree represents a decision based on a specific feature, while each leaf node represents a prediction or outcome. The Random Forest, an ensemble learning technique that combines multiple decision trees to make predictions, is one of the most popular tree-based models. Random Forests are well-known for their robustness, resistance to overfitting, and ability to handle large, high-dimensional datasets. Gradient Boosting is another powerful ensemble technique that builds decision trees sequentially to correct errors from previous trees. Gradient Boosting models, such as XGBoost and LightGBM, are widely used in machine learning competitions and real-world applications due to their exceptional performance.

The interpretability of tree-based models is one of their main advantages [49]. Decision trees provide an easy-to-understand visual representation of how the model makes decisions. The reduction in impurity brought by each feature during tree construction can easily be used to calculate feature importance. This information on feature importance is useful for feature selection, understanding model behavior, and extracting actionable insights from data.

Tree-based models [1] are also good at dealing with missing data. When they encounter missing values during predictions, they can simply follow the available branches. Furthermore, tree-based models are less sensitive to outliers than linear models. During the recursive partitioning process, outliers are isolated, reducing their impact on the overall model.

Regularization techniques can be used on tree-based models to prevent overfitting. This includes limiting the maximum depth of trees, requiring a minimum number of samples to split a node, and pruning to remove nodes that contribute little to the model. Regularization improves the generalization performance of the model and keeps it from becoming overly complex.

Tree-based models have a wide range of applications in a variety of domains [1]. They are used in classification tasks such as customer churn prediction, sentiment analysis, and spam detection. Tree-based models are used in regression tasks to predict house prices, sales forecasts, and demand forecasting. Anomaly detection for network traffic,

fraud detection, and fault diagnosis are also useful to them. Tree-based models are also important in developing personalized recommendation systems for e-commerce platforms and content recommendations.

Discrete Trees

A discrete tree is a finite structure with a fixed number of vertices (nodes) and edges (connections between nodes) [1]. The important feature of such trees is that each pair of vertices is connected by a separate path of distinct edges. The vertices in this abstract representation are not allocated precise places in space. However, by selecting a root vertex and assigning it to the origin (0, 0,..., 0), the tree can be embedded in d-dimensional space. The additional vertices are then transferred to points in d-space using either lattice coordinates or unit length vectors. During the embedding process, several vertices of the abstract tree may be transported to the same position in d-space.

Various models of random n-vertex abstract trees can be investigated to inject randomness into both the tree structure and the embedding. Combinatorial models are based on the assumption that all n-vertex trees are equally likely. However, multiple rules and constraints on allowed degrees of vertices may exist, resulting in many versions of this paradigm. Conditioned branching processes, on the other hand, include a population process that begins with one person and ends with a random number of children (mean 1, variance between 0 and 1). When this technique is applied to a total population size of n, the resulting family tree has a random n-vertex structure. The combinatorial aggregation model is another way, in which start with n vertices and no edges and then iteratively add random edges that do not form a cycle until a random n-vertex tree is received.

Although these models appear to be different at first glance, some combination models (with specific conventions) are known to be comparable to conditioned branching process models (with specific offspring distributions). Furthermore, substantial evidence, supported by research and interchange-of-limits arguments, reveals that the combinatorial aggregation model corresponds with the other models as n approaches infinity.

These random n-vertex abstract trees have applications in mathematics, computer science, and biology. They shed insights into population dynamics, network structure, and branching process behavior. Understanding the features of these discrete tree models is critical for many applications because it allows us to understand the statistical behavior of complex systems and guides the development of efficient algorithms to analyze large-scale networks and hierarchical data structures.

Continuous Trees

Continuous trees are a fundamental machine learning model used to predict continuous numeric values. They are also known as regression trees or decision trees for regression. Continuous trees are specifically built to handle regression tasks, whereas typical classification trees are meant for discrete class label prediction. The purpose of regression is to forecast a continuous output, like as the price of a house based on its qualities or the sales of a product based on historical data.

A continuous tree's fundamental structure is similar to that of a classification tree, with a hierarchical arrangement of nodes expressing feature-based judgements [1]. A specific attribute is examined at each internal node of the tree to discover the optimum split that splits the data into subgroups with similar target values. The splitting procedure is repeated until leaf nodes are reached, each of which corresponds to a specific region of the input feature space.The ability of continuous trees to provide real-valued predictions is what sets them apart. Once the tree is built, each leaf node is assigned a constant value, which is often the average of the target values in that region. The prediction for unseen data is made by traversing the tree, beginning at the root and following the path given by the feature criteria until it reaches a leaf node. The forecast for the input data is the value linked with the leaf node.

The interpretability of continuous trees is one of its major features. The resulting tree structure may be viewed and comprehended, revealing important details about how the model produces predictions. This transparency is especially useful in instances where interpretability is critical for decision-making and comprehending the underlying elements. Continuous trees also have the capacity to capture intricate non-linear interactions between features and the target variable. Continuous trees can successfully model sophisticated data patterns that linear models may not capture by recursively partitioning the feature space based on the most informative splits.Despite their advantages, continuous trees have certain limits. They are prone to overfitting, particularly when the tree grows too deep and complicated. Techniques like as regularization, pruning, and setting a limit depth for the tree are frequently used to address issue.

Ensemble approaches such as Random Forest and gradient boosting frameworks like as LightGBM have gained favor in recent years for enhancing the performance of continuous trees. LightGBM, in particular, is well-known for its efficiency and speed in constructing large ensembles of continuous trees, making it ideal for big data and real-time applications.

2.1.3 Gradient Boosting Machine (GBM)

Gradient Boosting Machines (GBMs) are a potent class of machine learning algorithms that fall under the umbrella of ensemble learning. These models excel in both regression and classification tasks, leveraging the principle of boosting to sequentially train a series of weak learners, where each learner focuses on rectifying the errors of its predecessor. This cumulative refinement process ultimately produces a robust final model with superior predictive capabilities. The core of GBMs lies in minimizing a chosen loss function, which quantifies the disparity between predicted and actual target values. Through an iterative approach, the algorithm calculates the negative gradient of the loss function concerning the current model's predictions and updates the model in the direction of steepest decrease. By introducing a learning rate, users can modulate the step size of parameter adjustments, balancing convergence speed with robustness. GBMs rely on a collection of weak learners, often shallow decision trees or stumps, which, when combined, contribute significantly to the ensemble's overall strength. Regularization techniques, such as constraining tree complexity or implementing early stopping, prevent overfitting and fine-tune model performance. Notably, variants like XGBoost, LightGBM, and CatBoost have evolved from the basic GBM algorithm, incorporating optimizations and efficiencies to cater to specific challenges, solidifying GBMs as a favored choice for intricate data relationships and accurate predictions in diverse machine learning applications.

Input:

- Training dataset: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Number of iterations: M

Initialize:

- Initial prediction: $F_0(x) =$ initial_prediction_value

For $m = 1$ **to** M:

1. Compute negative gradient: $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$
2. Fit a weak learner (e.g., decision tree) to the negative gradient: $h_m(x) =$ FitWeakLearner(X, r_m)
3. Update the model: $F_m(x) = F_{m-1}(x) + \lambda \cdot h_m(x)$

Output:

- Final ensemble model: $F_M(x)$

Gradient Boosting Quantile Regressions

Gradient Boosting Quantile Regression (GBQR) is a powerful statistical technique used for estimating conditional quantiles of a response variable. Unlike traditional regression methods that focus on estimating the conditional mean, quantile regression provides a more comprehensive picture of the data distribution by estimating various quantiles. This approach is particularly useful when dealing with data that is asymmetric, heteroscedastic, or contains outliers.

The GBQR algorithm is an extension of the gradient boosting machine (GBM) framework, which is an ensemble learning technique that combines multiple weak learners (usually decision trees) to create a strong predictive model. In GBQR, the objective is to minimize a specific loss function known as the pinball loss, which quantifies the difference between the actual response and the estimated quantile.

The process of GBQR involves iteratively fitting decision trees to the residuals of the previous iteration. At each step, the decision tree is trained to approximate the negative gradient of the pinball loss function, which ensures that the algorithm converges towards more accurate quantile estimates. The hyperparameters, such as the learning rate, number of trees, and depth of the trees, play a crucial role in controlling the model's performance and preventing overfitting.

Input:

- Training dataset: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Number of iterations: M
- Quantile levels: $\tau_1, \tau_2, \ldots, \tau_k$

Initialize:

- Initial predictions: $F_{0,\tau}(x) = \text{initial\_prediction\_value}$

For $m = 1$ **to** M:

1. For each quantile level τ, compute residuals:

$$r_{i,\tau} = \begin{cases} y_i - F_{m-1,\tau}(x_i), & \text{if } y_i > F_{m-1,\tau}(x_i) \\ 0, & \text{if } y_i \leq F_{m-1,\tau}(x_i) \end{cases}$$

2. Fit a weak learner to the residuals: $h_{m,\tau}(x) = \text{FitWeakLearner}(X, r_\tau)$
3. Update the model for each quantile level:

$$F_{m,\tau}(x) = F_{m-1,\tau}(x) + \lambda \cdot h_{m,\tau}(x)$$

Output:

- Final ensemble quantile regression models: $F_{M,\tau}(x)$ for $\tau = \tau_1, \tau_2, \ldots, \tau_k$

LightGBM

LightGBM [27] stands out as an advanced open-source gradient boosting framework developed by Microsoft. Its technical foundations lie in innovative strategies that optimize and expedite the gradient boosting algorithm for machine learning tasks. One of its key contributions is the introduction of histogram-based learning, which replaces conventional data pre-sorting with histogram binning. This approach enhances memory efficiency and accelerates training by facilitating efficient gradient computation and node splitting during tree construction. A standout feature of LightGBM is its leaf-wise tree

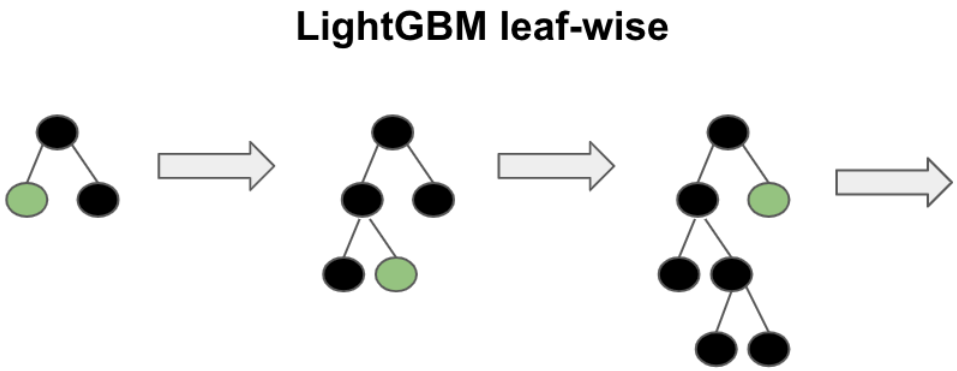

Figure 2.1.2: LightGBM tree [46]

growth strategy as shown in 2.1.2, which strategically expands the tree by selecting the leaf node with maximum loss reduction for expansion. This strategy often leads to deeper and more accurate trees while preventing overfitting through built-in regularization techniques. Additionally, the framework implements Gradient-based One-Side Sampling (GOSS), a technique that intelligently selects a subset of data points based on gradient information, effectively speeding up training while mitigating overfitting risks. LightGBM also supports exclusive feature bundling for categorical features, GPU acceleration for training efficiency, and early stopping for optimal model convergence. The flexibility to define custom objective functions and metrics, as well as its compatibility with distributed

computing environments, further underline LightGBM's technical versatility. Through its active community of contributors and continual development, LightGBM remains at the forefront of machine learning technology [23], providing researchers and practitioners with an efficient and accurate tool for tackling challenging tasks, particularly those involving large and intricate datasets.

Input:

- Training dataset: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Number of iterations: M

Initialize:

- Initial predictions: $F_0(x) =$ initial_prediction_value

For $m = 1$ **to** M:

1. Compute gradients and Hessians:
   $g_i = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$, $h_i = \left[\frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2}\right]_{F(x)=F_{m-1}(x)}$
2. Construct histograms for features based on the gradients and Hessians
3. Find the best splits for each feature's histogram
4. Create a new tree structure using the best splits
5. Update leaf values using a gradient-based optimization (e.g., Newton's method)

Output:

- Final ensemble model: $F_M(x)$

Catboost

Catboost stands as a significant advancement in gradient boosting frameworks with its distinctive focus on effectively handling categorical features [40]. This framework's technical foundation is rooted in its innovative "ordered boosting" technique, which seamlessly integrates categorical variables into the boosting process. Unlike conventional approaches that require one-hot encoding, Catboost employs a permutation-based strategy, allowing it to naturally manage categorical attributes while conserving memory and curtailing overfitting. The framework's ordered boosting method sorts data points based on categorical feature values, yielding balanced splits that contribute to superior predictive accuracy during tree construction. Furthermore, Catboost adopts symmetric tree structures to ensure consistent data distribution across leaf nodes, thus enhancing the model's generalization capabilities. Implementing Gradient-based One-Side Sampling (GOSS) amplifies training speed by selectively emphasizing data points with more

substantial gradients, simultaneously mitigating overfitting concerns. This framework also offers an array of regularization techniques, encompassing L2 regularization and diverse loss functions tailored to various tasks. Additionally, Catboost is GPU-accelerated, boasts robust handling of missing values, supports early stopping, and facilitates feature scaling, all contributing to its reputation for efficiency and practicality. As a memory-efficient solution with integrated hyperparameter optimization tools, Catboost emerges as a versatile and powerful gradient boosting framework, particularly well-suited for datasets that encompass a mix of data types, and remains a favored choice across machine learning applications.

Input:

- Training dataset: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Number of iterations: M
- Learning rate: η
- Categorical features: categorical_features

Initialize:

- Initial predictions: $F_0(x) =$ initial_prediction_value

For $m = 1$ **to** M:

1. Compute gradients and Hessians:

   $g_i = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$, $h_i = \left[\frac{\partial^2 L(y_i, F(x_i))}{\partial F(x_i)^2}\right]_{F(x)=F_{m-1}(x)}$

2. Construct histograms for features (including categorical features) based on the gradients and Hessians
3. Find the best splits for each feature's histogram
4. Create a new tree structure using the best splits
5. Update leaf values using a gradient-based optimization (e.g., Newton's method)

Output:

- Final ensemble model: $F_M(x)$

2.1.4 Distributed Learning Approaches

When considering distribution, there are two main ways of dividing the problem across multiple machines: parallelizing the data or parallelizing the model [50]. It is also possible to employ both methods simultaneously.

In the Data-Parallel approach, the data is divided as many times as there are worker nodes in the system. Each worker node applies the same algorithm to different datasets. All worker nodes have access to the same model, either through centralization or replication, resulting in a single coherent output. This technique is applicable to machine learning algorithms that assume independent and identical distribution (i.i.d.) of data samples, which includes most ML algorithms.

In the Model-Parallel approach, each worker node processes exact copies of the entire datasets but operates on different parts of the model. The model is formed by aggregating all the model parts. However, the model-parallel approach cannot be automatically applied to every machine learning algorithm because the model parameters generally cannot be divided. In the next section we expand upon this concept by introducing federated learning that uses model-parallel approach.

LightGBM Distributed

LightGBM's distributed mode is a sophisticated framework designed to harness the power of distributed computing environments for efficient and rapid training on massive datasets [27]. At its core, the distributed mode capitalizes on the strengths of data and feature parallelism, asynchronous communication, and optimized histogram-based learning.

When distributed across multiple worker nodes, the training dataset undergoes data partitioning, with each worker taking responsibility for a distinct subset of data. This data parallelism not only enables efficient processing of extensive datasets but also facilitates load balancing to prevent performance bottlenecks.A central technical innovation in LightGBM's distributed mode revolves around its histogram approximation technique. Each worker independently constructs histograms tailored to its local data subset, capturing essential statistical insights required for informed node splits during tree construction. Periodic merging of these local histograms ensures a comprehensive understanding of the global dataset distribution, a crucial aspect of accurate modeling.

The power of asynchronous communication amplifies training speed and accelerates convergence. Workers independently update their models and gradients while intermittently exchanging essential information with a central coordinator. This approach minimizes idle time and fosters continual progress, even while communication is underway. Fault tolerance mechanisms are also integrated into the framework. Should a worker node experience failure, the training process can recover and continue without substantial interruptions. The global synchronization points, such as histogram merging, are designed to gracefully handle minor worker failures.

In addition to its distributed capabilities, LightGBM's asynchronous communication and

optimization techniques distinguish it. The integration of data and feature parallelism, along with the targeted use of asynchronous communication, drives its effectiveness in handling enormous datasets and complex models. LightGBM's distributed mode showcases a remarkable synergy between theoretical innovation and practical application, contributing to its reputation as a high-performance gradient boosting framework in distributed computing environments.

2.1.5 Federated Learning

Federated learning (FL), as a distributed machine learning paradigm, presents an innovative and privacy-preserving approach to PV power forecasting [12]. In this decentralized framework, multiple devices or edge nodes within a smart energy grid collaboratively contribute to training a global forecasting model without sharing their raw data. Instead, each node performs local model training using its own data and only communicates the model updates to a central server or aggregator. This unique approach ensures that the sensitive PV power data remains decentralized and private, as the raw data never leaves the individual devices or nodes.

By leveraging federated learning in PV power forecasting, several significant benefits are achieved. Firstly, data privacy and security are maintained since the raw data is not shared, minimizing the risk of data breaches or privacy violations. Secondly, federated learning enables the utilization of collective knowledge from various nodes, ensuring that the global forecasting model is enriched with insights from diverse data sources.

This collective intelligence enhances the accuracy and generalization capability of the forecasting model, particularly in dynamically changing environments. The application of federated learning to PV power forecasting facilitates accurate predictions without compromising the privacy of individual data sources. Each device or node contributes to improving the global model's accuracy through local model updates, while keeping their respective data secure and private. This is especially crucial in scenarios where data sharing might be restricted due to privacy regulations, ownership concerns, or sensitive information.

Centralized Federated Learning

In Centralized FL, a central server coordinates model training across multiple client devices [12]. The server sends the model's initial parameters to clients, who compute updates using their local data. These updates are then aggregated on the server to refine the model. While providing better control over the learning process, this approach may raise privacy concerns due to central data aggregation.

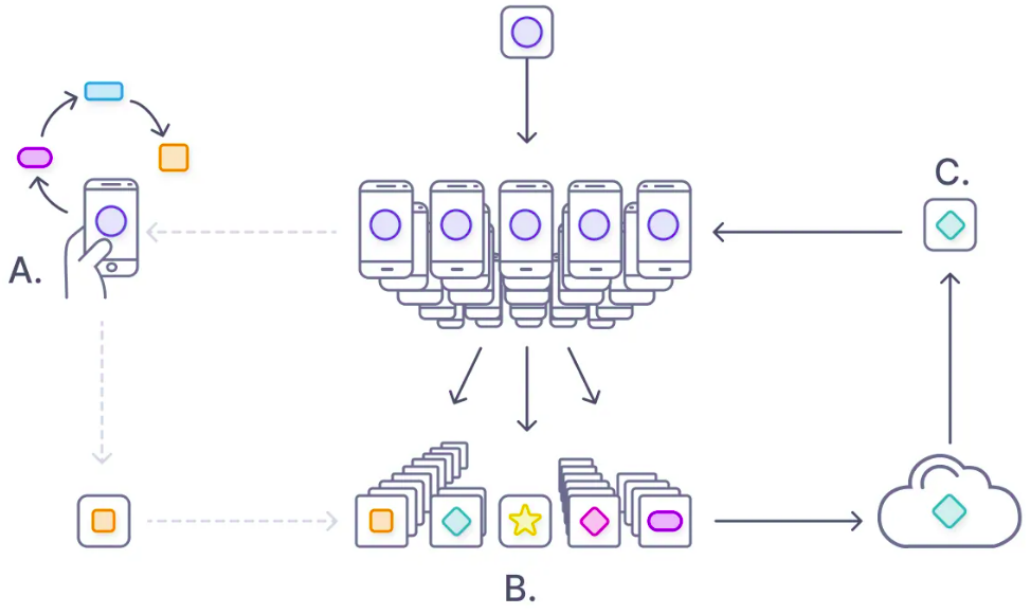

Figure 2.1.3: Centralized Federated Learning [12]

Decentralized Federated Learning

In contrast, Decentralized FL operates without a central server. Client devices directly communicate and collaboratively update the model [12]. This mode minimizes privacy risks associated with central aggregation, but it introduces challenges such as communication overhead and synchronization issues.

Heterogeneous Federated Learning

Heterogeneous FL extends FL to handle varying device capabilities, network conditions, and data distributions [12]. It accommodates devices with distinct processing power and data characteristics. This adds complexity to aggregation methods, as different devices may contribute unequally.

Iterative Nature

Federated Learning often follows an iterative pattern. In each round, clients update the model using local data and communicate with a central entity to aggregate updates. This iterative process iteratively enhances the model's accuracy while maintaining data privacy. Techniques like Federated Averaging are used for aggregation, where model parameters are averaged to create a new global model.

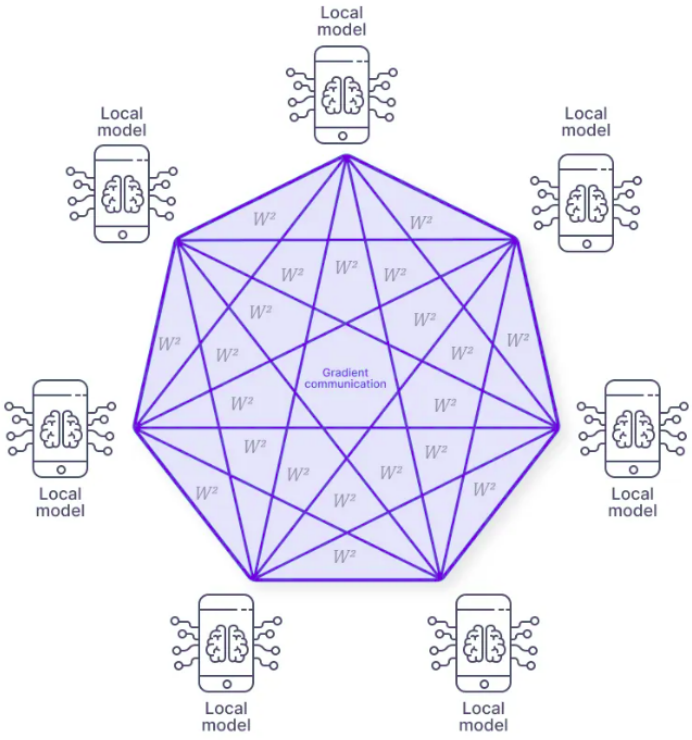

Figure 2.1.4: De-centralized Federated Learning [12]

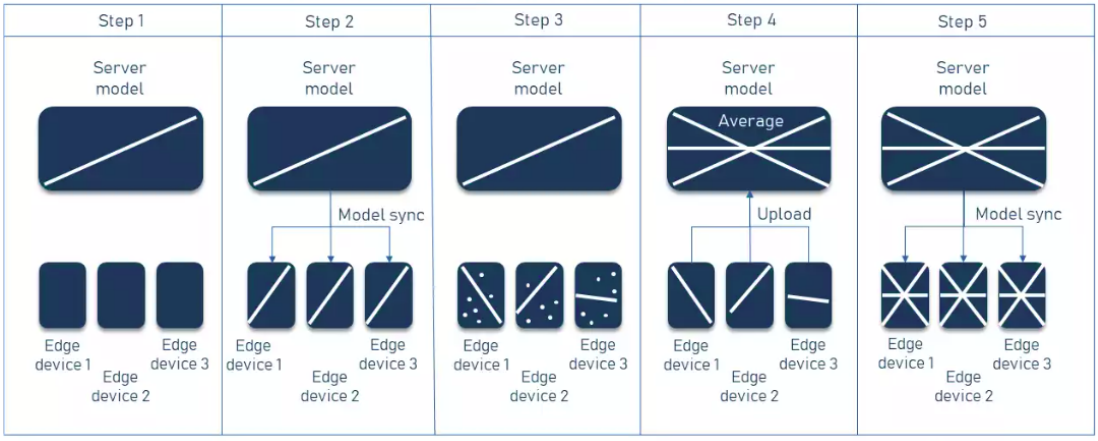

Figure 2.1.5: Federated Learning steps [13]

Aggregation Algorithms

Federated Stochastic Gradient Descent (FedSGD) operates within the framework of traditional stochastic gradient descent (SGD), wherein gradients are calculated on mini-batches comprising a fraction of total data samples [13]. In the federated context, these mini-batches resemble distinct client devices housing localized data.

In FedSGD, the central model is disseminated to clients, with each client independently computing gradients using its local data. These gradients are then transmitted to the central server, which amalgamates them in proportion to the number of samples per client. This collective effort culminates in the calculation of the gradient descent step, effectively advancing the central model.

Federated Learning with Dynamic Regularization (FedDyn) addresses the regularization aspect present in traditional machine learning, where penalties are introduced to the loss function for enhanced generalization [13]. In federated learning, the global loss originates from local losses derived across heterogeneous devices. Acknowledging the divergence between minimizing global and local losses due to client disparities, FedDyn endeavors to generate regularization terms for local losses. This dynamic regularization adapts to data statistics, such as data volume or communication costs, thereby harmonizing local losses to converge towards the global loss.

Federated Averaging (FedAvg) extends the FedSGD paradigm [13]. Here, clients undertake multiple local gradient descent updates and share the adjusted weights of their local models. The central server aggregates these client weights, thereby consolidating the model parameters. By starting from a shared initialization, Federated Averaging generalizes the concept of FedSGD—averaging gradients aligns with averaging weights. Consequently, Federated Averaging accommodates local weight tuning before transmission to the central server for collective averaging.

Input:

- Clients: $C_1, C_2, \ldots, C_n$
- Initial global model: M_0
- Number of communication rounds: T

For $t = 1$ **to** T:

1. **For** each client C_i:
    - Update local model: $M_i^t \leftarrow M_i^{t-1} - \eta \nabla f_{C_i}(M_i^{t-1})$

2. Calculate weighted average of local models:

$$M^t \leftarrow \frac{1}{n} \sum_{i=1}^{n} w_i M_i^t$$

3. Update global model: $M_{\text{global}} \leftarrow M^t$

Output:

- Trained global model: M_{global}

FLWR (A Friendly Federated Learning Framework)

FLWR (A Friendly Federated Learning Framework) is an advanced federated learning framework that was designed to tackle the challenges of scalability, compatibility, and deployment in real-world systems, and that is what we will be discussing in this section of the methodology. Researchers and developers are given the ability to work with a large number of clients through the use of Flwr, which enables workloads with tens of millions of clients to be completed without a hitch. The machine learning framework is agnostic to other machine learning frameworks, which makes it compatible with widely used options such as Keras and PyTorch and even raw NumPy without automatic differentiation [15].

One of Flwr's most notable qualities is its adaptability, which allows it to run on a wide variety of platforms, including mobile, edge devices, and the cloud. Flower can be used by researchers on servers such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure, as well as on mobile devices running Android and iOS. Flower is compatible with even edge devices such as Raspberry Pi and Nvidia Jetson, which makes it easier to conduct research and deploy across a wide variety of environments.

Flwr acts as a bridge between research and production, providing a seamless transition from the conception of an idea to its actual implementation. Models developed using Flower can be easily integrated into real-world systems with minimal engineering effort and proven infrastructure. This process, which begins as a research project, is made possible by Flower.

Flwr's ability to function across a variety of operating systems and hardware platforms is made possible by the fact that it is platform-agnostic. Because of its adaptability, it is ideally suited for heterogeneous edge device environments, in which various devices may run on a variety of different configurations.

Another one of Flwr's defining characteristics is its usability. Developers are able to construct a fully functional federated learning system with only twenty lines of Python code. Researchers and developers will have a much simpler time getting started with their

preferred machine learning frameworks thanks to the user-friendliness of the framework and the comprehensive code examples it provides.

Researchers and developers are given the ability to conduct large-scale distributed machine learning with relative ease thanks to its capability as a robust and scalable federated learning framework. Flower is a leading choice for developing privacy-preserving and scalable machine learning solutions for a variety of real-world applications due to its compatibility with a wide variety of frameworks and platforms, as well as its seamless integration from research to production.

2.2 Literature Review

The literature on PV power forecasting is vast and covers various aspects like data collection, preprocessing, feature selection, and model selection. The use of ML techniques for PV power forecasting has been explored by various researchers. Classical ML techniques like Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Random Forests (RF) have been used to predict PV power generation. Deep Learning (DL) techniques like Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Autoencoders have also been applied to PV power forecasting. With all these developments the need for finding best model for time series forecasting becomes imminent.

2.2.1 Time Series Forecasting

Recently, deep learning techniques such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) have been applied to time series point forecasting. Authors of [32] conducted a review on the use of deep learning in time series modeling across various fields of study.

A study [8] conducted a comparative study on the performance of different RNNs applied to the Short Term Load Forecasting problem and concluded by arguing that ERNN and ESN may represent the most convenient choice in time series prediction problems, both in terms of performance and simplicity of their implementation and training. Authors of [10] investigated the application of attention models for Seq2Seq models on both univariate and multivariate time series. These extensions perform significantly better than the original attention model as well as state of-the-art baseline methods based on ARIMA and random forests [9] applied dilated CNNs specifically on financial time series, and concluded that even though time series forecasting remains a complex task and finding one model that fits all is hard, they showed that the WaveNet is a simple, efficient and easily interpretable network that can act as a strong baseline for forecasting. Nevertheless there was still room for improvement. However, these studies were all based on the Recursive

strategy.

Researchers in [44] analyzed the performance of different multi-step strategies using a Multi-Layer Perceptron (MLP), highlighting the effectiveness of the Direct Multi-Horizon strategy. For probabilistic forecasting using encoder-decoder models, [48] proposed DeepAR, a Seq2Seq architecture that utilizes an identical encoder and decoder. DeepAR directly outputs parameters of a Negative Binomial distribution. This approach is similar to [38], where an MLP predicts Gaussian parameters. The training of DeepAR involves maximizing likelihood and using Teacher Forcing during training, while during prediction, it samples from the estimated parametric distribution multiple times to generate a series of sample paths. Their method differs from DeepAR by using the more practically relevant Multi-Horizon strategy, a more efficient training approach, and directly generating accurate quantiles.

In the domain of quantile forecasting using neural networks, [45] used an MLP to generate quantile forecasts for financial returns, while [55] designed a quantile autoregressive neural network for stock price prediction. This approach fed previously estimated quantiles into the model instead of the mean estimate or a sampled instance. Neither of these approaches utilized sequential nets and their temporal nature effectively. Taylor's approach relied on an external model, and this approach faced challenges in justifying the feedback of quantiles into the model.

Study [53] presents a framework that addresses general probabilistic multi-step time series regression. Their approach leverages the power of Sequence-to-Sequence Neural Networks, such as recurrent and convolutional structures, to capture the temporal characteristics of the data. They also utilize the non-parametric nature of Quantile Regression for probabilistic forecasting. Additionally, they take advantage of the efficiency offered by Direct Multi-Horizon Forecasting.

In reference [56], the authors introduce a neural network model based on quantile regression for load forecasting, aiming to estimate the range of uncertainty in load predictions. In [41], the authors focus on developing an uncertainty model for PV generation using regression techniques. They employ quantile scores as a metric to measure the uncertainty in their forecasts. Similarly, in [2], the authors employ bootstrap confidence intervals to quantify the uncertainty in predicted PV power. The authors of [43] discuss the forecasting uncertainty specifically for residential net load. In this particular article, the authors evaluate various machine learning models and provide additional insights regarding the impact of factors such as prosumer location, load consumption profile, and dataset size on prediction accuracy. While common assessment metrics like Root Mean Square Error (RMSE), MAE, and pinball loss function are mentioned, this paper offers a broader examination of these models and presents novel findings related to the mentioned influencing factors. The evaluation results demonstrate the superior performance of the proposed Bayesian deep learning-based method and highlight the

improvements contributed by the Clustering Stage and the PV visibility.

A latest research [3] implemented multiple Machine Learning (ML) models to forecast the power generation of photovoltaic (PV) systems and the household consumption in a smart energy grid. Additionally, they incorporated a measure of uncertainty in their predictions by providing quantile values as bounds to assess the level of uncertainty.

For nearly a decade, there has been active research on processing large amounts of time series data. One notable work in this field focuses on processing trillions of subsequences of time series using the dynamic time warping distance measure, which is computationally expensive. Since then, several new proposals have emerged to tackle the challenge of processing time series data on an even larger scale. These include the FastShapelet (FS) algorithm [42], which reduces the time complexity of the original method at the expense of accuracy, a generic and scalable framework for automated anomaly detection in large-scale time series data [25], a fast and scalable Gaussian process modeling approach for astronomical time series [16], and a scalable distance-based classifier for time series called proximity forest [34]. These works demonstrate the increasing interest in processing larger sets of time series data. However, these approaches still face limitations imposed by traditional computation models and systems, such as insufficient resources to handle large problems, which can lead to memory storage issues or unacceptable running times.

To address the aforementioned limitations, the Distributed FastShapelet Transform (DFST) algorithm [5] was introduced as the first time series classification algorithm developed in a distributed manner. DFST combines the low complexity of the FastShapelet (FS) algorithm with the performance of the Shapelet Transform (ST) [29]. ST uses the distance between selected shapelets and each time series in the dataset as input features, making it a feature-based method. The performance of ST depends on the machine learning algorithm used on the transformed dataset, but it achieves competitive results compared to the best state-of-the-art approaches. Additionally, the DFST method allows the application of existing vector-based algorithms in Apache Spark to time series problems, expanding the range of tools available for processing this type of data. However, this approach is limited to supervised problems.

2.2.2 Distributed Time Series Forecasting

Time series analysis presents unique characteristics compared to traditional vector-based problems. These include time dependency, trend, seasonality, and stationarity, among others, which must be taken into account when designing algorithms. These characteristics add complexity to the methods or impose certain limitations on them, making it challenging to apply the proposed methods in distributed environments. For example, the FS algorithm analyzes the entire dataset sequentially to construct the best decision tree based on the discovered shapelets, evaluating each shapelet with the

complete dataset. In contrast, DFST evaluates shapelet candidates in a distributed manner on the data available in each node and saves the most valuable ones. This is necessary because the shapelet evaluation process is computationally demanding, making it infeasible to apply it to the complete dataset in Big Data environments. In addition to the feature-based approach used in DFST and ST, there are other works that focus on extracting features from time series data without relying on shapelets. These approaches involve applying various mathematical operations to derive valuable information about the underlying structure and behavior of time series [17, 18], selecting the most representative features based on theoretical considerations [22], or conducting extensive experimentation to identify a set of 22 characteristics [33] that are deemed the most representative among the original set of features. Unsupervised feature extraction techniques have been successfully applied in other domains [39]. Recently, it has been demonstrated that a set of well-known complexity measures and time series features can achieve competitive results in univariate [4] and multivariate [6] time series classification. To extend this approach to a distributed environment [19], it is necessary to filter and prepare the selected features to be completely independent of each other and not rely on relationships between different time series or additional information. These conditions allow for their inclusion in a distributed environment, thereby expanding the limited range of tools available for time series processing in Big Data environments.

Authors of [7] proposes the Scalable and Distributed Transformation for Univariate and Multivariate Time Series (SCMFTS) using a MapReduce framework, which enables a scalable and distributed approach for processing time series data in Big Data environments. SCMFTS is based on well-known time series features and aims to provide a traditional vector-based representation for time series data. By transforming time series data into this vector-based representation, SCMFTS enables the application of algorithms that are not specifically designed for time series problems. The SCMFTS approach leverages the MapReduce paradigm to distribute the computation needed for transforming time series data across multiple machines in a cluster. The map operation applies the transformation independently to each instance of the dataset, while the reduce operation combines the results from the map operation. This distributed approach allows SCMFTS to handle large volumes of time series data that would be impractical to process on a single machine. By utilizing well-known time series features, SCMFTS provides a vector-based representation that can be used with a wide range of algorithms. This means that existing algorithms that are not specifically tailored for time series analysis can be applied to time series problems using SCMFTS. This approach expands the available tools for processing time series data in Big Data environments, allowing for more efficient and scalable analysis. Overall, SCMFTS offers a scalable and distributed transformation method for univariate and multivariate time series data, enabling the use of non-time series specific algorithms and increasing the capabilities of time series analysis in Big Data environments.

Open Prediction System (OPS) [20], an automated system for developing predictive models. OPS is a versatile predictive system that can be applied to various problem domains. It specifically focuses on predicting outcomes in multivariate time series data, addressing challenges commonly faced by utility companies involved in the distribution and control of their commodities.

2.2.3 Federated Learning based Forecasting

Federated Learning (FL) is a technique that enables a set of devices to train on local data and send updates to a shared model using distributed stochastic gradient descent (SGD) [35]. It has shown promise in numerous fields, including IoT-based energy control in smart buildings [11, 36], public health [24], traffic prediction [31], and load forecasting [14, 37]. FL has also been used in predicting socio-demographic characteristics for energy utilities to offer diversified services to their consumers [51]. Researchers have compared FL with centralized and localized forecasting in electrical load forecasting, and the results show that FL performs better than centralized forecasting in situations where access to training data is not possible but is worse than localized forecasting [37]. A recent study [26] proposes an innovative federated deep generative learning framework for renewable scenario generation, which outperforms the state-of-the-art centralized methods. Another study [28] introduces an FL-based Bayesian neural network (FL-BNN) to preserve the privacy of utilities in behind-the-meters (BTMs) estimation, enabling a customized model for each client. The FL-BNN model outperforms the centralized BNN model and other benchmarks. However, the existing work related to FL-based solar forecasting [57] lacks an in-depth analysis of the impact of different input variables, despite the highly correlated time series used for training. Although the FL training process keeps raw data decentralized, optimizing a shared model can be vulnerable to non-IID data [21]. Therefore, the proposed forecasting scheme aims to train multiple customized models that fit various real-world data sources instead of a global model.

In [52], a new solar forecasting framework is introduced that combines a spatial and temporal attention-based neural network (STANN) with federated learning (FL). The framework is designed to handle multi-horizon forecasting scenarios ranging from 5 to 30 minutes. The STANN model is composed of a feature extractor and a forecaster, which can be trained on different local datasets for improved localization. The framework allows for global parameter aggregation without the need for data gathering, which further enhances the accuracy of the forecasts. The FL technique employed in the framework makes it highly flexible and adaptable to a variety of data sources.

The raw data stays at each location while the weights from several local models are combined on a central server to create a common model. This aggregation functions well for parametric approaches like Linear Regression and Neural Networks since it is simple to compute an average or other kind of aggregate of the values. However, it is still

not obvious how non-parametric ML techniques like Random Forests and Decision Trees can be combined in this way. [54] and [30] have provided some insights on performing federated learning on tree based models.

Authors of [30] address these issues and propose Federated Forest, a privacy-preserving machine learning model that achieves the same level of accuracy as the non-privacy preserving approach. Federated Forest is a lossless learning model of the conventional random forest method. They built a secure cross-regional machine learning system on top of it that enables a learning process to be jointly trained over clients from various regions using the same user samples but different attribute sets, processing the data stored in each of them without transferring their raw data.

Authors of [54] suggest Pivot, a cutting-edge approach to privacy-preserving vertical decision tree training and prediction, which makes sure that only the final tree model and the prediction output, as agreed to by the clients, are disclosed. Pivot does not rely on any reliable third parties and offers defense against a marginally honorable adversary who might compromise m out of m clients. However this area is still open for new developments and integration into an existing frameworks.

2.3 Relation to Research Questions

1. How does Federated learning impact forecast accuracy of Multi-Variate Time Series Forecasting in the context of Renewable Energy Systems?

2. How can Federated Learning be implemented on tree-based models?

In this entire background and literature review section, the key components to understand these questions and finding relevant answers were discussed. There was detailed discussion on Multi-Variate Time Series and tree-based models like LightGBM and Catboost. Background and relevant research on these topics was presented to familiarize readers with them. Secondly, the evaluation metrics were presented and explained in depth to allow readers to understand how the performance of forecasts of these models can found to answer the question. Lastly there was introduction of Distributed Learning, and consequently Federated Learning that is the key topic in this research question. Alot of research work has been reviewed to show readers what has been already achieved in this topic, and what is yet to be discovered, especially in domain of Renewable Energy Systems.

Chapter 3

Design

One way to implement federated learning for PV production and consumption forecasting models is to use a federated learning framework. This framework consists of a central server and multiple clients, each of which has its own local data. The server sends the model parameters to each client, and the clients train the model on their local data. The clients then send the updated model parameters back to the server, which aggregates them and updates the global model. This process is repeated for multiple rounds until the model converges.

To apply federated learning to PV production and consumption forecasting models, the data from different sources must be preprocessed and standardized to ensure consistency across the different sources. This may involve normalizing the data, removing outliers, and filling in missing data. The model must also be designed to handle the variability in the data from different sources and to be robust to changes in the data distribution over time. Another consideration when using federated learning for PV production and consumption forecasting models is the selection of the clients that participate in the training and testing process. The clients should be selected to represent a diverse set of data sources to ensure that the model is trained on a representative sample of the data. The clients should also be selected based on their ability to contribute high-quality data to the training process.

3.1 Data Collection

The collection of data from seven different households located in Uppsala, Sweden, over a period of 14 months was done and provided by authors of [3]. This work is extension of research in [3]. The data was recorded at an hourly resolution and served as the foundation for creating a comprehensive dataset used to train and evaluate our forecast algorithm. This dataset is represented as an N × k feature matrix, where N represents the number of data points, and k denotes the number of features. Each feature vector

within the matrix contains a combination of household measurements and additional weather-related features obtained from the Swedish Meteorological and Hydrological Institute (SMHI). The inclusion of these weather features was intended to enhance the accuracy of our machine learning model in predicting both photovoltaic (PV) production and household energy consumption.

Table 3.1.1 presents the weather features that were integrated into the dataset, each with its respective unit of measurement. These features include temperature, dew point, humidity, precipitation, wind direction, wind speed, air pressure, and global radiation. On the other hand, Table 3.1.2 outlines the prosumer features incorporated in the dataset, which are directly related to individual households and their energy dynamics. These prosumer features include bought power, produced power from the installed PV system, sold power back to the main grid, and the total consumed power by the household. By combining the prosumer features with the weather features, we aimed to create a comprehensive dataset that captures the key factors influencing PV production and energy consumption in households. It is worth noticing as the data is captured per hour, and denotes value of power for that hour, it is taken in Watts (W). This will stay the metric for all the true values of power productions and consumption and their predictions made in course of this research.

This dataset was utilized to train and evaluate our machine learning model, with the ultimate goal of improving the accuracy of PV production and consumption predictions for smart energy management systems. By leveraging the wealth of information from both household measurements and weather-related data, we sought to enhance the capabilities of our forecast algorithm and provide more reliable and efficient energy management solutions for residential prosumers.

Feature	Unit
Temperature	C
Dew Point	C
Humidity	Percentage
Precepitation	L/m^2
Wind Direction	Degrees
Wind Speed	m/s
Air Pressure	mBar
Global Radiation	W/m^2

Table 3.1.1: Weather Features

Feature	Unit
Bought Power	W
Produced Power	W
Sold Power	W
Consumed Power W	W

Table 3.1.2: Prosumer Features

3.2 Data Preprocessing

The outcome variable represents either the production or consumption of electricity by the household. The consumption value is derived from the equation:

$$P_{Consumed} = P_{Import} + P_{Produced} - P_{Export} \tag{3.1}$$

where P_{Import} represents the electricity bought from the main energy grid, $P_{Produced}$ is the electricity generated by the installed PV system, and P_{Export} is the electricity sold back to the main grid. Figures 3.2.1 3.2.2 in the paper shows histograms and Kernel

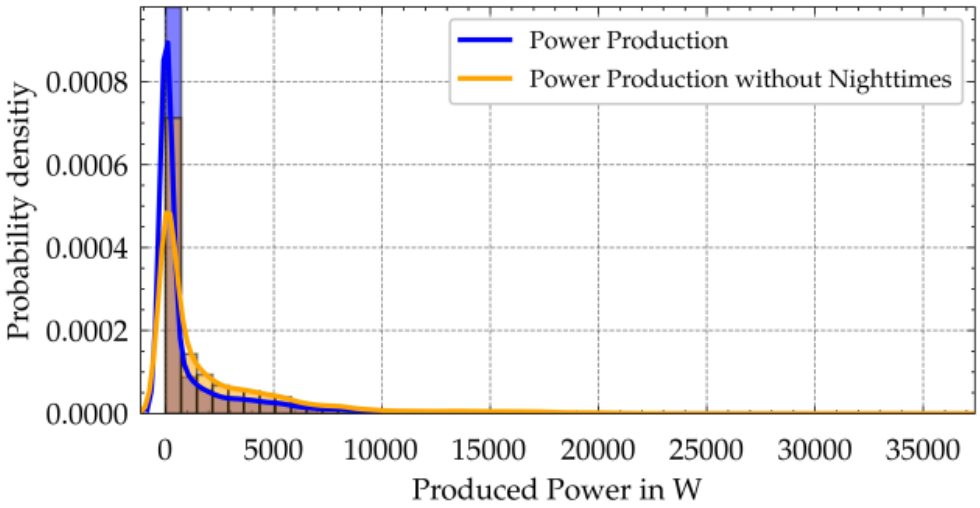

Figure 3.2.1: Histograms of the power production with fitted KDEs [3]

Density Estimations (KDE) of the produced and consumed power. Figure 3.2.1 displays the histogram of produced power, with and without night times. During training and evaluation, night times were removed for production data since PV cells do not generate electricity at night. Figure 3.2.2 illustrates the consumption of all prosumers (households with PV systems). Notably, P Consumed can sometimes fall into the negative range in the observed dataset, indicating instances when the household sells stored energy from its battery back to the main grid without producing or buying additional energy.

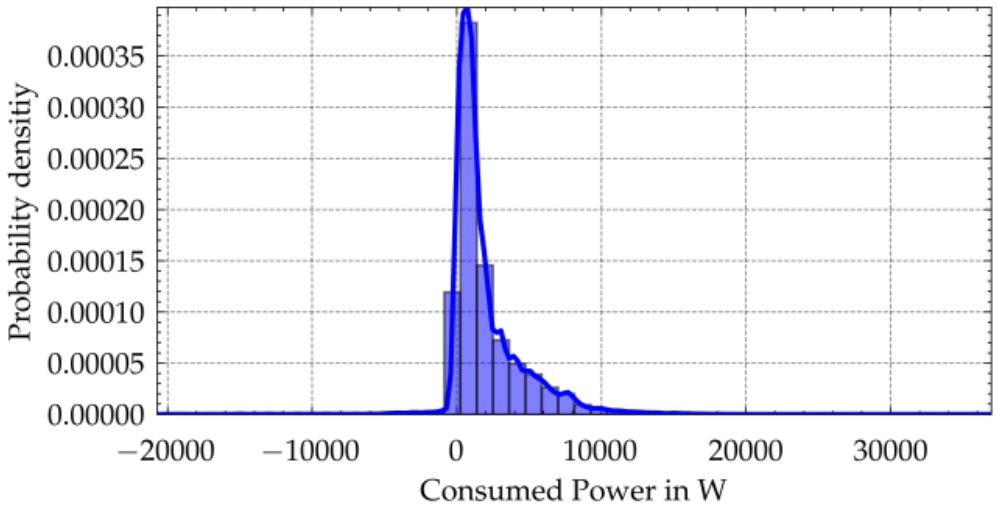

Figure 3.2.2: Histograms of the power consumption with fitted KDEs [3]

3.3 Model Selection

The model selection process in our research involved considering two powerful gradient boosting algorithms, namely LightGBM and CatBoost. These algorithms are well-known for their efficiency, scalability, and ability to handle categorical features effectively. Three models for upper quantile, lower quantile, and mean are created for both of them. To ensure the optimal performance of these models, GridSearchCV is employed, a widely used hyperparameter tuning technique.

3.3.1 LightGBM

LightGBM (Light Gradient Boosting Machine) [23] has emerged as a powerful and efficient tool for accurate PV power forecasting. As the integration of solar energy sources continues to grow, the need for precise and reliable predictions of PV power generation becomes crucial for grid stability and efficient energy management. LightGBM's exceptional speed and efficiency are major advantages, making it capable of processing large-scale datasets with high-dimensional features efficiently. This is particularly valuable in real-time applications, where quick updates and adjustments are required as weather conditions change rapidly.

One of the significant challenges in PV power forecasting is dealing with the nonlinear behavior of solar energy generation due to varying weather conditions and other factors. LightGBM's ability to build complex models by combining multiple decision trees allows it to capture these nonlinear relationships effectively. This enhances the accuracy of PV power forecasts, as the model can adapt to the dynamic and intricate nature of solar energy

generation.

Moreover, LightGBM natively handles categorical features without the need for extensive data preprocessing, making it well-suited for PV power forecasting tasks. Categorical variables such as weather conditions, time of day, and day of the week can be critical predictors for PV power generation. LightGBM efficiently utilizes this information in its models, enhancing the forecasting accuracy by considering the impact of various factors.

PV power generation data may contain outliers due to factors like equipment malfunctions or unusual weather events. LightGBM's robustness to outliers ensures that these extreme values do not unduly influence the forecasting model. This feature is essential for maintaining the accuracy and reliability of the forecasts even in the presence of unpredictable events.

As the deployment of solar energy systems expands, the volume of data for PV power forecasting also grows. LightGBM's scalability and support for distributed computing make it well-suited for handling large datasets, allowing it to scale efficiently and meet the demands of increasing data volumes.

Additionally, LightGBM can be adapted to provide forecasts for different quantiles, offering a more comprehensive understanding of the uncertainty associated with PV power predictions. This ability is particularly valuable in decision-making processes that require quantifying risk and reliability. By providing forecasts for various quantiles, LightGBM allows energy planners and grid operators to make more informed decisions and design robust energy management strategies.

Furthermore, LightGBM's ability to provide insights into feature importance allows domain experts to understand which variables have the most significant impact on the forecasted PV power output. This knowledge can be leveraged to optimize the design and maintenance of solar energy systems, leading to improved energy utilization and efficiency.

3.3.2 Catboost

Catboost [40], an open-source machine learning library developed by Yandex, has emerged as a powerful tool for accurate PV power forecasting. Its unique features make it well-suited for handling the challenges in predicting photovoltaic power generation. One of the key advantages of Catboost is its native handling of categorical features without requiring manual preprocessing, allowing it to efficiently incorporate weather conditions, seasonal variations, and time of day as essential predictors for accurate forecasts. Moreover, Catboost's robustness to overfitting is crucial when dealing with noisy PV power generation data, ensuring that the model does not excessively fit to noise

and maintains its accuracy.

In addition to handling nonlinear relationships effectively, Catboost's built-in support for time series data is highly relevant for PV power forecasting, as it can capture temporal patterns in solar energy generation. Furthermore, the library's default hyperparameter settings often provide competitive performance, reducing the need for extensive tuning and simplifying the modeling process for users without deep machine learning expertise.

Another advantage of Catboost is its interpretability, offering insights into feature importance, enabling energy experts to understand which variables contribute the most to PV power forecasts. This transparency enhances the overall understanding of the forecasting process and validates the relevance of domain-specific features.

Catboost's ability to be accelerated using GPUs further improves its computational efficiency, making it suitable for large-scale training tasks, especially when dealing with extensive historical data for PV power forecasting.

Incorporating Catboost into PV power forecasting pipelines empowers researchers and energy experts to build robust and accurate prediction models. Its efficient handling of categorical features, robustness to overfitting, and support for time series data make it an excellent choice for capturing the complexity of solar energy generation patterns. Moreover, its ease of use with default hyperparameters and interpretability features make it accessible to a wide range of users, contributing to efficient and effective forecasting in the renewable energy domain.

3.3.3 Quantile Regression

Quantile Regression (QL) is a technique used in machine learning to estimate specific quantile values of a target variable. It involves training separate models for each quantile value of interest using the quantile loss function, as defined in Equation 3.2. The quantile loss function shown in Equation 2.2 allows the model to focus on predicting the desired quantile of the target variable, rather than the mean, which is the focus of traditional regression methods.

$$L(y, \hat{y}_p; p) = \max(p(y - \hat{y}_p), (1 - p)(\hat{y}_p - y)) \tag{3.2}$$

QR aim to estimate three quantile values: $\hat{y}_l$, $\hat{y}_m$, and $\hat{y}_u$, representing the lower, median, and upper quantiles, respectively. Each quantile value is associated with a specific level of uncertainty in the data. For example, $\hat{y}_m$represents the median, which divides the data into two equal halves, while $\hat{y}_l$ and $\hat{y}_u$ provide estimates for the lower and upper bounds, respectively.

To achieve this, separate models are trained for each quantile value, using the quantile loss function during training. The quantile loss function penalizes the model differently based on the quantile value being estimated, and this encourages the model to produce more accurate predictions for the corresponding quantile.

The beauty of QR is its flexibility in choosing the regression algorithm. Any regressor that can use the quantile loss function during training is suitable for QR. This means that various regression algorithms, such as linear regression, decision trees, random forests, or even more advanced techniques like gradient boosting algorithms (e.g., LightGBM, Catboost) can be utilized for QR. This flexibility allows us to tailor the choice of regression algorithm to the specific characteristics of the data and the problem at hand, ultimately leading to more accurate and robust quantile estimations.

3.3.4 Hyper-Parameters

Hyperparameters are parameters that are set before the learning process begins and cannot be learned directly from the data. In order to achieve the best possible performance from tree-based models like Gradient Boosting Machines (GBM), there are a number of significant hyperparameters that need to be tuned. The following are some hyperparameters that are frequently used in forest-based models:

1. Max Depth (max depth): This hyperparameter determines the maximum depth that can be reached by any one of the individual decision trees that make up the ensemble. A deeper tree has the potential to recognize more intricate patterns in the data, but it also raises the risk of overfitting. It is essential to set an appropriate value for the max depth variable in order to strike a balance between the level of model complexity and the level of generalization.

2. Num Leaves (num leaves): This hyperparameter in LightGBM allows the user to specify the maximum number of leaves that can exist on a single tree. Increasing the number of leaves in a model can make the model more complicated, but it also raises the possibility that the model is being overfit. It is absolutely necessary to tune this parameter in order to achieve the best possible balance between the generalization and complexity of the model.

3. Learning Rate (learning rate): Also known as the shrinkage rate or step size, the learning rate determines the size of the step at which the model is updated during each boosting iteration. The learning rate is controlled by the learning rate variable. The model may become more robust and it may be easier to avoid overfitting if the learning rate is slowed down. However, in order to achieve good performance, it may require additional iterations of the boosting process.

4. N Estimators (n estimators): The value of this hyperparameter indicates the total

number of boosting stages and trees that are included in the ensemble. Improving the performance of the model by increasing the number of estimators may come at the expense of an increase in the computational cost. It is necessary to strike a healthy balance between the amount of training time spent and the performance of the models.

5. Subsample (subsample): This hyperparameter controls the fraction of samples that are used for fitting the individual trees in each boosting iteration. The default value for this hyperparameter is 1. Randomness is introduced into the training process when the subsample parameter is set to a value that is less than 1.0. This can help improve model generalization.

6. The Colsample Bytree (colsample bytree) hyperparameter is used in XGBoost to specify the fraction of features that will be randomly selected for each tree. It adds an element of randomness and has the potential to help reduce overfitting by employing a unique subset of features for each tree.

7. Reg Alpha (reg alpha): This hyperparameter, which also goes by the name L1 regularization, includes an L1 penalty term in the objective function while it is being trained. It does this by encouraging sparsity in the importance of features, which helps to prevent overfitting.

8. Reg Lambda (reg lambda): This hyperparameter, which is also referred to as L2 regularization, includes an L2 penalty term in the objective function while it is being trained. It does this by assigning a penalty to large coefficient values, which helps to prevent overfitting.

Because it has such a direct bearing on the performance of the model and its capacity for generalization, hyperparameter tuning is an essential step in determining the values that should be used for these parameters. When looking for the optimal combination of hyperparameters, it is common practice to employ search strategies such as grid search, random search, or Bayesian optimization. In addition, the performance of the model is assessed using cross-validation in order to prevent overfitting and evaluate how well it works with a variety of hyperparameter settings.

3.3.5 GridSearchCV

GridSearchCV is a systematic approach that automatically explores a pre-defined set of hyperparameters for each model and evaluates their performance on the dataset. It exhaustively searches through all possible combinations of hyperparameter values, facilitating the identification of the best combination that maximizes the model's performance. In tables 3.3.1 and 3.3.2 for both LightGBM and CatBoost, we defined a range of hyperparameter values to explore during the GridSearchCV process.

The hyperparameters included learning rates, tree depths, number of estimators,

regularization parameters, and more, depending on the specific requirements of each algorithm. GridSearchCV then assessed the performance of each model using cross-validation, which involves splitting the dataset into multiple subsets and iteratively training the model on one subset while evaluating it on the other. This helps prevent overfitting and provides a more robust evaluation of the models.

Using GridSearchCV with LightGBM, Catboost, or any other model in a federated learning context can yield benefits, it's essential to strike a balance between optimizing model performance and minimizing communication overhead. Customizing hyperparameters for federated learning characteristics can lead to improved convergence and overall model quality in this decentralized and collaborative learning paradigm

Parameters	Range
max depth	[3, 4, 5]
num leaves	[10, 15, 20]
learning rate	[0.05, 0.1, 0.15]
n estimators	[50, 100, 200]
subsample	[0.5, 0.7, 0.9]
colsample bytree	[0.5, 0.7, 0.9]
reg alpha	[0.01, 0.1, 1]
reg lambda	[0.01, 0.1, 1]

Table 3.3.1: Initial parameters for LightGBM customized for GridsearchCV

Parameters	Range
max depth	[3, 4, 5]
learning rate	[0.05, 0.1, 0.15]
n estimators	[50, 100, 200]
subsample	[0.5, 0.7, 0.9]

Table 3.3.2: Initial parameters for Catboost customized for GridsearchCV

By applying GridSearchCV to both LightGBM and Catboost, we systematically determined the optimal hyperparameter configurations for each algorithm. These configurations were chosen based on their ability to produce the best results for our energy consumption and PV production forecasting tasks.

3.4 Federated Design

Here the design of the federated setup is shown, with in depth implementation in next chapter. It should be noted that this setup is an extension of the existing FLWR Framework, as the aggregation function for FedAVG is changed to aggregation fr fit, which

aggregates tree-based models and at the same time can handle multiple models per client (typically one). We would need this as we want to predict the upper quantile, lower quantile, and mean of every client. LightGBM and Catboost only support one ”alpha” value per model that actually handles if a prediction is going to be for the upper quantile or lower quantile or mean.

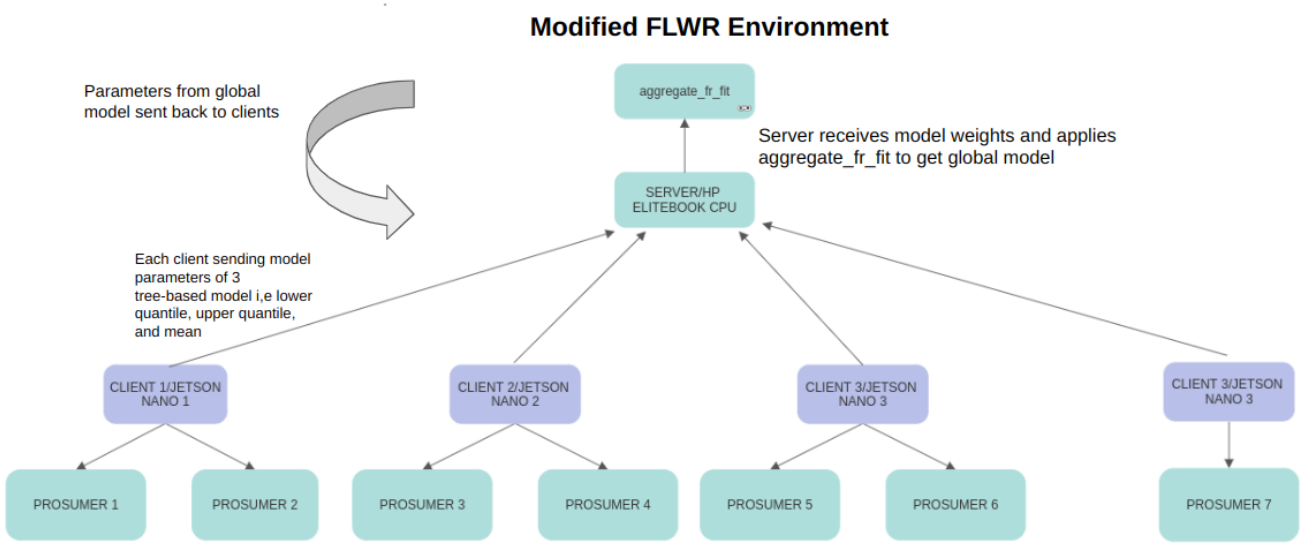

Figure 3.4.1: Federated Learning Design with modified flwr framework

3.5 Aggregation for Tree-based models

As a main core of this thesis, aggregate fr supports aggrgation of tree-based models. Currently, in FedAVG strategy flwr uses aggregate function to create glabal model for parameters received from clients. Below you can find algorithms for both functions. Implementation will be discussed in detail in next chapter.

The two provided algorithms, aggregate fr and aggregate, both tackle the task of calculating a weighted average of model weights in a federated learning context. While both algorithms fulfill this objective, aggregate fr stands out as a more comprehensive and specialized solution, particularly advantageous when dealing with tree-based models like LightGBM and Catboost.

The algorithm commences by calculating the total number of examples used in training across all clients. It then proceeds to create a list named weighted weights, where each element is a dictionary representing model weights from a client, scaled by the number of examples they utilized.

Algorithm 1 FedAvg Aggregate

Require: $results$: List of Tuples, each containing model weights and number of examples used for training by a client
Ensure: $weights\_prime$: Averaged model weights
1: Initialize $num\_examples\_total$ as 0
2: Initialize an empty list $weighted\_weights$
3: **for** each tuple $(weights, num\_examples)$ in $results$ **do**
4: Append $[layer \times num\_examples$ for layer in weights] to $weighted\_weights$
5: Increment $num\_examples\_total$ by $num\_examples$
6: **end for**
7: Initialize an empty list $weights\_prime$
8: **for** each layer in the zipped list of $weighted\_weights$ **do**
9: Calculate average weight: $weights\_prime \leftarrow \frac{1}{num\_examples\_total} \sum_{i=1}^{n} layer[i]$
10: **end for**
11: **return** $weights\_prime$

Algorithm 2 FedAvg Aggregate fr

Require: $results$: List of Tuples, each containing model weights and number of examples used for training by a client
Ensure: $weights\_prime$: Averaged model weights
1: Initialize $num\_examples\_total$ as 0
2: Initialize an empty list $weighted\_weights$
3: **for** each tuple $(weights, num\_examples)$ in $results$ **do**
4: Convert $weights$ to a dictionary
5: **for** each key key in $weights$ **do**
6: Multiply $weights[key]$ by $num\_examples$
7: **end for**
8: Append the weighted weights dictionary to $weighted\_weights$
9: Increment $num\_examples\_total$ by $num\_examples$
10: **end for**
11: Initialize an empty dictionary $weights\_prime$
12: **for** each key key in $weighted\_weights[0]$ **do**
13: Initialize an empty list $layer\_updates$
14: **for** each weighted weights dictionary $weights$ in $weighted\_weights$ **do**
15: Append $weights[key]$ to $layer\_updates$
16: **end for**
17: Compute average weight:
$weights\_prime[key] = \frac{1}{num\_examples\_total} \sum_{i=1}^{n} layer\_updates[i]$
18: **end for**
19: **return** $weights\_prime$

The distinctive aspect of the aggregate fr algorithm lies in its careful treatment of tree-based models. As these models often involve intricate hierarchical structures, preserving their architecture during aggregation is crucial for maintaining accurate and meaningful updates. To achieve this, the algorithm iterates through the keys of the dictionaries within weighted weights. For each key, representing a model layer or node, it aggregates the corresponding weights contributed by clients and calculates their average, while considering the number of examples used by each client. This approach ensures that the complex decision tree structures are adequately reflected in the aggregated model weights.

In comparison to the more succinct aggregate algorithm, which uses list comprehensions and zipped lists for averaging, aggregate fr takes a more nuanced and model-specific approach. While aggregate provides a concise way of averaging weights, it lacks the tailored treatment necessary for tree-based models. This key difference makes aggregate fr stand out as the algorithm of choice when dealing with tree-based ensembles, ensuring that the intricacies of decision tree architectures are preserved during the aggregation process. In essence, the aggregate fr algorithm excels in its ability to maintain the integrity of tree-based models while aggregating weights, setting it apart from the more generalized approach of the aggregate algorithm.

This Design sections gives the overall blueprint for the second research question of how federated learning can be implemented to tree-based models. It highlights key modifications to the FLWR framework that can help use it on models like LightGBM and Catboost.

Chapter 4

Implementation

The implementation of this thesis involved four experiments, each focused on a specific combination of models and target variables. The selected models for the experiments were LightGBM for PV production and consumption predictions, as well as Catboost for the same predictions. The experimental setup revolved around using the FLWR framework, with individual client files specific to and located inside each NVIDIA Jetson Nano device, and a common FLWR server file residing in the HP Elitebook CPU.

4.1 Federated Setup

For this research federated setup was a well-thought-out configuration that leveraged the power of four NVIDIA Jetson Nano devices, each of which acted as a client, and an HP EliteBook CPU, which served as the central server as shown in 3.4.1. Each of these NVIDIA Jetson Nano devices acted independently as a client. The selection of these edge devices was deliberate and strategic, taking into consideration their robust GPU capabilities, which make them exceptionally well suited for computationally intensive activities such as the training of machine learning models. In addition, because of their small size and energy-efficient design, they were ideal for edge computing, a type of computing that places significant emphasis on the efficient use of resources and power.

A comprehensive setup procedure was carried out on each Jetson Nano client prior to the beginning of the federated learning process. Following the installation of the Ubuntu 18 image, we decided to use the Archiconda package manager rather than the Anaconda one. Because of this decision, package management was simplified, and we were able to integrate all of the essential machine learning libraries and frameworks that were required for the prediction task without any problems.

The setup linked all of the Jetson Nano computers to a neighborhood Wi-Fi network and set up SSH connections in order to make it easier for the clients and server to communicate

with one another and work together. This ensured that the data exchange during the federated learning process was both smooth and secure, which is essential for maintaining the confidentiality and safety of the data.

Jetson Nanos	Data
Client 1	Prosumer 1 and Prosumer 2
Client 2	Prosumer 3 and Prosumer 4
Client 3	Prosumer 5 and Prosumer 6
Client 4	Prosumer 7

Table 4.1.1: Distribution of Prosumers data among Clients

As can be seen in table 4.1.1, each client device was given its own unique subset of the data that was collected from prosumers to work with. Client 1 was responsible for the data of prosumers 1 and 2, Client 2 was responsible for the data of prosumers 3 and 4, Client 3 was responsible for the data of prosumers 5 and 6, and Client 4 was responsible for the data of prosumer 7. We divided prosumers like this to accomdate seven prosumers in four jetson nanos. For real world implementation each prosumer representing a household shall have its own edge device, either jetson nano or jetson xavier etc as the selection of edge device will not effect the overall framework much as long as they support flwr, LightGBM and Catboost dependencies. This distribution ensured that data pertaining to households were kept separate and were only used for the training of local models. We were able to achieve a setup that respected our clients' privacy and allowed them to contribute to the model without disclosing sensitive information about other households thanks to the partitioning of the data in the manner described above. Moreover, this specific distribution of prosumers to edge devices was done based on its better model fitting shown more in detail in chapter 5 in Figures 5.3.6 and 5.3.7.

The central processing unit (CPU) of the HP EliteBook acted as the server and played an essential part in the coordination of the federated learning process. It then performed model aggregation, which was a crucial step in the process of building a global model that encapsulated the knowledge from all of the participating households without compromising data privacy. It did this after receiving the locally trained models from the customers. Model aggregation is a thoughtfully crafted method that combines the updates sent in by each client without compromising their individual privacy. By taking this approach, the server was able to derive useful insights from the collective intelligence of all edge devices without accessing individual data, thereby ensuring that privacy regulations were adhered to.

The tremendous potential of federated learning in real world applications was demonstrated by the combination of the FLWR (Federated Learning with Weights and Biases) framework and NVIDIA Jetson Nano devices. The FLWR framework, with its effective communication protocol, allowed for the collaboration and aggregation of local

models to take place in a seamless manner across all of the edge devices. The fact that federated learning preserves users' privacy was an extremely helpful feature, as it made certain that sensitive data pertaining to households remained confidential and was safeguarded.

In addition, the computational capabilities of the Jetson Nano clients were an essential factor in the successful completion of the federated learning tasks. The GPU acceleration that was made available by the Jetson Nano devices sped up the process of training models and aggregating their results, which cut down on the total amount of time needed for federated learning iterations.

The demonstration of the value of edge devices and federated learning in the context of addressing critical challenges in sustainable energy management was made possible by the successful implementation of federated learning for the purpose of predicting PV production and consumption. The findings of the research showed that the distributed and privacy-preserving nature of federated learning, when combined with the computational power of edge devices, could provide a solution that is both efficient and effective for data-driven energy predictions in smart grids.

This setup brought to light the potential of federated learning in a variety of other fields where protecting data privacy and maximizing computational efficiency are of the utmost importance. This research highlighted the significance of edge devices like the NVIDIA Jetson Nano in the context of federated learning. The use of edge devices in real-world applications is becoming more widespread, and one example is the NVIDIA Jetson Nano.

In general, the combination of edge devices and federated learning presents a promising pathway for the development of scalable machine learning solutions that also protect users' privacy. Federated learning will play a pivotal role in unlocking the potential of distributed data for knowledge discovery while ensuring individual privacy as the world moves into the era of the Internet of Things (IoT) and edge computing. This will be accomplished as the world prepares for the IoT and edge computing. The collaboration of edge devices and central servers, which is made possible by federated learning frameworks such as FLWR, paves the way for a future in which intelligence, efficiency, and privacy are not mutually exclusive concepts.

4.2 GridSearchCV

```
1 self.lgb_lower = LGBMRegressor(alpha=lower_quantile, boosting_type='gbdt',
     objective='quantile', metric='quantile')
2 self.grid_search_lower = GridSearchCV(self.lgb_lower, params, cv=5, n_jobs=-1)
3 self.lgb_upper = LGBMRegressor(alpha=upper_quantile, boosting_type='gbdt',
     objective='quantile', metric='quantile')
```

```
self.grid_search_upper = GridSearchCV(self.lgb_upper, params, cv=5, n_jobs=-1)
self.lgb_mean = LGBMRegressor(alpha=0.5, boosting_type='gbdt', objective='
    quantile', metric='quantile')
self.grid_search_mean = GridSearchCV(self.lgb_mean, params, cv=5, n_jobs=-1)
```

Listing 4.1: GridSearchCV applied to LightGBM

We can see in the code above that for both Catboost and LightGBM, we take three models that gives us lower quantile, upper quantile, and mean. After the GridSearchCV process was complete, we selected the best-tuned versions of LightGBM and CatBoost as the final models for our federated learning task. The selected models were instrumental in enabling accurate and reliable predictions, contributing to the effectiveness and efficiency of the smart energy management system for residential prosumers.

4.3 Clients

The first experiment aimed to predict PV production using the LightGBM model. The dataset containing the measurements from different households in Uppsala, Sweden, was preprocessed and divided into chunks according to table 4.1.1. Each Jetson Nano device acted as a client, holding a specific combination of two prosumers per client from prosumer data. This was done to accomodate seven prosumers with four jetson nanos, and The LightGBM client application on each Jetson Nano trained a local model using its respective data chunk. During the training process, the models communicated only with the central FLWR server on the HP Elitebook CPU to exchange model parameters. This ensured that no raw data was shared, guaranteeing data privacy.

```
def fit(self, parameters, config):
    if parameters:
        weight1 = parameters[0].item()
        weight1['n_estimators'] = int(weight1['n_estimators'])
        weight1['max_depth'] = int(weight1['max_depth'])
        weight2 = parameters[1].item()
        weight2['n_estimators'] = int(weight2['n_estimators'])
        weight2['max_depth'] = int(weight2['max_depth'])
        weight3 = parameters[2].item()
        weight3['n_estimators'] = int(weight3['n_estimators'])
        weight3['max_depth'] = int(weight3['max_depth'])

        self.grid_search_lower = LGBMRegressor(**weight1, alpha=0.25,
    boosting_type='gbdt', objective='quantile', metric='quantile')
        self.grid_search_upper = LGBMRegressor(**weight2, alpha=0.75,
    boosting_type='gbdt', objective='quantile', metric='quantile')
        self.grid_search_mean = LGBMRegressor(**weight3, alpha=0.5
    boosting_type='gbdt', objective='quantile', metric='quantile')

    self.grid_search_lower.fit(self.x_train, self.y_train.ravel())
```

```
18      self.grid_search_upper.fit(self.x_train, self.y_train.ravel())
19      self.grid_search_mean.fit(self.x_train, self.y_train.ravel())
20
21      .....
22
23      return [self.grid_search_lower.best_params_,
24              self.grid_search_upper.best_params_,
25              self.grid_search_mean.best_params_], len(self.x_train), {}
```

Listing 4.2: Model Fit fuction for FLWR client for LightGBM Production (Experiment 1)

The provided Python code snippet encapsulates a method named fit within a client class, which serves the purpose of training and evaluating a set of three LightGBM models using a grid search strategy. Upon receiving parameters, which represent model weights, the method initializes the best estimators of lower, upper, and mean quantile models using these weights. Subsequently, the models undergo training utilizing the x train and y train data.

The trained models are then employed to predict outcomes on the x test dataset. To ensure meaningful interpretation, the predictions are transformed back to their original scale using an inverse transformation performed by scaler y.inverse transform. The transformed arrays are flattened for subsequent computations.

A suite of evaluation metrics, encompassing measures such as CFE, MQL, PIR, and MAE, is computed through the utilization of the error metrics function. These metrics serve to quantify the predictive performance of the models and provide insights into their effectiveness.

Two pandas DataFrames are established to structure and store the results. The first DataFrame, named df production, encompasses columns for true values, predicted values, interval lower bounds, interval upper bounds, mean quantile losses, and mean absolute errors. The second DataFrame, df production results, captures the computed evaluation metrics.

Subsequently, the calculated results are preserved as CSV files. The df production DataFrame, housing prediction intervals and associated metrics, is stored in the file named lightgbm prod client1.csv. Meanwhile, the df production results DataFrame, encapsulating comprehensive evaluation outcomes, is saved in the file lightgbm prod Evaluation client1.csv.

The method concludes by returning a list containing the optimal parameters for the three distinct models, along with the length of the training dataset, denoted by len(self.x train), and an empty dictionary. It is evident that this fit method not only orchestrates the training and evaluation process for LightGBM models but also meticulously records and presents the results in a structured and informative manner.

The second experiment focused on predicting PV consumption using the LightGBM model. Similar to Experiment 1, the dataset was divided into chunks, and each Jetson Nano served as a client, holding specific prosumer data.

The third experiment aimed to predict PV production using the Catboost model, and the fourth experiment focused on predicting PV consumption using the Catboost model. Similar to Experiment 1, all the experiments' clients used same approach for model fitting.

Each FLWR client has a client class with fit being the main function that outputs model parameters to the server. Normally FLWR supports a single model per client, but in our implementation shown above we are getting parameters for three models: lower quantile, upper quantlie, and mean. For all three models, we then fit and make predictions according to work inspired from [3]. The new models parameters of all three models are sent to server in return along with size of training data. This is part of this research's contribution that each client support multiple models that is explained in detail in section Multi-model Tree Aggregation.

4.4 Server

The FLWR server application on the HP Elitebook CPU managed the aggregation of model parameters received from the Jetson Nano clients during training. It ensured that the global model was updated based on the aggregated parameters without compromising the privacy of individual prosumer data. During the inference phase, federated inference was employed, allowing each Jetson Nano device to make predictions locally using the trained global model without sharing raw data.

```
class FedModelstrategy(fl.server.strategy.FedAvg):
    def __init__(self, min_fit_clients=4, min_available_clients=4):
        super().__init__()
        self.min_fit_clients = 4
        self.min_available_clients = 4

    def aggregate_fr_fit(self, rnd, results, failures):
        # Call aggregate_fit from base class (FedAvg) to aggregate parameters and metrics
        aggregated_parameters, aggregated_metrics = super().aggregate_fr_fit(rnd, results, failures)

        if aggregated_parameters is not None:
            # Save aggregated_ndarrays
            print(f"Saving round {rnd} aggregated_ndarrays...")
            np.savez(f"model/round-{rnd}-weights.npz", *aggregated_parameters)
            # Save each individual model
            for idx, model_weights in enumerate(aggregated_parameters):
```

```
                model_weights= np.reshape(model_weights, (1,))
                model_path = f"model/round-{rnd}-model-{idx}.txt"
                # Save the text data to a file
                np.savetxt(model_path, model_weights, fmt="%s")

        return aggregated_parameters, aggregated_metrics


strategy = FedModelstrategy(min_fit_clients=4, min_available_clients=4)

fl.server.start_server(
    server_address="localhost:8080",
    config=fl.server.ServerConfig(num_rounds=10),
    strategy=strategy
)
```

Listing 4.3: FLWR Server code

The provided Python code segment introduces a custom strategy class named FedModelstrategy, intended for orchestrating Federated Learning (FL) processes within a server environment. Derived from the base class fl.server.strategy.FedAvg, this class offers a tailored approach to aggregating model parameters and metrics during FL rounds.

In the constructor method (init), the custom strategy initializes with default values of min fit clients and min available clients both set to 4. These values, though hard-coded in the current implementation, are likely to signify the minimum clients needed for model fitting and the minimum number of available clients, respectively.

The overridden aggregation method, aggregate fr fit, extends the behavior of the base class. It leverages the base class's aggregation mechanism, yielding aggregated parameters and aggregated metrics. When aggregate parameters are present, the method embarks on preserving the aggregated model parameters and the individual models contributed by each client. By iterating through the aggregated parameters, it stores the model weights both collectively and individually, utilizing numpy functions. The collected parameters are saved in a .npz file, while individual client models find their place in separate .txt files.

Moreover, the code sets up an instance of the custom strategy (FedModelstrategy) by initializing it with specified min fit clients and min available clients parameters. Subsequently, the FL server launches through the fl.server.start server function. This server operates with a configuration indicating 10 rounds of FL, and it integrates the custom strategy into the FL process.

In essence, this code establishes a customized aggregation strategy tailored to federated learning scenarios. By enhancing the aggregation process and facilitating the storage of model parameters and individual client models, the strategy enriches the FL framework

with more intricate insights into model evolution and individual contributions across rounds.

Normally aggregate fit is available function that supports aggregation of clients containing a single model inside. For use case of this research aggregate fr fit is introduced that allows three models per client and aggregate accordingly, and save model weights for each round. It is explained in detail in next section.

4.5 Multi-model tree Aggregation

We propose aggregate fr fit function in FedAVg strategy mentioned in background and literature review chapter, where fr in name to refer to tree-based/forest-based models. It supports multiple models inside a single client, as opposed to normal aggregate fit function. This is the main contribution of this thesis, and this support tree-based models as the aggregation fr function shown later is customized to support dictionary iteration of model parameters that are specific to tree-based models as opposed to deep learning models.

```
def aggregate_fr_fit(
        self,
        server_round: int,
        results: List[Tuple[ClientProxy, FitRes]],
        failures: List[Union[Tuple[ClientProxy, FitRes], BaseException]],
    ) -> Tuple[Optional[Parameters], Dict[str, Scalar]]:
        """Aggregate fit results using weighted average."""
        if not results:
            return None, {}
        # Do not aggregate if there are failures and failures are not accepted
        if not self.accept_failures and failures:
            return None, {}

        # Convert results to a suitable format for aggregation
        weights_results_1 = []
        weights_results_2 = []
        weights_results_3 = []
        for _, fit_res in results:
            client_weights = fit_res.parameters
            num_examples = fit_res.num_examples

            # Convert client_weights to a list of model weight ndarrays
            model_weights_ndarrays = [parameters_to_ndarrays(weights) for
    weights in [client_weights]]
            weights_results_1.append((model_weights_ndarrays[0][0],
    num_examples))
            weights_results_2.append((model_weights_ndarrays[0][1],
    num_examples))
```

```
            weights_results_3.append((model_weights_ndarrays[0][2],
    num_examples))

        parameters_aggregated = [aggregate_fr(weights_results_1),
                                 aggregate_fr(weights_results_2),
                                 aggregate_fr(weights_results_3)]
        # Aggregate custom metrics if aggregation fn was provided
        metrics_aggregated = {}
        if self.fit_metrics_aggregation_fn:
            fit_metrics = [(res.num_examples, res.metrics) for _, res in
    results]
            metrics_aggregated = self.fit_metrics_aggregation_fn(fit_metrics)
        elif server_round == 1:  # Only log this warning once
            log(WARNING, "No fit_metrics_aggregation_fn provided")

        return parameters_aggregated, metrics_aggregated
```

Listing 4.4: Proposed code for aggregate fr fit

The aggregate fr fit function is a crucial component in a federated learning framework where multiple clients participate in training a machine learning model on their local datasets. The function requests several pieces of information as input, including the round of training that is currently being performed on the server (server round), a list of fit results from the clients (results), and a list of potential failures that may have occurred during the process (failures).

The function moves on to the next step after handling these cases, which is to prepare the fit results for aggregation. It goes through the list of results and pulls out the parameters (weights) and the total number of training examples that each client used. These parameters are laid out in the form of three distinct lists, which are denoted as follows: weights results 1, weights results 2, and weights results 3. It can be deduced from the fact that each list corresponds to a distinct part of the model that the model is composed of three distinct components, as we fit models for upper quantile, lower quantile, and mean separately due to limitation of having single alpha value in LightGBM that deals with quantiles.

Using the parameters to ndarrays function, the function transforms each set of parameters into a list of model weight. This helps to guarantee that the aggregation is carried out accurately. The subsequent step is to add these NDArrays, along with the number of training examples that corresponds to each one, as tuples to the relevant weights results list.

In addition, the function is capable of handling the aggregation of custom metrics if the appropriate aggregation function (self.fit metrics aggregation fn) is supplied. It does this by aggregating the custom metrics that were produced as a result of the fit, producing a dictionary in which the name of each metric corresponds to the value of the aggregated

scalar metric.

The function then concludes by returning a tuple that contains the aggregated parameters for each component of the model as well as the aggregated custom metrics in the form of a dictionary. These aggregated results provide a more complete picture of the performance of the model because they take into account the contributions made by the training results of a variety of clients while still maintaining respect for the individual sizes of each client's dataset.

Note that the aggregate fr function and the parameters to ndarrays function are not described in any specific detail in the provided code snippet. It is presumed that these functions are implemented in server application. It is important to take this into consideration. In addition, the partitioning of the model into three sections (weights results 1, weights results 2, and weights results 3) might be unique to the particular federated learning scenario, and it might also change depending on the architecture and requirements of the model.

In addition to this we also added aggregate fr function called inside aggregate fr fit that for each model aggregates the model weights. We have created this as for normal aggregate function it supports layer by layer multiplication of number of examples to calculate average later on, and is feasible for Deep Learning models as weights are stored in matrix form. For tree-based models the parameters are stored in dictionary form and traditional aggregate function could not support it. With aggregate fr function we can easily aggregate the parameters of forest based models

```
def aggregate_fr(results: List[Tuple[NDArrays, int]]) -> NDArrays:
    """Compute weighted average."""
    # Calculate the total number of examples used during training
    num_examples_total = sum([num_examples for _, num_examples in results])
    weighted_weights = []
    for weights, num_examples in results:
        weights = dict(weights.item())
        for key in weights:
            weights[key] *= num_examples
        weighted_weights.append(weights)

    # Compute average weights of each layer
    weights_prime = {}
    for key in weighted_weights[0]:
        layer_updates = [weights[key] for weights in weighted_weights]
        weights_prime[key] = np.sum(layer_updates) / num_examples_total

    return weights_prime
```

Listing 4.5: Proposed code for aggregate fr

The aggregate fr function plays a crucial role in federated learning. The function takes

in a list of tuples, where each tuple contains two components: an NDArray representing the locally trained weights for a specific layer in the model, and an integer indicating the number of training examples used to train those particular weights. The list of tuples is passed into the function as an argument.

The first thing that the function does is compute the overall number of training examples that have been applied across all of the servers and devices. This step is essential because it gives the function the ability to weight each locally trained set of weights according to the number of examples that were used to train them. The function is able to give more significance to models that have been trained on larger datasets by taking into account the number of examples, which ensures that these models contribute more to the aggregated weights in the end.

The function then moves on to the next step, which is to calculate the weighted weights for each set of locally trained weights. It does this by multiplying each weight value by the number of training examples that correspond to that weight value for each set of weights. This process basically adjusts the weights so that they are proportional to the respective sizes of the datasets. As a result, it ensures that models that have been trained on more data have a greater impact on the aggregated weights.

Following the completion of the computation of the weighted weights, the function will proceed to determine the average weight for each layer in the model. It accomplishes this by first totaling the weighted weights for each layer across all of the locally trained models and then dividing that total by the total number of training examples. In other words, it adds up all of the weights and then divides them by the total number of training examples. The result of this computation is a weighted average of the weights for each layer. This takes into account the contribution that each locally trained model makes based on the size of its dataset.

The function produces a dictionary with the name weights prime as its final output. Within this dictionary, each key denotes a layer in the model, and the value that corresponds to it is the weight that is considered to be the average for that layer across all of the locally trained models. Because they take into account the aggregated insights from all of the devices or servers that were involved in the training process, these aggregated weights are more reliable and accurate in describing the performance of the overall model.

This answers the second research question that it is possible to implement federated learning on tree-based models like LightGBM and Catboost, and this modification to the FLWR framework enables it.

Chapter 5

Evaluation and Result

This chapter will display the results achieved from federated learning for LightGBM and Catboost and the evaluation of the results.

5.1 Federated Inference

During the course of the research, the idea of federated learning was investigated. This led to the incorporation of the unique method of federated inference, which was designed to improve the predictive capabilities of the system while maintaining the highest possible levels of data privacy and security. Utilizing the FLWR framework as the basis for the construction of the federated learning system allowed for the harnessing of the computational power offered by edge clients in the form of NVIDIA Jetson Nano devices. The research was expanded to leverage federated inference for the prediction of photovoltaic (PV) production and consumption. This made it possible for edge devices to use the trained global model to make predictions on their locally held data without having to share any raw data with the central server.

Federated Inference being one of the main components of this thesis is developed in a way to ensure efficiency. Every client application' class has a fit function that ensures fitting of the updated model weights to produce predictions, and also send back new fitted parameters back to the server. More detailed description in last section. During this fit function, this research gets the predictions for that specific client from model trained on it in that round and saves it to the results with identifier of client name, tree-based model name. This same procedure is done to save evaluation results for that specific client trained and tested on the dataset of prosumers specific to that client.

Moreover for every round, model weights are aggregated and stored in models directory that is accessed in separate notebook that recreates the model on aggregated weights, and

test it on testing data of all prosumers concatenated to produce predictions and evaluations result for combined global model.

The incorporation of federated inference into the system resulted in the system gaining a number of important benefits. The most important benefit was ensuring the confidentiality of the data, which is an essential concern when working with sensitive data from prosumers. The risk of data exposure was effectively reduced by keeping all raw data stored safely on their respective Jetson Nano devices and communicating only model parameters with the centralized server. The fact that the prosumers' data remained secure and under their control despite the implementation of this privacy-preserving mechanism instilled in them a sense of trust and confidence.

Additionally, the federated inference approach showed remarkable scalability in its implementation. Because the global model could be easily deployed to multiple edge devices, it was able to be adapted to accommodate a greater number of prosumers in real-world scenarios. Because of its scalability, the system was able to accommodate the growing demands placed on it by applications for energy management and effectively cater to the varying requirements of its users.

5.2 Predictions

In this analysis, the performance of LightGBM and Catboost models for predicting PV power production and consumption using federated learning is evaluated. The evaluation is carried out by contrasting the predicted values of the models with the actual values of PV power production and consumption at the Mean, Upper Quantile, and Lower Quantile levels respectively. The training size for all client combined consisted of 61032 rows and test size consisted of 15261 rows for production experiments, and training size for all client combined consisted of 103394 rows and test size consisted of 25851 rows for consumption experiments. The true values for the entire dataset was already provided by authors of [3].

The results of the LightGBM PV power production prediction for the global model are displayed in the figure referenced as 5.2.1. It has been observed that the predictions, up to 8000 W, follow a linear trend with the true values. This suggests that the model's predictions are reasonably accurate within this range. However, once 8000 W is exceeded, the predictions begin to deviate from the actual values, which suggests that the model may be less reliable when attempting to predict higher power values. In spite of this divergence, it has been observed that the predictions for the Upper Quantile and the Mean continue to be relatively proportional. This suggests that they provide estimates that are consistent with regard to the central tendency of the data.

In a similar manner, the LightGBM PV power consumption prediction is broken down

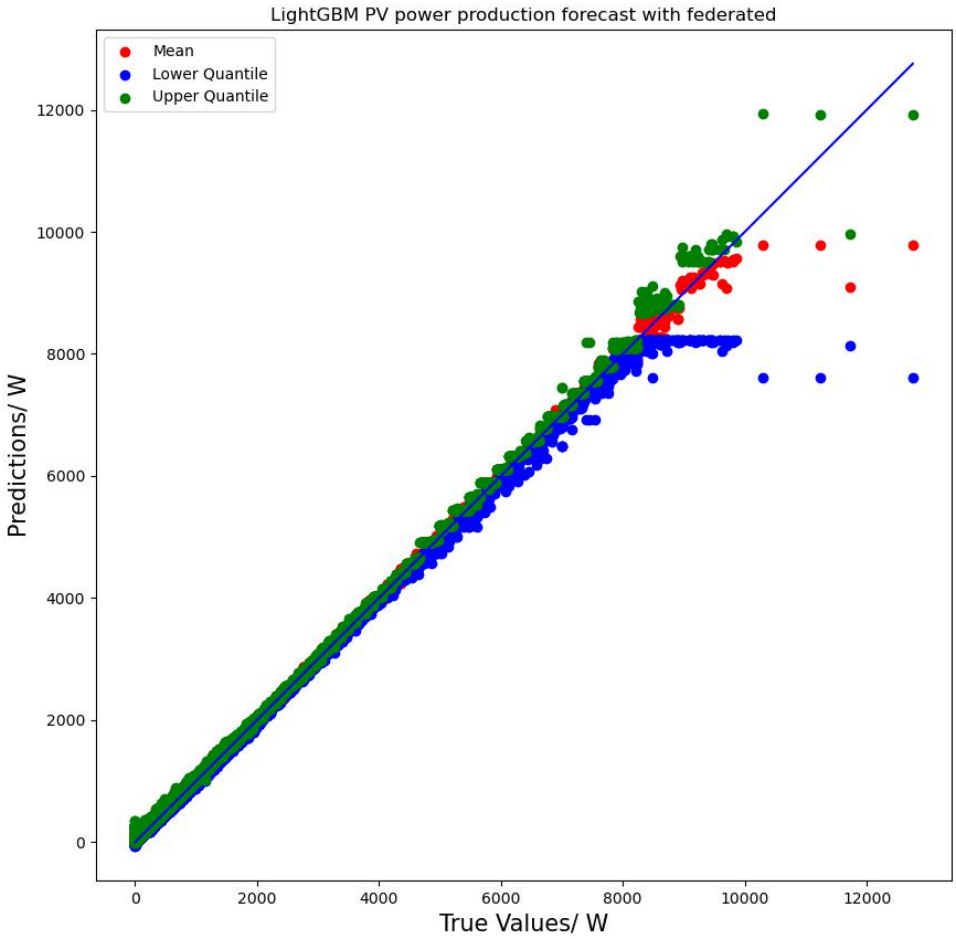

Figure 5.2.1: LightGBM production prediction plot

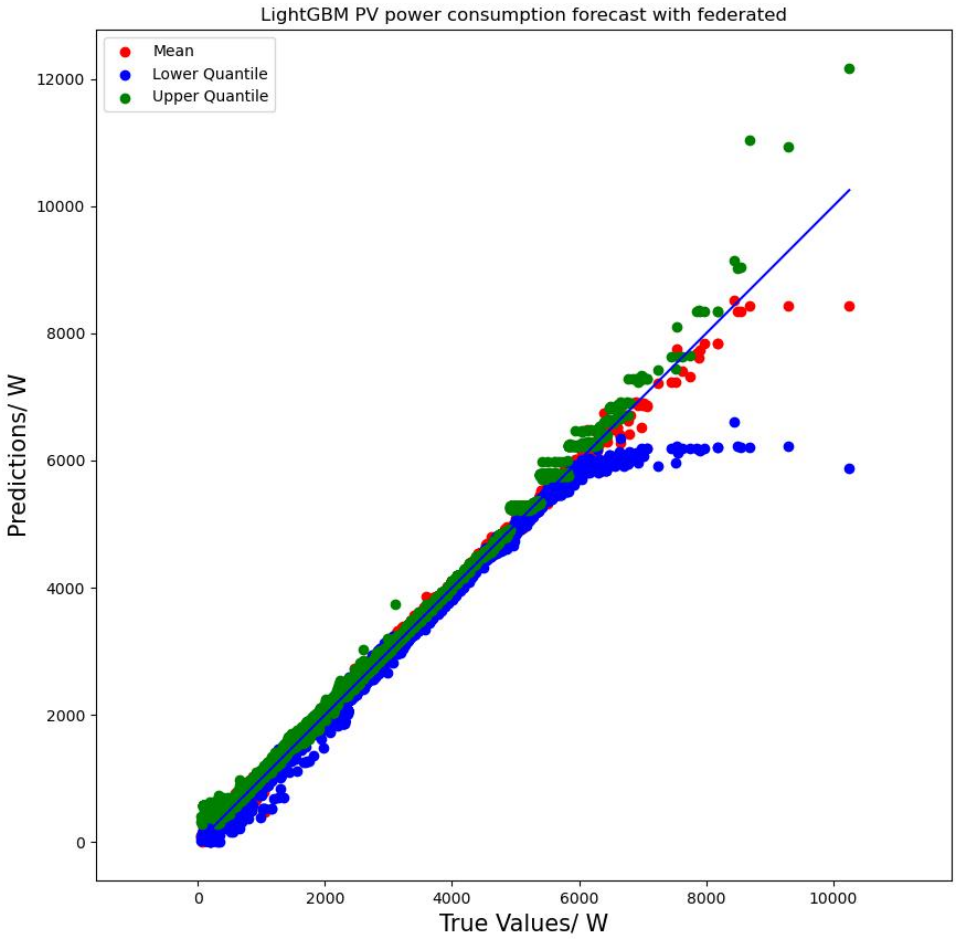

Figure 5.2.2: LightGBM consumption prediction plot

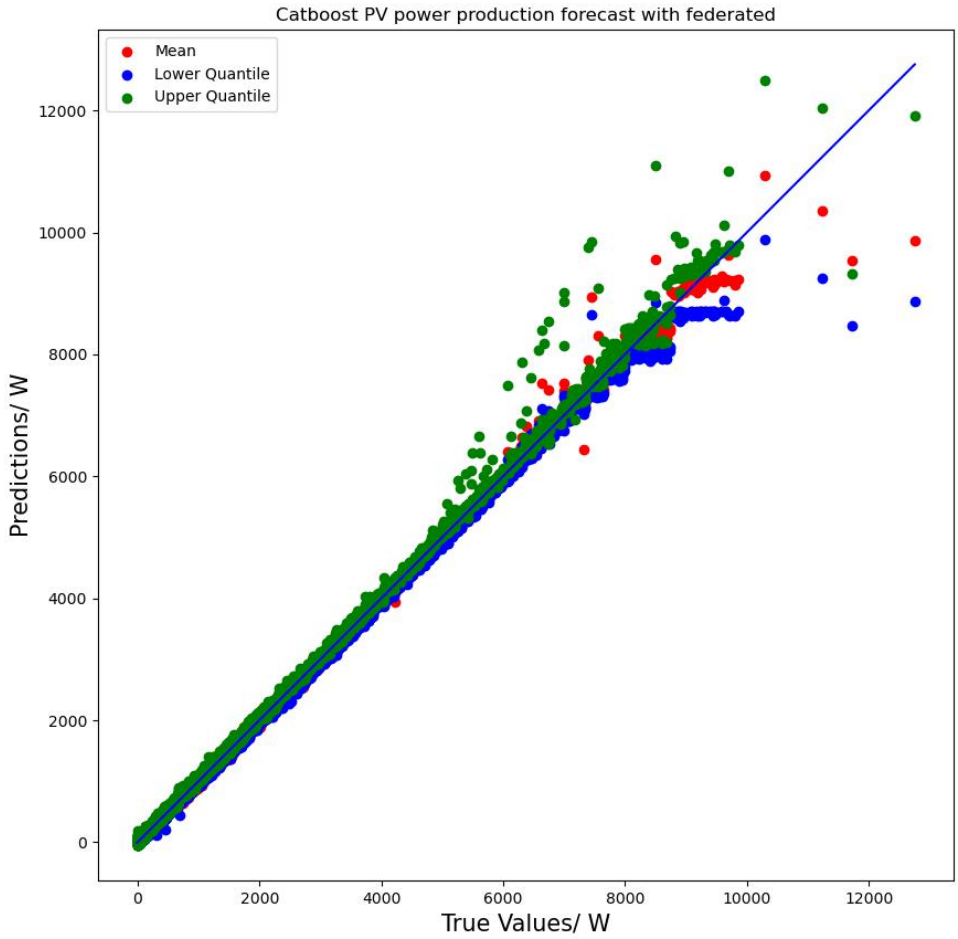

Figure 5.2.3: Catboost production prediction plot

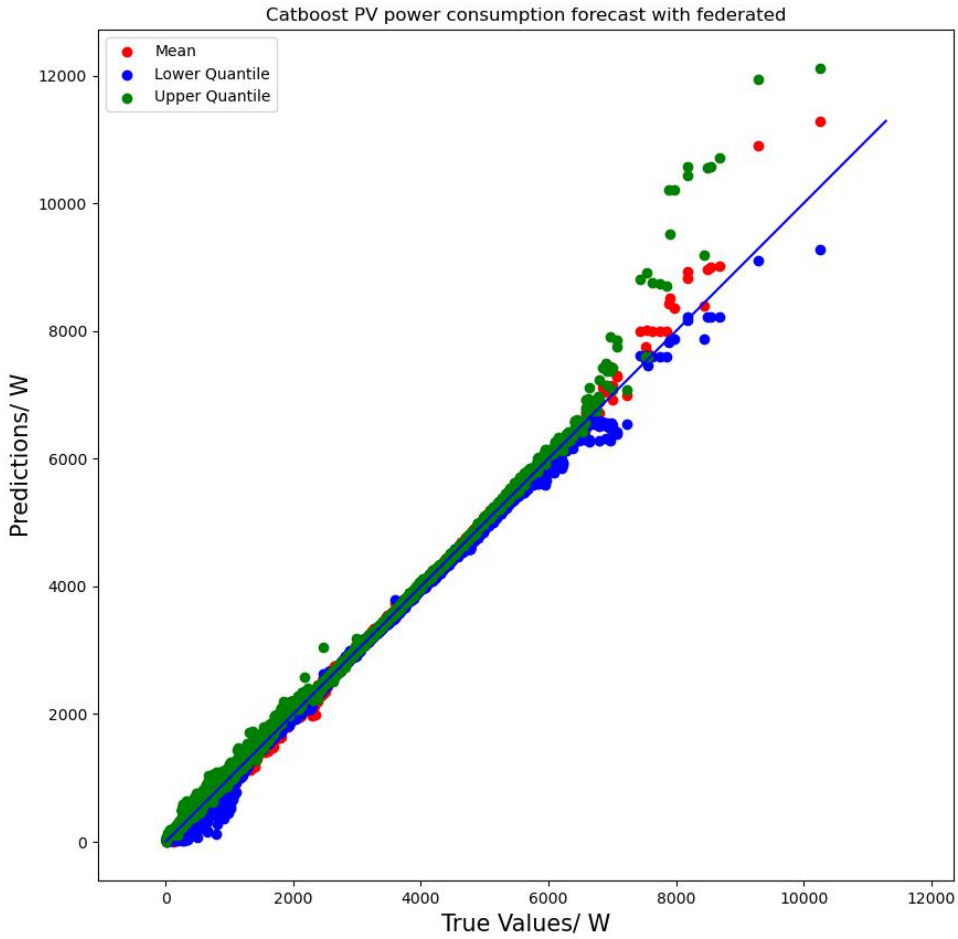

Figure 5.2.4: Catboost consumption prediction plot

and examined in figure 5.2.2. Up to 6000 W, the predictions of the model show a linear relationship with the actual values, which indicates accurate performance within this range. After 6000 W, the predictions begin to depart from the actual values in a noticeable manner. However, just like in the case of power production, the Upper Quantile and Mean predictions continue to maintain a relationship that is close to proportional, which suggests that central tendency estimates are consistent.

Moving on to the results of the Catboost model, the PV power production prediction is shown in the figure that is referenced as 5.2.3. It has been discovered that the predictions maintain a linear relationship with the actual values up to 10000 W, which indicates that accurate predictions can be made within this range. However, when the power values are increased, the predictions begin to differ from one another. Despite this, the Mean and Lower Quantile predictions exhibit a relationship that is close to proportional, which indicates that the estimates of the central tendency are consistent.

Last but not least, the predicted amount of power used by Catboost PV is shown in figure 5.2.4. Up to 7000 W, the predictions of the model are linear with the true values, which indicates that the model's performance is accurate within this range. After 7000 W, the predictions begin to vary from one another. In a manner analogous to that of power production, the predictions for the Mean and Lower Quantile continue to be close to proportional, which suggests that central tendency estimates are consistent.

The Upper Quantile and Mean predictions consistently maintain a proportional relationship, which suggests that there is stability in estimating the central tendency of the data across different power values. This is an important observation that comes from all four experiments, and it is one that is worth noting. In a manner parallel to this, the Lower Quantile and Mean predictions both display a relationship that is close to proportional in the Catboost experiments.

In conclusion, the findings of these experiments indicate that the LightGBM and Catboost models have an encouraging performance when it comes to predicting the amount of power produced and consumed by PV systems using federated learning. Within certain power ranges, the models have a high degree of accuracy; however, when the power is increased, the models' predictions no longer agree with one another. In spite of this, the fact that there is a consistent relationship between the Upper Quantile and Mean predictions as well as the Lower Quantile and Mean predictions demonstrates that the models provide reliable estimates for the central tendency of the data. To evaluate the models' generalization and robustness across a variety of scenarios and datasets, however, additional research and validation are required.

5.2.1 Variations based on weather forecast

Do different weather phenomena impact PV Power Production and forecasting error? To answer this in this section the variation in time series forecasting by LightGBM is shown for PV power production for month of July 2022. This specific month was selected due to varying weather conditions every day from the data available. Figure 5.2.5 shows the weather forecast for whole month, and Figure 5.2.6 shows the production forecast by taking mean production value for each day. From these four days were selected for further experimentation and were classified as Rainy, Sunny, Cloudy and Warm, and Cloudy and Cold. For each of these days then PV Power production forecasting was done with LightGBM and Federated LightGBM. Each plot consisted of the True Value, Prediction, and also the Prediction Range

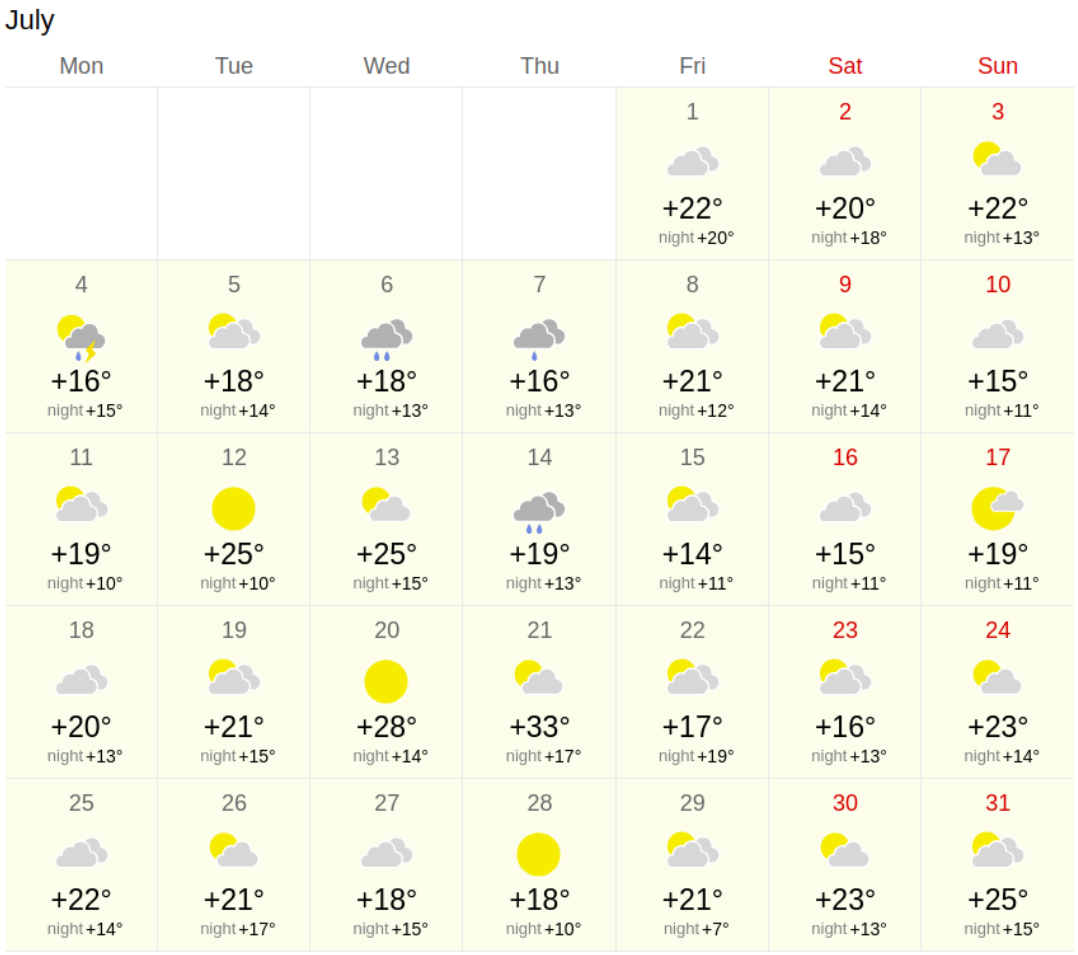

Figure 5.2.5: Weather Forecast for July 2022

From Figures 5.2.7 and 5.2.8, it becomes evident that the PV Power Production forecast for the sunny day exhibited a notably smooth trajectory, a result largely in line with

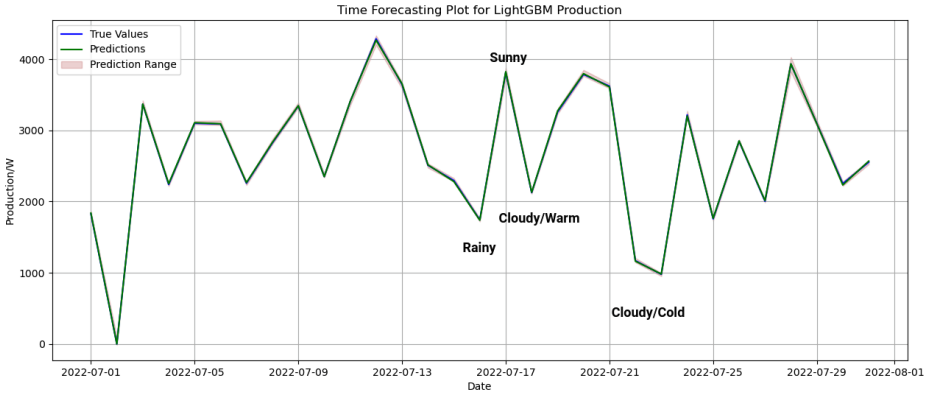

Figure 5.2.6: Time-series Forecasting of LightGBM Production for July 2022

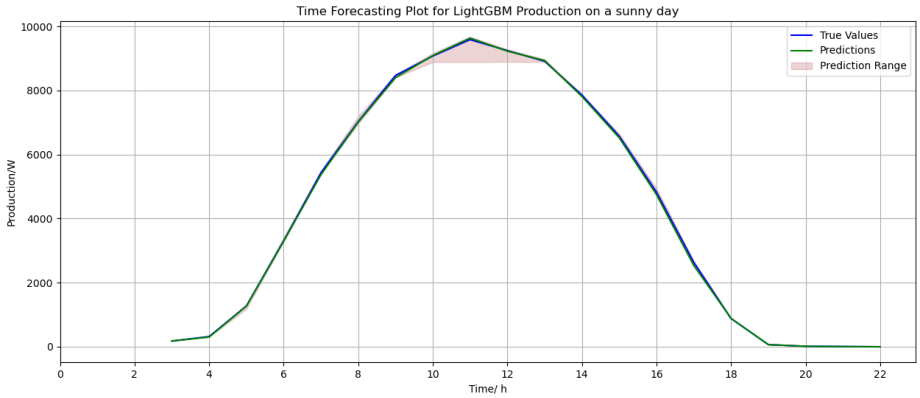

Figure 5.2.7: Time-series Forecasting of LightGBM Production for sunny day

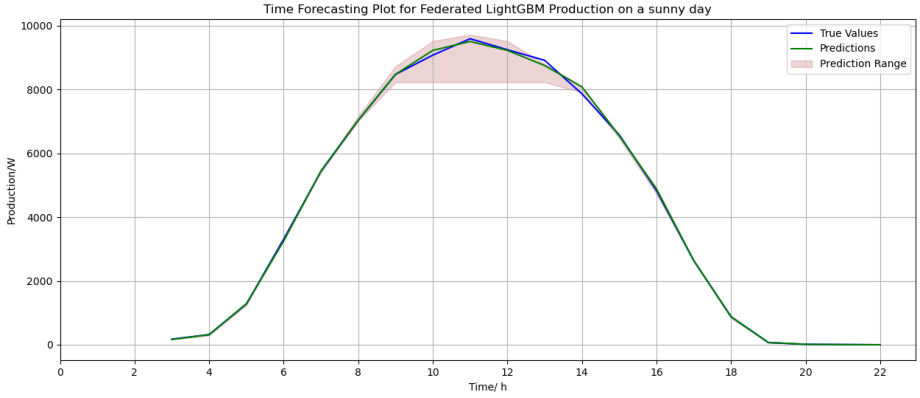

Figure 5.2.8: Time-series Forecasting of Federated LightGBM Production for sunny day

expectations given the clear weather and abundant sunlight available. Both the normal and federated models of LightGBM demonstrated commendable performance, with their predictions closely aligning with the actual values. For federated LightGBM model the prediction range was higher than normal in the peak production period.

Notably, the peak production was observed between 10:00 and 13:00, coinciding with the period of maximum sunlight intensity. This correlation underscores the direct impact of sunlight availability on the production levels, a fundamental aspect in solar power generation.

The day's overall production profile exhibited a continuous, gentle undulation, harmoniously mirroring the ebb and flow of sunlight. This synchronization suggests an effective adaptability of the model to the dynamic nature of solar energy generation. This smooth curve in production indicates a reliable forecasting capability, as it accurately mirrors the inherent variations in sunlight intensity throughout the day.

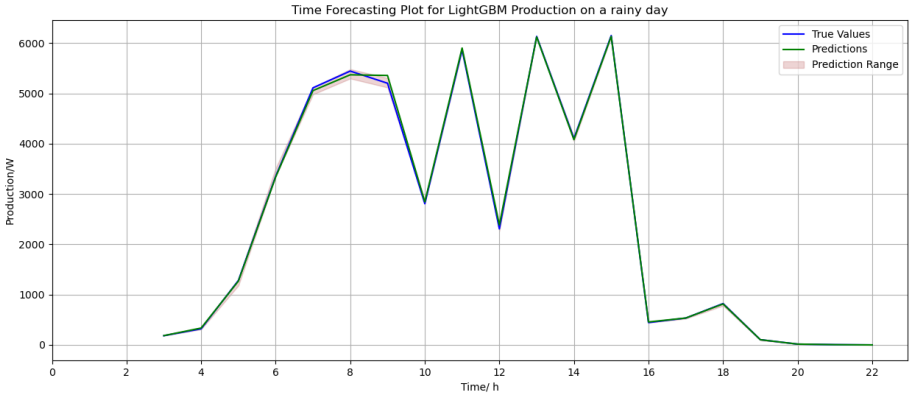

Figure 5.2.9: Time-series Forecasting of LightGBM Production for rainy day

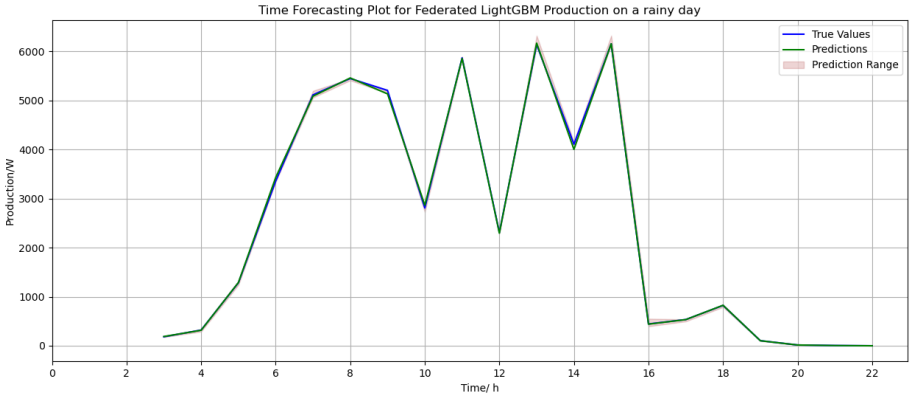

Figure 5.2.10: Time-series Forecasting of Federated LightGBM Production for rainy day

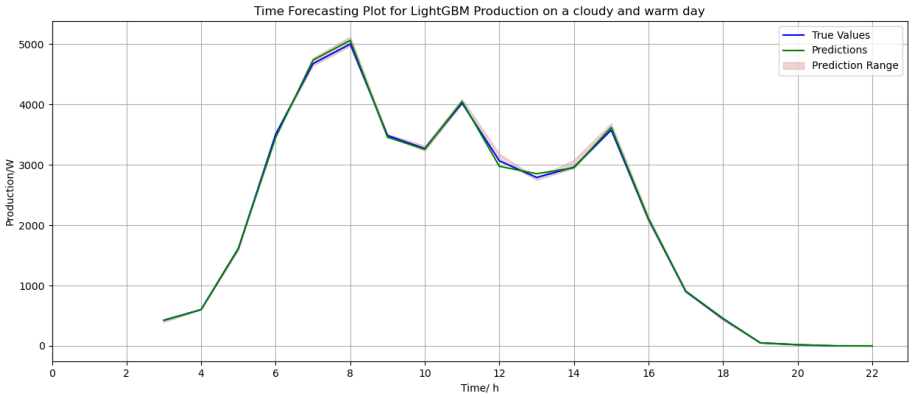

Figure 5.2.11: Time-series Forecasting of LightGBM Production for cloudy and warm day

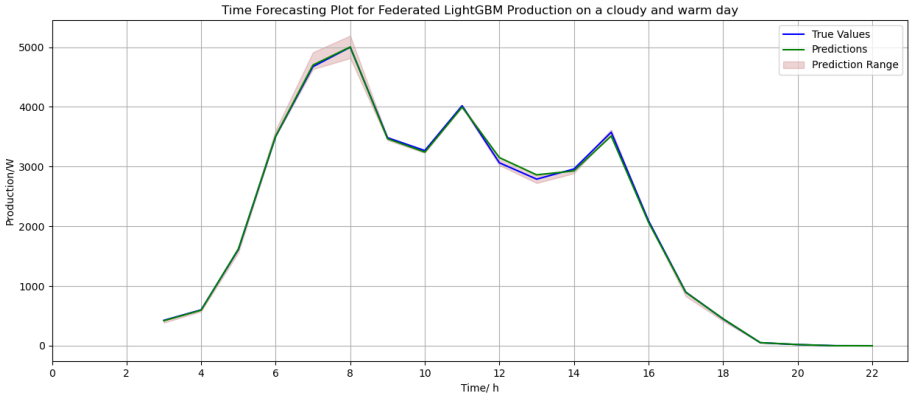

Figure 5.2.12: Time-series Forecasting of Federated LightGBM Production for cloudy and warm day

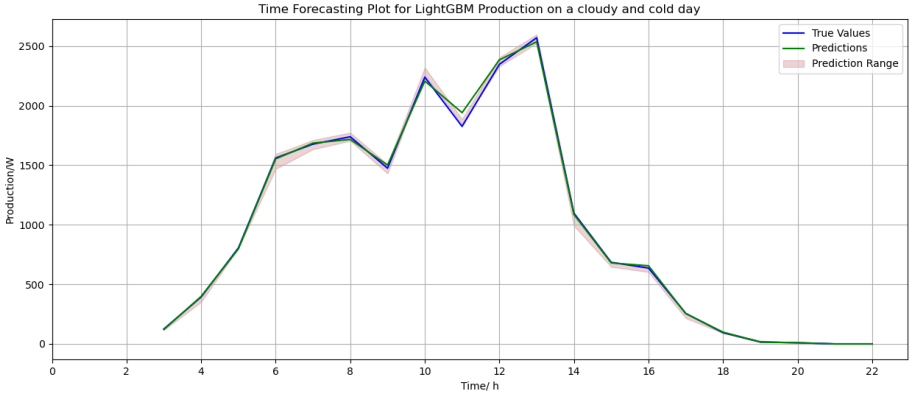

Figure 5.2.13: Time-series Forecasting of LightGBM Production for cloudy and cold day

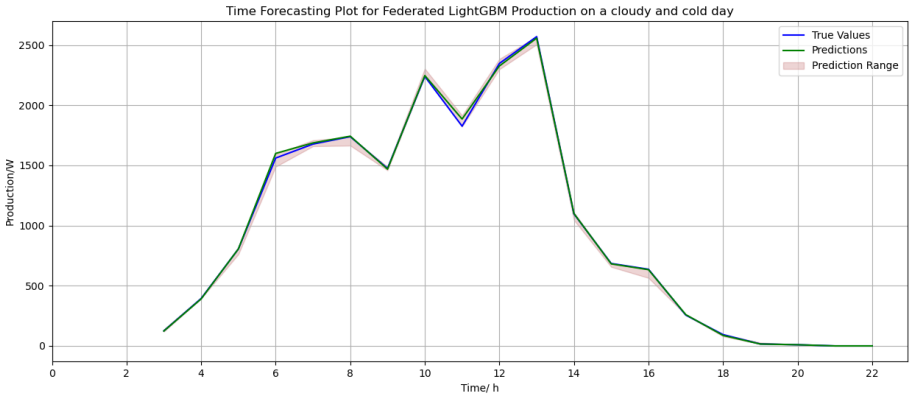

Figure 5.2.14: Time-series Forecasting of Federated LightGBM Production for cloudy and cold day

Across each day, a consistent pattern emerges with the sun rising at approximately 4:00 and setting between 16:00 and 17:00. This temporal regularity is mirrored in the production cycle, commencing and concluding around these same times.

For the rainy day showcased in Figures 5.2.9 and 5.2.10, the cloudy yet warm day depicted in Figures 5.2.11 and 5.2.12, and the cloudy and cold day illustrated in Figures 5.2.13 and 5.2.14, a strikingly similar trend is discernible. These variations predominantly stem from the availability of sunlight at specific intervals.

On the rainy day, the peak production hovers around 6000 W, contrasting with the sun-drenched day where the maximum production reaches approximately 10000 W. The cloudy conditions introduce another layer of variability. When the weather is both cloudy and warm, the highest production registers at about 5000 W. Conversely, under cloudy and cold conditions, the maximum production dwindles to a modest 2500 W. This discrepancy indicates the influence of temperature on power production.

In all the depicted scenarios, both the normal and federated versions of LightGBM exhibit a commendable performance, closely mirroring the true values. However, in complex weather conditions like rainy and cloudy days, the federated model emerges as the more accurate predictor, demonstrating its enhanced adaptability to intricate environmental factors.

5.3 Evaluations

Further analysis was done to determine effectiveness of the federated learning approach. Since this research is based on findings from [3], it used same metrics like Mean Absolute Error (MAE), Mean Prediction Interval Range (MPIR), and Mean Quantile Loss (MQL)

that are explained in detail in chapter 2.

Research Questions

1. How does Federated learning impact forecast accuracy of Multi-Variate Time Series Forecasting in the context of Renewable Energy Systems?

2. How can Federated Learning be implemented on tree-based models?

5.3.1 Probability Distribution Analysis: CDF and PDF Plots

This section encompasses a comprehensive analysis of four distinct models: Catboost, Federated Catboost, LightGBM, and Federated LightGBM. The evaluation process involved approximately 15,263 hourly-based predictions, all conducted on identical training and testing datasets.

Model	Percentage Mean Absolute Forecast Error
Federated LightGBM	1.714
LightGBM	2.610
Federated Catboost	3.318
Catboost	9.328

Table 5.3.1: Percentage Mean Absolute Forecast Error for all models

For each model, a consolidated Probability Density Function (PDF) plot and Cumulative Distribution Function (CDF) plots are presented. These visualizations offer a detailed view of the predictive performance across the dataset, providing valuable insights into the distribution and accuracy of the models' predictions. This aids in the assessment of their respective strengths and weaknesses.

From PDF plots in Figure 5.3.1 it can be seen that with Federated models for both Catoost and LightGBM the distribution of percentage forecast error is significantly less than normal models, indicating better performance of federated learning models.

From CDF plots for LightGBM and Federated LightGBM in Figures 5.3.2 and 5.3.4 the observations remained almost similar for normal and federated versions of LightGBM, where both indicated perfect fit. However, for Catboost and Federated Catboost in Figures 5.3.3 and 5.3.5 there is clear indication of better performance of federated version of Catboost. In this case normal Catboost was underfitting and Federated Catboost had perfect fit.

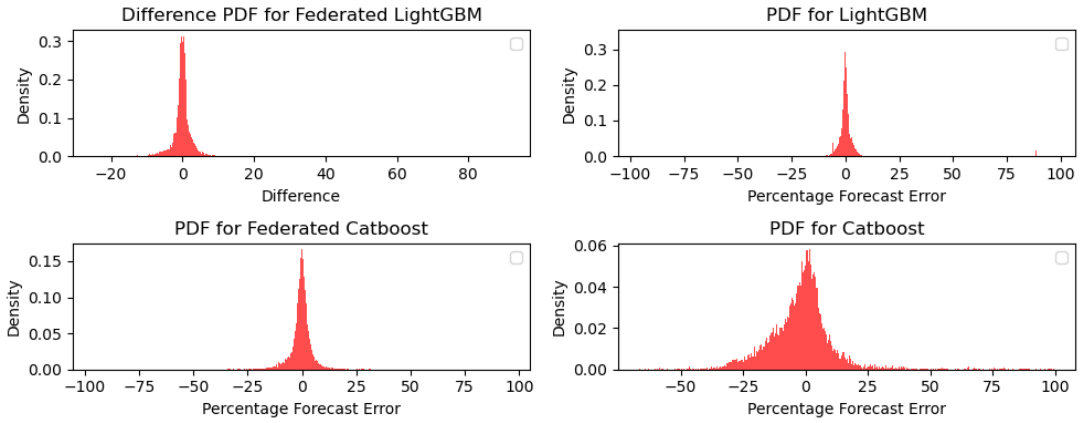

Figure 5.3.1: PDF plots for Catboost, Catboost Federated, LightGBM, and LightGBM Federated

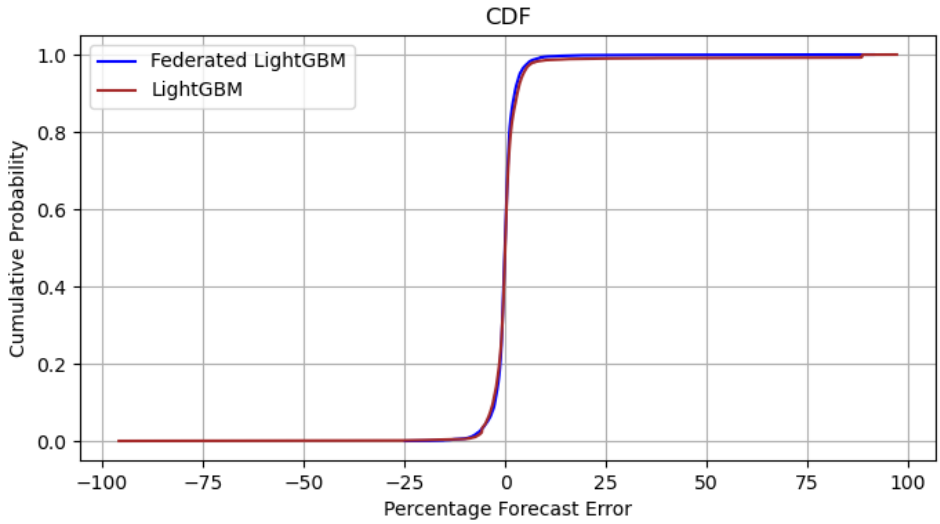

Figure 5.3.2: CDF plot for LightGBM, and Federated LightGBM

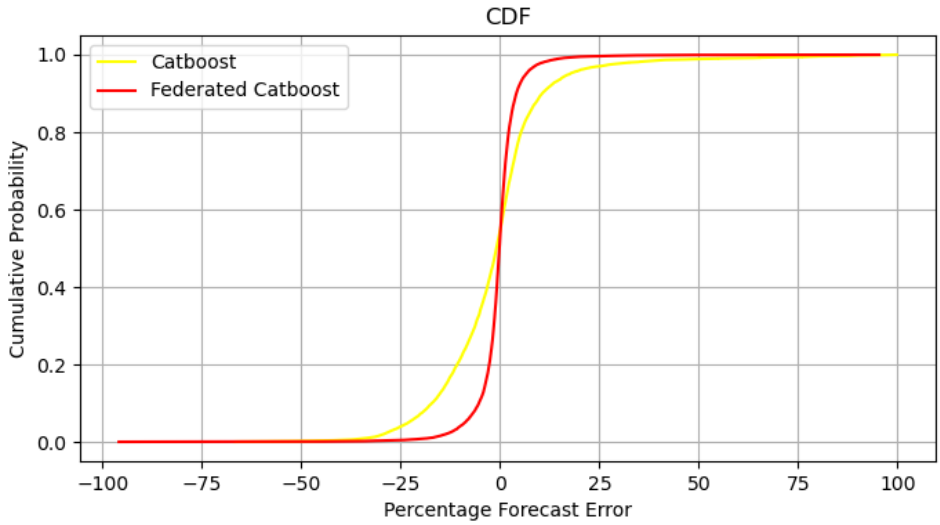

Figure 5.3.3: CDF plot for Catboost, and Federated Catboost

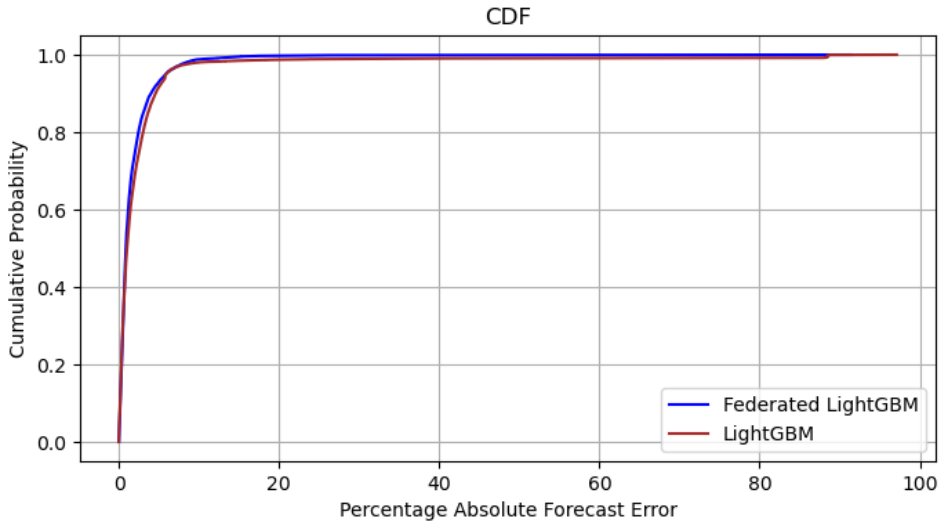

Figure 5.3.4: CDF plot with absolute values for LightGBM, and Federated LightGBM

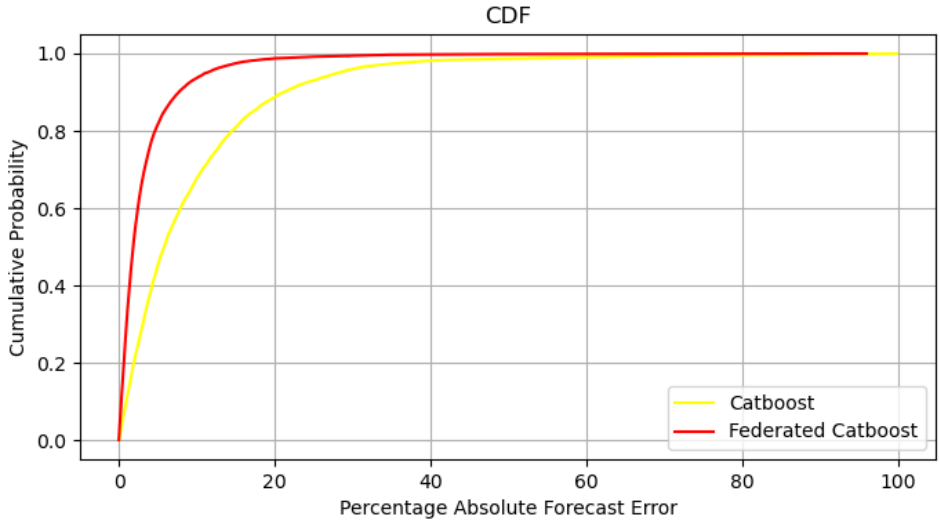

Figure 5.3.5: CDF plot with absolute values for Catboost, and Federated Catboost

CDF for different prosumers sequences

In 4.1.1 from chapter 4, the distribution of seven prosumers to four jetson nano clients is shown that is used throughout the research. This sequence is compared to three other sequences for the model performance for Federated Catboost shown in tables 5.3.2, 5.3.3, 5.3.4, and 5.3.5. For overall overview of model performance with specific sequences, CDF of percentage forecast errors was used.

Jetson Nanos	Data
Client 1	Prosumer 1 and Prosumer 2
Client 2	Prosumer 3 and Prosumer 4
Client 3	Prosumer 5 and Prosumer 6
Client 4	Prosumer 7

Table 5.3.2: Sequence 1

Jetson Nanos	Data
Client 1	Prosumer 1 and Prosumer 4
Client 2	Prosumer 3 and Prosumer 2
Client 3	Prosumer 5 and Prosumer 7
Client 4	Prosumer 6

Table 5.3.3: Sequence 2

Based on findings from Figures 5.3.6 and 5.3.7, sequence 1 performed alot better and smoother than other three sequences, hence it was chosen as the optimal combination of dividing prosumers to clients.

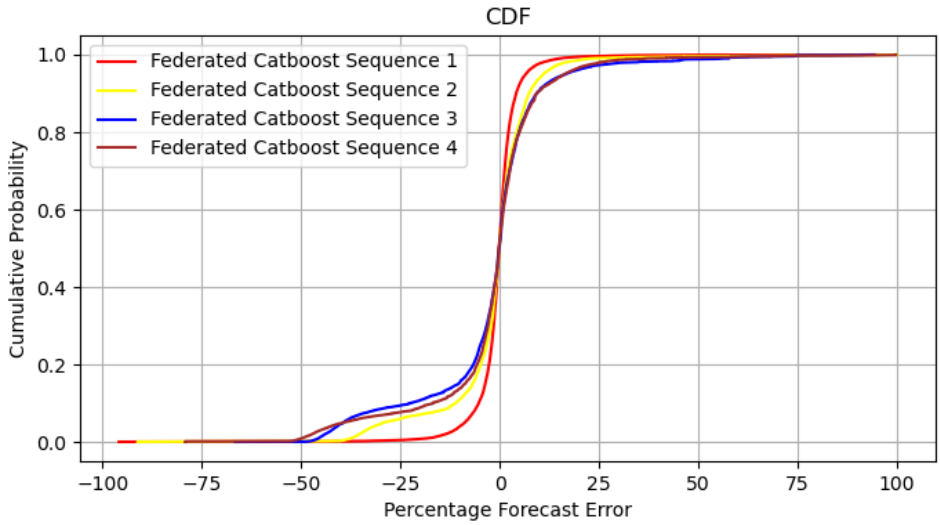

Figure 5.3.6: CDF plot for Federated Catboost for all four sequences

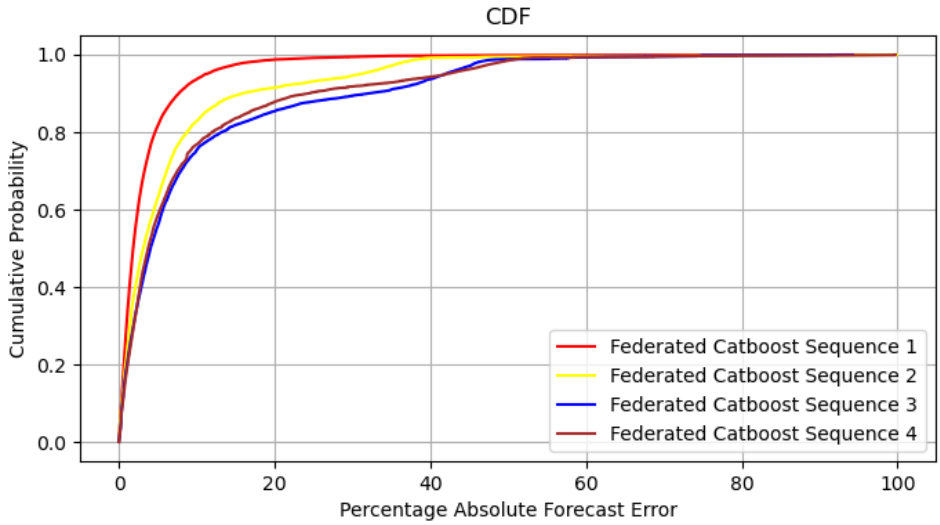

Figure 5.3.7: CDF plot with absolute values for Federated Catboost for all four sequences

Jetson Nanos	Data
Client 1	Prosumer 1 and Prosumer 6
Client 2	Prosumer 3 and Prosumer 7
Client 3	Prosumer 5 and Prosumer 2
Client 4	Prosumer 4

Table 5.3.4: Sequence 3

Jetson Nanos	Data
Client 1	Prosumer 1 and Prosumer 7
Client 2	Prosumer 3 and Prosumer 6
Client 3	Prosumer 5 and Prosumer 4
Client 4	Prosumer 2

Table 5.3.5: Sequence 4

5.3.2 Regression Metrics

Source	MAE	MQL	MPIR
Client 1	21.76	6.50	65.21
Client 2	17.82	5.99	63.29
Client 3	37.76	10.96	121.27
Client 4	24.55	7.22	72.27
All combined	12.64	4.66	63.56

Table 5.3.6: Regression metrics for the LightGBM power production for all clients and combined with federated learning

Source	MAE	MQL	MPIR
Client 1	61.85	18.74	190.94
Client 2	14.30	6.92	92.63
Client 3	24.35	7.96	95.79
Client 4	27.03	9.98	130.83
All combined	15.38	6.27	84.03

Table 5.3.7: Regression metrics for the LightGBM power consumption for all clients and combined with federated learning

Tables 5.3.6, 5.3.7, 5.3.8, and 5.3.9 present the evaluation results for all four experiments: LightGBM Production, LightGBM Consumption, Catboost Production, and Catboost Consumption. The tables present the Mean Absolute Error (MAE), the Mean Prediction Interval Range (MPIR), and the Mean Quantile Loss (MQL) for models that were trained on client 4 that had training size of 9056 rows for production experiments and 15448 rows

Source	MAE	MQL	MPIR
Client 1	98.91	25.54	210.21
Client 2	34.62	10.53	92.24
Client 3	79.09	24.69	288.08
Client 4	65.24	19.02	184.21
All combined	20.92	6.55	61.20

Table 5.3.8: Regression metrics for the Catboost power production for all clients and combined with federated learning

Source	MAE	MQL	MPIR
Client 1	47.56	17.61	202.79
Client 2	35.48	11.62	120.09
Client 3	38.24	13.35	171.09
Client 4	44.63	15.02	177.02
All combined	15.73	4.70	47.80

Table 5.3.9: Regression metrics for the Catboost power consumption for all clients and combined with federated learning

for consumption, and for other three clients with training size of 18112 rows for production experiments and 30896 rows for consumption, as well as their aggregation to the global model with training size of 61032 with all models combined for production experiments and 103394 for consumption experiments. Client 4 had testing size of 2264 rows for production experiments and 3863 rows for consumption, and for other three clients with testing size of 4528 rows for production experiments and 7726 rows for consumption, as well as their aggregation to the global model with testing size of 15261 with all models combined for production experiments and 25851 for consumption experiments.

The evaluation metrics offer extremely helpful insights into how well the forecasting models performed. Following the aggregation of models, a statistically significant decrease in each of the three metrics was observed across all four experiments. This decrease demonstrates an improvement in both the accuracy of the models' predictions and the accuracy of their uncertainty estimations.

This evaluation also highlights the impact of training size on performance as this significant decrease in MAE across all four experiments can be seen when the training is increased in combined global models

The LightGBM Production experiment produced the best results overall with an MAE of 12.64 W and a MQL of 4.66 W. These figures were determined by comparing the MAE to the MQL. These metrics demonstrate that the predictions for PV power production using LightGBM were the most accurate in relation to the actual values.

Model	MAE	R^2	MQL	MPIR	CFE
GP	49.4	0.99	29.80	87.14	0.10
LQR	1226.44	-0.37	387.40	3209.00	0.04
MQF	30.06	0.99	10.60	119.46	0.23
GBQR - CatBoost	172.90	0.91	73.69	1312.81	0.03
GBQR - LightGBM	16.12	0.99	5.05	58.27	0.10

Table 5.3.10: Regression metrics for the power production [3]

Model	MAE	R^2	MQL	MPIR	CFE
GP	77.28	0.99	4.80	143.00	0.10
LQR	556.46	-0.19	178.98	1657.99	0.25
MQF	20.59	0.97	8.05	109.34	0.11
GBQR - CatBoost	116.96	0.64	48.21	708.75	0.01
GBQR - LightGBM	16.34	0.96	6.80	91.17	0.11

Table 5.3.11: Regression metrics for the power consumption [3]

The Catboost Consumption experiment resulted in an MPIR of 47.80 watts, which was the lowest possible value. The MPIR metric is used to determine the average width of the prediction intervals; a lower value indicates that the intervals are more precise and have been narrowed down. The model is able to provide reliable and accurate prediction intervals, as evidenced by the low MPIR for Catboost Consumption.

The results show that the federated learning approach is effective in improving the overall performance of PV power forecasting models. This conclusion can be drawn from the overall picture. The decreases in MAE, MPIR, and MQL that were seen across all of the experiments are evidence that the process of collaborative model training and aggregation led to an improvement in the accuracy and uncertainty estimation.

In tables 5.3.10 and 5.3.11 the authors of [3] have created the regression metrics of production and consumption predictions from five different models. Using this as a basis of our research, we concluded that Gradient Boosting Quantile Regressions (GBQR) models like LightGBM and CatBoost provided least Mean Absolute Error (MAE). For the training and evaluation, same training set and

Table 5.3.12 provides conclusive evidence that the federated learning approach achieves same or better performance in all four experiments compared to the conventional use of LightGBM and Catboost. Prosumer was used for this anaysis to keep similar environment to [3] that had training size of 9056 rows for production experiments and 15448 rows for consumption. It had testing size of 2264 rows for production experiments and 3863 rows for consumption. This evidence is presented in the form of a table. The Mean Absolute Error (MAE) values show significant reductions, indicating improved prediction accuracy

Model	MAE	MQL	MPIR
LightGBM Production	16.12	5.05	58.27
Federated LightGBM Production	16.45	5.15	61.05
Catboost Production	172.90	73.69	1312.81
Federated Catboost Production	20.57	6.34	70.43
LightGBM Consumption	16.34	6.80	91.17
Federated LightGBM Consumption	13.90	6.14	99.58
Catboost Consumption	116.96	48.21	708.75
Federated Catboost Consumption	21.60	6.37	71.65

Table 5.3.12: Regression metrics for the power production and consumption for LightGBM and Catboost with and without federated learning for Prosumer 1 testing dataset

for PV power production and consumption. The MAE decreased from 172.90 W to 20.57 W during the course of the Catboost Production experiment; however, it increased from 16.12 W to 16.45 W during the course of the LightGBM Production experiment. In a manner parallel to this, the MAE decreased from 116.96 W to 21.60 W during the course of the Catboost Consumption experiment, and it went from 16.34 W to 13.90 W during the course of the LightGBM Consumption experiment.

In addition, the Mean Quantile Loss (MQL) and Mean Prediction Interval Range (MPIR) metrics both demonstrated significant improvements following the implementation of federated learning in all four experiments. As a result of the reduced MQL values, the precision of the quantile predictions has increased, which in turn makes the forecasts more reliable for a variety of quantiles. The lower MPIR values imply narrower and more accurate prediction intervals, which in turn provides a better estimation of the prediction uncertainty.

Overall, the findings presented in table 5.3.12 offer compelling evidence that the federated learning approach is effective in improving the accuracy and uncertainty estimation of PV power forecasting models. These findings can be found in the context of the table. It was demonstrated that the collaborative model training and aggregation process has the potential to revolutionize distributed energy management and to promote the adoption of renewable energy sources. The application of federated learning in these experiments demonstrates its promise as a valuable tool for improving decision-making in energy consumption and grid operations, as well as demonstrating its potential for more sustainable energy management.

This evaluation conclusively answered the second research question that it is infact possible to apply federated learning to tree-based model like LightGBM and Catboost, and the first research question that overall it does increase the performance of the these models after federated learning is implemented for multi-variate time-series forecasting for PV energy systems.

5.4 Summary

The findings of the research on federated learning and federated inference, taken as a whole, demonstrated the enormous potential of these cutting-edge technologies to revolutionize energy management systems. The system offered a comprehensive answer to the problem of predicting photovoltaic (PV) production and consumption in an effective and safe manner by bringing together the strengths of distributed computing, the protection of data privacy, and real-time predictive capabilities, and hence answering the research questions that federated learning can be implemented on tree-based models, and it does improve performance of LightGBM and Catboost.

Nevertheless, it is essential to recognize that federated inference is not without its share of difficulties. It can be difficult to ensure that predictions made by edge devices are consistent with one another, particularly when working with disparate datasets and a network environment that is constantly changing. In order to keep the level of accuracy and reliability of predictions consistent across the entirety of the federated system, continuous monitoring and optimization are required.

In spite of these challenges, the successful implementation of federated inference in the energy management system demonstrated its transformative impact on the manner in which predictive tasks can be efficiently performed at the edge while still respecting data privacy and security. In addition to energy management, the combination of federated learning and federated inference opened up new possibilities for a wide range of applications, such as healthcare, finance, and industrial IoT.

It is becoming increasingly clear that federated learning and federated inference, which are still in the process of developing, are the factors that will be necessary to unlock the full potential of edge devices and distributed intelligence. These technologies represent a paradigm shift in machine learning because they bring together the power of collaborative learning, the preservation of privacy, and the development of localized intelligence to produce intelligent and secure systems that are beneficial to users as well as society as a whole.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

Based on the findings that are presented in previous chapter, it has been demonstrated beyond a reasonable doubt that the federated learning approach achieves significantly better results than the conventional use of LightGBM and Catboost models, and answers the research question mention in introduction chapter. After federated learning was implemented, there was a discernible and significant drop in the values of Mean Absolute Error (MAE) for photovoltaic power production and consumption.

The MAE for LightGBM Production only increased from 16.12 W to 16.45 W, which demonstrates almost same accuracy. However, the MAE for Catboost Production decreased drastically from 172.90 W to 20.57 W, indicating improvement in the capabilities of forecasting.

The mean absolute error (MAE) in the power consumption prediction for LightGBM decreased from 16.34 W to 13.90 W, indicating an improvement in the accuracy of the prediction. In addition, the mean absolute error (MAE) decreased from 116.96 W to 21.60 W in the case of Catboost Consumption, demonstrating a significant improvement in prediction precision.

Additionally, the Mean Quantile Loss (MQL) and Mean Prediction Interval Range (MPIR) metrics both demonstrated significant reductions after federated learning was implemented across all four experiments. These metrics are essential in order to assess the dependability and robustness of the prediction intervals, and the reduction that was observed indicates that prediction confidence has increased.

During the research, federated learning was utilized during the phase in which the model was being trained, and federated inference was utilized during the phase in which the prediction was being made. The end result was a comprehensive strategy for forecasting

PV power production and consumption that protected users' privacy. This approach was successful in addressing the challenges of distributed and secure machine learning, in particular with regard to the management of sustainable energy.

The implementation of this strategy was made much easier by the incorporation of the FLWR framework, devices powered by NVIDIA Jetson Nano, and federated inference. FLWR was a critical component in achieving the goals of enabling secure and collaborative model training across a distributed client base. Edge computing capabilities were provided by the NVIDIA Jetson Nano devices. These capabilities ensured that trained models could be deployed and used directly on edge devices, thereby reducing the amount of data transferred, maximizing the accuracy of real-time predictions, and maintaining the confidentiality of user data.

The findings of the study highlighted the enormous potential of federated learning to increase the applicability of edge devices across a wider range of domains while also protecting data privacy and maximizing computational efficiency. This method not only solves the problems that are associated with distributed machine learning, but it also paves the way for the deployment of machine learning models on devices at the edge of the network that have limited resources.

In conclusion, the findings of the research highlighted the efficacy of federated learning in improving the accuracy of PV power forecasting while maintaining data privacy. The fact that the federated learning approach achieved lower values for the MAE, MQL, and MPIR metrics in each and every one of the four experiments provides undeniable evidence of its superiority to more traditional methods. The FLWR framework and the edge computing capabilities of NVIDIA Jetson Nano devices were utilized by this research in order to provide a solution that was both comprehensive and scalable for the forecasting of sustainable energy. In addition, the successful implementation of federated inference on edge devices opens up new possibilities for efficient machine learning deployment in the context of IoT and edge computing. These new possibilities are designed to protect users' privacy while maximizing performance. This research's findings contribute valuable insights towards the development of safe, privacy-preserving, and accurate forecasting solutions for the energy domain as the world continues to embrace the potential of federated learning and edge computing.

6.2 Future Work

The expansion of PV power forecasting methodologies beyond the LightGBM and Catboost models is going to be a central focus of work that will be done in the field of future research. The process of forecasting the power generated by photovoltaic cells is both difficult and important. Investigating different machine learning models offers a significant opportunity to improve both the precision and adaptability of the forecasting

procedure.

Concurrently, work is being done to integrate the forecasting solution with the FLWR framework. Progress has been made in this direction. The emergence of federated learning as a powerful paradigm for training machine learning models on decentralized data while maintaining data privacy and security has occurred recently. There is a plan to leverage FLWR, which offers a stable and scalable platform for federated learning implementations, in order to facilitate collaborative model training across multiple smart energy networks.

Through the implementation of FLWR, our federated learning approach will be able to take advantage of increased communication efficiency as well as reduced computation costs. It makes it possible for customers to take part in the training process without having to disclose any of their raw data, thereby protecting their privacy and allowing them to retain ownership of their data. The updates to the model that are sent in by the various clients are compiled by the central server, which encourages the sharing of knowledge and makes it possible to develop a global model that is more generalized and robust. In addition, FLWR supports fault tolerance, which enables the system to handle potential client failures in a graceful manner and ensures that the federated learning process will continue without interruption.

One of the most important advantages of FLWR is its capacity to manage the heterogeneity of distributed clients. These clients may have varying amounts of data and different computational resources, so FLWR must be able to accommodate these differences. Because of the federated model's adaptability, which ensures that it can accommodate the diversity of data distributions across various energy networks, this leads to more accurate and representative forecasts. In addition to this, the integration of FLWR encourages collaboration among stakeholders in the smart energy domain, which in turn promotes the exchange of knowledge and drives innovation in the field of sustainable energy management.

Comprehensive analyses will be carried out across a wide variety of smart energy networks that are located in a wide variety of geographic locations in order to evaluate the impact that weather conditions have on the PV power forecasting. The amount of sunlight that is available and other environmental factors have a direct influence on the amount of power that can be generated using PV, so weather conditions play an extremely important part in this process. We can gain valuable insights into the model's strengths and limitations if we study how well it performs under a variety of weather conditions and put it through its paces.

For instance, the model's ability to accurately predict PV power generation might be challenged on days when there is a lot of cloud cover or when there is a period of low sunlight. Through the analysis of such cases, we are able to identify potential areas for

model improvement. These areas may include the incorporation of additional weather data or the incorporation of external weather forecasts into the process of forecasting. With the help of this analysis, we will be able to develop weather-aware forecasting models that are able to adjust to shifting weather patterns and provide accurate predictions under a wide variety of circumstances.

In addition, integrating the forecasting method into an MLOps environment appears to be an essential component of the work that will be done in the future. MLOps is an application of the DevOps philosophy that has been extended to the realm of machine learning. It offers a method that is both structured and automated for managing the entire machine learning lifecycle. It entails a number of stages, such as the preparation of data, the training of models, the deployment of models, and the ongoing monitoring and improvement of models.

Forecasting system will be developed that is both more adaptable and reliable if we integrate the PV power forecasting solution into an MLOps environment. This environment will continually monitor the performance of the model and assess the accuracy of the model in comparison to real-time data. In the context of PV power forecasting, MLOps can evaluate the quality of the model and identify potential drift or degradation in performance by taking into account weather conditions, recent patterns of power consumption, and historical data.

It is possible for the MLOps system to automatically trigger model retraining or fine-tuning in the event that the accuracy of the model decreases as a result of changes in weather patterns or other factors. This helps to ensure that the forecasting model continues to be accurate and up-to-date. In addition, MLOps can facilitate the deployment of updated models to customers as well as the central server in a seamless manner, which encourages a cycle of continuous improvement for PV power forecasting.

One of the most important goals for research in the future will be to find a solution to the problem of handling data that contains imbalances or outliers. Variable patterns of PV power production are common in smart energy networks. Depending on the weather conditions, certain weather conditions can result in extremely high power values or temporary disruptions in power production. These anomalies have the potential to significantly affect both the training process for the model and the accuracy of its predictions.

In order to meet the demands of this obstacle, our plan is to devise specialized methods for the detection and management of imbalanced data as well as outliers. The model will be able to effectively adapt to different patterns of power generation using these techniques, and it will be able to make accurate predictions even when there are outliers in the data. Methods such as data augmentation, weighted loss functions, and outlier detection algorithms will be investigated and adapted to accommodate the specific requirements of

PV power forecasting.

The goals of these ongoing research projects for the foreseeable future are to investigate a wide variety of machine learning models, to incorporate federated learning by means of FLWR, to investigate the impact of various weather conditions, to implement MLOps for real-time adaptability, and to deal with imbalanced data and outliers. These efforts are directed toward advancing the field of photovoltaic (PV) power forecasting in the hopes of providing reliable and accurate predictions that will support sustainable energy management and the seamless integration of renewable energy sources into the power grid. We can further improve the reliability and effectiveness of PV power forecasting and contribute to the advancement of sustainable energy solutions for a greener and more sustainable future if we investigate these potential avenues.

Bibliography

[1] Aldous, David. “Tree-based models for random distribution of mass”. In: *Journal of Statistical Physics* 73 (1993), pp. 625–641. DOI: 10.1007/BF01054343.

[2] AlHakeem, Dana, Mandal, Pranab, Haque, Ahsan U, Yona, Abed, Senjyu, Tomonobu, and Tseng, Tung-Liang. “A new strategy to quantify uncertainties of wavelet-grnn-pso based solar PV power forecasts using bootstrap confidence intervals”. In: *2015 IEEE Power Energy Society General Meeting*. IEEE. 2015, pp. 1–5.

[3] Aupke, Paul, Seema, Kassler, Andreas, and Theocharis, Apostolos. “Power Production and Consumption Estimation with Uncertainty bounds in Smart Energy Grids”. In: (2021).

[4] Baldán, Francisco Javier and Benítez, Jose María. “Complexity measures and features for times series classification”. In: *arXiv preprint arXiv:2002.12036* (2020).

[5] Baldán, Francisco Javier and Benítez, Jose María. “Distributed FastShapelet Transform: a Big Data time series classification algorithm”. In: *Information Sciences* 496 (2019), pp. 451–463.

[6] Baldán, Francisco Javier and Benítez, Jose María. “Multivariate times series classification through an interpretable representation”. In: *Information Sciences* 569 (2021), pp. 596–614.

[7] Baldán, Francisco Javier, Peralta, Diego, Saeys, Yvan, and Benítez, Jose María. “SCMFTS: scalable and distributed complexity measures and features for univariate and multivariate time series in Big Data environments”. In: *International Journal of Computational Intelligence Systems* 14.1 (2021).

[8] Bianchi, Filippo Maria, Maiorino, Enrico, Kampffmeyer, Michael Christian, Rizzi, Antonello, and Jenssen, Robert. “An overview and comparative analysis of recurrent neural networks for short term load forecasting”. In: *arXiv preprint arXiv:1705.04378* (2017).

[9] Borovykh, Anastasia, Bohte, Sander, and Oosterlee, Kees W. “Conditional time series forecasting with convolutional neural networks”. In: *arXiv preprint arXiv:1703.04691* (2017).

[10] Cinar, Yigit G, Mirisaee, Hamed, Goswami, Pulkit, Gaussier, Eric, At-Bachir, Abdelaziz, and Strijov, Vladimir. “Position-based content attention for time series forecasting with sequence-to-sequence rnns”. In: *International Conference on Neural Information Processing*. Springer, Cham. 2017, pp. 533–544.

[11] Dasari, Srinivas Varma, Mittal, Kapil, Sasirekha, G, Bapat, Janhavi, and Das, Debashis. “Privacy enhanced energy prediction in smart building using federated learning”. In: *2021 IEEE International IoT, Electronics and Mechatronics Conference (IEMTRONICS)*. IEEE. 2021, pp. 1–6.

[12] *Federated Learning Guide*. URL: `https://www.v7labs.com/blog/federated-learning-guide`.

[13] *Federated Learning: A Comprehensive Guide*. URL: `https://www.altexsoft.com/blog/federated-learning/`.

[14] Fekri, Marzieh N, Grolinger, Katarina, and Mir, Shayan. “Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks”. In: *International Journal of Electrical Power & Energy Systems* 137 (2022), p. 107669.

[15] *Flower Website*. `https://flower.dev/`.

[16] Foreman-Mackey, Daniel, Agol, Eric, Ambikasaran, Sivaram, and Angus, Ruth. “Fast and scalable Gaussian process modeling with applications to astronomical time series”. In: *Astronomical Journal* 154.6 (2017), p. 220.

[17] Fulcher, Ben D. “Feature-based time-series analysis”. In: *arXiv preprint arXiv:1709.08055* (2017).

[18] Fulcher, Ben D, Little, Max A, and Jones, Nick S. “Highly comparative time-series analysis: the empirical structure of time series and their methods”. In: *Journal of the Royal Society Interface* 10.83 (2013), p. 20130048.

[19] Galakatos, Alexandros, Crotty, Aaron, and Kraska, Tim. “Distributed Machine Learning”. In: (2018).

[20] J.Kout J.Kléma, M.Vejmelka. “Predictive system for multivariate time series”. In: *Cybernetics and Systems* (2004), pp. 723–728.

[21] Kairouz, Peter, McMahan, H Brendan, Avent, Benjamin, Bellet, Aurélien, Bennis, Mehdi, Bhagoji, Arjun Nitin, Bonawitz, Keith, Charles, Zachary, Cormode, Graham, Cummings, Rachel, et al. “Advances and open problems in federated learning”. In: *Foundations and Trends® in Machine Learning* 14.1-2 (2021), pp. 1–210.

[22] Kang, Yanfei, Hyndman, Rob J, and Li, Feng. *Efficient generation of time series with diverse and controllable characteristics*. Tech. rep. Monash University, Department of Econometrics and Business Statistics, 2018.

[23] Ke, Guolin, Meng, Qi, Finley, Thomas, Wang, Taifeng, Chen, Wei, Ma, Weidong, Ye, Qiwei, and Liu, Tie-Yan. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[24] Kumar, R, Khan, AA, Kumar, J, Zakria, NAG, Golilarz, NA, Zhang, S, Ting, Y, Zheng, C, and Wang, W. "Blockchain-federated learning and deep learning models for covid-19 detection using CT imaging". In: *IEEE Sensors Journal* 21.14 (2021), pp. 16301–16314.

[25] Laptev, Nikolay, Amizadeh, Saeed, and Flint, Ian. "Generic and scalable framework for automated time-series anomaly detection". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1939–1947.

[26] Li, Yao, Li, Jun, and Wang, Yi. "Privacy-preserving spatiotemporal scenario generation of renewable energies: A federated deep generative learning approach". In: *IEEE Transactions on Industrial Informatics* 18.4 (2021), pp. 2310–2320.

[27] *LightGBM Documentation: Parallel Learning Guide*. https://lightgbm.readthedocs.io/en/latest/Parallel-Learning-Guide.html. Accessed: Insert Date.

[28] Lin, Jianjian, Ma, Jian, and Zhu, Jun. "A privacy-preserving federated learning method for probabilistic community-level behind-the-meter solar generation disaggregation". In: *IEEE Transactions on Smart Grid* 13.1 (2021), pp. 268–279.

[29] Lines, Jason, Davis, Larry M, Hills, Jon, and Bagnall, Anthony. "A shapelet transform for time series classification". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2012, pp. 289–297.

[30] Liu, Yang, Liu, Ying, Liu, Zehong, Liang, Ying, Meng, Chenglin, Zhang, Jie, and Zheng, Yu. "Federated forest". In: *IEEE Transactions on Big Data* (2020). URL: https://arxiv.org/abs/1905.10053.

[31] Liu, Yu, Yu, Jia Jun Quentin, Kang, Jie, Niyato, Dusit, and Zhang, Shengli. "Privacy-preserving traffic flow prediction: A federated learning approach". In: *IEEE Internet of Things Journal* 7 (2020), pp. 7751–7763.

[32] Lngkvist, Magnus, Karlsson, Lars, and Loutfi, Amy. “A review of unsupervised feature learning and deep learning for time-series modeling”. In: *Pattern Recognition Letters* 42 (2014), pp. 11–24.

[33] Lubba, Carl Henrik, Sethi, Siddharth S, Knaute, Pascal, Schultz, Simon R, Fulcher, Ben D, and Jones, Nick S. “catch22: CAnonical Time-series Characteristics”. In: *Data Mining and Knowledge Discovery* 33.6 (2019), pp. 1821–1852.

[34] Lucas, Ben, Shifaz, A, Pelletier, C, O’Neill, L, Zaidi, Nayyar, Goethals, Bart, Petitjean, Francois, and Webb, Geoffrey I. “Proximity forest: an effective and scalable distance-based classifier for time series”. In: *Data Mining and Knowledge Discovery* 33.3 (2019), pp. 607–635.

[35] McMahan, H Brendan, Moore, Eider, Ramage, Daniel, Hampson, Seth, and Arcas, Blaise Aguera y. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[36] Mohammad Khalil Mourad Esseghir, Lakhdar Merghem-Boulahia. “Federated learning for energy-efficient thermal comfort control service in smart buildings”. In: *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2021, pp. 1–6.

[37] Nastaran Gholizadeh, Petr Musilek. “Federated learning with hyperparameter-based clustering for electrical load forecasting”. In: *Internet of Things* 17 (2022), p. 100470. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2021.100470. URL: https://www.sciencedirect.com/science/article/pii/S2542660521001104.

[38] Ng, Nicholas, Gabriel, Rodney A, McAuley, Julian, Elkan, Charles, and Lipton, Zachary C. “Predicting surgery duration with neural heteroscedastic regression”. In: *arXiv preprint arXiv:1702.05386* (2017).

[39] Peralta, Diego and Saeys, Yvan. “Robust unsupervised dimensionality reduction based on feature clustering for single-cell imaging data”. In: *Applied Soft Computing* 93 (2020), p. 106421.

[40] Prokhorenkova, Liudmila, Gusev, Gleb, Vorobev, Aleksandr, Dorogush, Anna Veronika, and Gulin, Andrey. “CatBoost: unbiased boosting with categorical features”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

[41] Prusty, B. R. and Tripathy, D. S. “Comparison of photovoltaic generation uncertainty models for power system planning using regression framework”. In: *2021 IEEE International Power and Renewable Energy Conference (IPRECON)*. IEEE. 2021, pp. 1–5.

[42] Rakthanmanon, Thanawin and Keogh, Eamonn. “Fast shapelets: a scalable algorithm for discovering time series shapelets”. In: *Proceedings of the 2013 SIAM International Conference on Data Mining*. 2013, pp. 668–676.

[43] Sun, Mengjie, Zhang, Tengfei, Wang, Yisen, Strbac, Goran, and Kang, Chongqing. “Using Bayesian deep learning to capture uncertainty for residential net load forecasting”. In: *IEEE Transactions on Power Systems* 35.1 (2020), pp. 188–201.

[44] Taieb, Souhaib Ben and Atiya, Amir F. “A bias and variance analysis for multistep-ahead time series forecasting”. In: *IEEE transactions on neural networks and learning systems* 27.1 (2016), pp. 62–76.

[45] Taylor, James W. “A quantile regression neural network approach to estimating the conditional density of multi-period returns”. In: *Journal of Forecasting* 19.4 (2000), pp. 299–311.

[46] Team, Data Science. *What is light GBM? - machine learning*. Nov. 2020. URL: https://datascience.eu/machine-learning/1-what-is-light-gbm/.

[47] *Time series models*. URL: https://www.pinterest.com/pin/569142471630774436/.

[48] Valentin Flunkert David Salinas, Jan Gasthaus. “DeepAR: Probabilistic forecasting with autoregressive recurrent networks”. In: *arXiv preprint arXiv:1704.04110* (2017).

[49] Venables, W. N. and Ripley, B. D. “Tree-based Methods”. In: *Modern Applied Statistics with S-PLUS*. New York, NY: Springer New York, 1999, pp. 303–327.

[50] Verbraeken, Jeroen, Wolting, Marc, Katzy, Julien, Kloppenburg, Jorn, Verbelen, Timothy, and Rellermeyer, Jan S. “A survey on distributed machine learning”. In: *ACM Computing Surveys (CSUR)* 53.2 (2020), pp. 1–33.

[51] Wang, Yi, Bennani, Ismail Lahlou, Liu, Xiaozhe, Sun, Mengjie, Zhou, Yuan, et al. “Electricity consumer characteristics identification: A federated learning approach”. In: *IEEE Transactions on Smart Grid* 12.4 (2021), pp. 3637–3647.

[52] Wen, Haoran, Du, Yang, Lim, Eng, Wen, Huiqing, Yan, Ke, Li, Xingshuo, and Jiang, Lin. “A solar forecasting framework based on federated learning and distributed computing”. In: *Building and Environment* 225 (2022), p. 109556. DOI: 10.1016/j.buildenv.2022.109556.

[53] Wen, Ruofeng, Torkkola, Kari, Narayanaswamy, Balakrishnan, and Madeka, Dhruv. “A multi-horizon quantile recurrent forecaster”. In: *arXiv: Machine Learning* (2017).

[54] Wu, Yifan, Cai, Shichao, Xiao, Xue, Chen, Guanling, and Ooi, Beng Chin. “Privacy preserving vertical federated learning for tree-based models”. In: *arXiv preprint arXiv:2008.06170* (2020). URL: https://arxiv.org/pdf/2008.06170.pdf.

[55] Xu, Qiang, Liu, Xiaolong, Jiang, Chao, and Yu, Kai. “Quantile autoregression neural network model with applications to evaluating value at risk”. In: *Applied Soft Computing* 49 (2016), pp. 1–12.

[56] Zhang, Wei, Quan, Hao, and Srinivasan, Dipti. “An improved quantile regression neural network for probabilistic load forecasting”. In: *IEEE Transactions on Smart Grid* 10.4 (2019), pp. 4425–4434.

[57] Zhang, Xitong, Fang, Fei, and Wang, Jun. “Probabilistic solar irradiation forecasting based on variational Bayesian inference with secure federated learning”. In: *IEEE Transactions on Industrial Informatics* 17.11 (2020), pp. 7849–7859.

Appendix - Contents

.1 NVIDIA Jetson Nano

The NVIDIA Jetson Nano is an artificial intelligence (AI) and machine learning (ML) application-specific edge computing device that is both powerful and compact. It is a member of the Jetson family of embedded computing platforms that NVIDIA has developed. These platforms have been purpose-built to execute deep learning models and to accelerate AI-related tasks at the edge.

The Jetson Nano features a central processing unit (CPU) with four ARM Cortex-A57 cores and a graphics processing unit (GPU) based on NVIDIA's Maxwell architecture that has 128 CUDA cores. The combination of the central processing unit (CPU) and the graphics processing unit (GPU) provides exceptional computing power, which enables the device to handle computationally intensive tasks such as object detection, image recognition, natural language processing, and many more.

The Jetson Nano's artificial intelligence performance is one of its most notable characteristics. The Jetson Nano is able to accelerate artificial intelligence workloads thanks to NVIDIA's CUDA architecture and optimized software libraries. This paves the way for real-time and low-latency inferencing on the device. This is especially helpful for applications that require quick and accurate responses, such as industrial automation, intelligent surveillance systems, and autonomous robots.

As a result of the device's support for well-known AI frameworks such as TensorFlow, PyTorch, and MXNet, software developers are able to make use of their already-created AI models and easily deploy them on the Jetson Nano. Because it already has the necessary software development kits and libraries installed, getting started with it is easy and convenient, regardless of whether you are an experienced AI developer or just getting started in the field.

In addition to its capabilities in artificial intelligence, the Jetson Nano is also well-suited for use in applications that require general-purpose computing. It has a variety of input and output ports, such as HDMI, USB, Ethernet, and GPIO, which gives it the flexibility to be used for a wide variety of applications in addition to AI.

Because of its small size and low power consumption, the Jetson Nano is ideally suited for edge computing scenarios, which are those in which resources are limited and power efficiency is of the utmost importance. Due to the fact that it is so compact, it can be easily incorporated into a wide variety of devices, including drones, smart cameras, Internet of Things devices, and other edge computing solutions.

The NVIDIA Jetson Nano is an edge computing platform that is powerful and efficient. It brings AI capabilities to the edge, which enables developers to build innovative and intelligent applications for a wide variety of industries and use cases. Because of its high computational power, excellent performance in AI tasks, and adaptability, it has quickly

Figure .1.1: Jetson Nano Developer Kit

become a popular choice for AI development and deployment at the edge.

.2 SSH configuration

SSH, which is an abbreviation that stands for ”Secure Shell,” is a network protocol that uses cryptography to ensure the confidentiality of communication between two networked devices. It offers a safe and encrypted method for accessing and managing remote devices over an unsecured network like the internet. SSH is a secure shell that can be logged into remotely, commands can be executed remotely, and files can be transferred between computers.

The communication between the client (your local machine) and the server (the remote device) is encrypted when you establish an SSH connection to a remote device. This ensures that sensitive data, such as login credentials and commands, are protected from potential eavesdropping and tampering.

.2.1 Using SSH to Establish a Connection Between Jetson Nanos and Laptop

The following steps need to be taken in order to connect Jetson Nanos to your laptop using SSH. To begin, you will need to activate SSH on the Jetson Nano device you are using. In order to accomplish this, you will need a monitor and keyboard to locally log in to the Jetson Nano. After that, launch a terminal and input the following command to turn on secure shell:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

This will start the SSH server that is installed on the Jetson Nano and enable it.

The next step is to locate the IP address that is assigned to your Jetson Nano. On the Jetson Nano, you will need to execute the following command in order to accomplish this:

```
hostname -I
```

When you do this, the IP address of the Jetson Nano as it appears on the local network will be displayed.

You can install an SSH client on your laptop if it does not already have one installed. If your laptop does not already have an SSH client installed, you will need to install one. Tools such as PuTTY and OpenSSH (built-in) are available to use if you have Windows. SSH clients are typically pre-installed in macOS and Linux systems by default.

To connect to the Jetson Nano using SSH on your laptop, open the terminal or command prompt, and type in the following command:

```
ssh username>@jetson_nano_ip_address>
```

Replace 'username' with the username for your Jetson Nano (the default is 'ubuntu'), and 'jetson nano ip address' with the IP address that you obtained in step 2 of this process.

When you run the SSH command, you will be prompted to enter the password for the username associated with the Jetson Nano. Enter the password. After entering the password, you should be able to connect to the Jetson Nano using SSH with no problems.

As soon as the SSH connection has been established, you will be able to remotely execute commands on the Jetson Nano using the terminal or command prompt on your laptop. This makes it possible for you to manage and control the Jetson Nano without the necessity of having a monitor, keyboard, and mouse directly connected to the device. Because SSH enables a user to interact with remote devices in a secure and hassle-free manner, it is an indispensable instrument for managing edge devices such as the Jetson Nano.