Imperial College London

Massachusetts Institute of Technology

MEng Final Year Project Report

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Laboratory for Information and Decision Systems (LIDS)

Ensemble Machine Learning with Time Series Forecasting for Optimized Electric Vehicle Charging

---

Academic Supervisor:
Dr. Mardavij Roozbehani
(MIT)

Author:
Giorgos Iacovides

Second Marker:
Prof. Alessandro Astolfi
(Imperial College London)

June 2024

Acknowledgments

First and foremost, I would like to express my sincere gratitude to Dr. Mardavij Roozbehani for his continuous guidance and support. Our weekly meetings stimulated my academic curiosity and pushed me to strive for more. I hope the relationship we have developed continues for years to come.

I would also like to express my appreciation to Professor Alessandro Astolfi, who selected me as the sole student from the EE department of Imperial College London to represent Imperial at MIT, allowing me to experience both MIT and the USA in general. His trust in me is something I will always be grateful for.

To all my friends, thank you for standing by my side and supporting me throughout these four years at university. Together, we have created some wonderful and amazing memories that I will cherish forever.

Above all, I would like to thank the four most important people in my life: my parents, sister, and grandfather. They are my motivation and inspiration, and the reason I strive to be the best version of myself. Their unconditional support is truly unique and something I never take for granted. I hope I have made you proud.

Abstract

The large integration of Electric Vehicles (EVs) can stress the distribution network, leading to demand spikes and voltage instabilities. Smart charging scheduling is critical for mitigating these effects and relies heavily on predicting EV users' charging behavior, including stay duration and energy consumption. Therefore, in this project, we propose using cross-correlations between users and an ensemble machine learning model, that bases decisions on the entropy-sparsity ratio of EV users' historical data to optimize the EV charging schedule. This approach accounts for both indicators, as it is noted that prediction errors increase with data entropy and decrease with data sparsity. Our predictive model demonstrates a significant improvement in performance, reducing both the stay duration and energy consumption prediction errors by 25% compared to previous studies using the same dataset. The prediction results are then applied to an optimal EV charging scheduling algorithm to minimize peak load while reducing EV charging cost. A numerical simulation using real charging data and three-phase infrastructure constraints is conducted to show the effectiveness of our improved predictions and EV load management strategy. The results show that our charging scheduling, combined with our ensemble model predictions, achieves a daily average reduction of 76% in peak load and 16% in charging cost when compared to existing uncontrolled charging methods. Moreover, recognizing the importance of day-ahead scheduling for aggregate demand in large-scale charging facilities, our work extends to predicting the arrival times of EV users using time-series forecasting. Our hybrid model, which combines statistical and deep learning methods with noise filtering techniques, achieves state-of-the-art results, whilst also being more robust, interpretable, and significantly smaller in terms of the number of trainable parameters, compared to the current leading deep learning methods available.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

Key energy initiatives worldwide aim to achieve full independence from fossil fuels, focusing on the massive integration of renewable energy sources. The European Union, for instance, aspires to become climate-neutral by 2050 [1], a goal that entails significant reductions in greenhouse gas emissions across all sectors. Similarly, the United States plans to eliminate fossil fuels as a form of energy generation by 2035 [2]. In these initiatives, Electric Vehicles (EVs) play a crucial role. This is because, they not only offer the potential to increase energy efficiency in transportation and reduce greenhouse gas emissions, but also possess storage capabilities that can mitigate the inherent volatility of renewable energy sources, thus stabilizing energy flows [3].

Recognizing their critical role in achieving environmental sustainability, governments around the world have adopted aggressive targets to accelerate the transition to electric transportation. For example, Norway has vowed to end the sale of gas and diesel cars by 2025, India by 2030, and Britain and France by 2040 [4]. In addition, California has set the goal of having 5 million electric and other zero-emission vehicles on the road by 2030 [5]. As a result, the International Energy Agency projects that EVs will make up more than 60% of vehicles sold globally by 2030 [6].

However, the increasing number of EVs also means that the rise of energy demand is now becoming a challenge to the electricity grid, as it has the potential to stress the distribution grid and cause voltage instability [7]. This is because, based on the EV charging data collected on the University of California, Los Angeles (UCLA) campus, the average energy consumption is about 8 kWh per charge, which is similar to a daily household energy demand [8]. EV charging load often shows two peaks in a day, one in the morning when people plug in the EV at the workplace and the other in the evening when people get home from work. Without proper energy management for EV charging, the huge power demand due to a large number of plugged-in EVs can stress the distribution grid, degrade the power quality [9][10], and impact the wholesale electricity market [11].

On the other hand, the AAA Foundation report reveals that US drivers spend only 0.8 hours on average behind the wheel each day [12], leaving their vehicles parked for the majority of the time. This implies that EVs can have great flexibility for charging, diminishing the necessity for immediate charging upon connection. Consequently, EV charging scheduling plays an important role in distributing and allocating the charging time according to the EVs' availability for overall load management. A proper EV load management not only mitigates the adverse effects of EV charging but also enhances grid efficiency through load valley filling and peak shaving [13]. These practices distribute electricity demand more evenly throughout the day, effectively reducing peaks and filling in valleys in daily consumption patterns.

However, the stochasticity of EV user charging behaviors, including arrival times at charging stations, duration of stay, and energy requirements poses a significant challenge for the management and optimization of charging scheduling. Additionally, asking EV owners for their input directly has proven to be quite unreliable [14]. This unreliability primarily stems from the lack of incentives for users to provide true values for their expected departure time and energy demands. Moreover, 'range anxiety', a phenomenon that describes EV owners' concerns about their vehicle's battery sufficiency for a planned journey [3], further exaggerates this issue, as it leads to users overestimating their demand requirements.

As such, our research can be separated into two pivotal stages. In the first stage, we aim to learn the EV user charging behavior by implementing personalized parameter estimation algorithms, capable of predicting the users' expected departure time and required energy demand based on their arrival times. This will eliminate the reliance on users to manually input these charging parameters, making the process more robust and accurate [14]. Moreover, to mitigate the impact of noisy data from individual users, *correlations* between users will be utilized. This approach enables the amalgamation of data from both the individual and their most correlated peers into each prediction, enhancing predictions' resilience against data variability and unforeseen daily fluctuations. At the same time, as both statistical and deep learning predicting algorithms are based on historical data and the historical charging patterns may be very different for each user, there is no one-size-fits-all predicting method for

all EV owners. Acknowledging this, our research aims to classify distinct charging patterns to inform the selection of the most appropriate predictive algorithm for each user. This tailored approach will be implemented using an *ensemble* machine learning algorithm, designed to dynamically select the model that most closely aligns with the individual charging behavior of each EV owner, thereby significantly enhancing the overall prediction accuracy.

In the second stage, using the predicted charging parameters of each user, an adaptive charging algorithm will be formulated as a constrained optimization problem, aiming to optimize the EV charging, subject to power and infrastructure constraints. The objective of our optimization problem is twofold: minimizing the peak load, which is equivalent to load variance minimization [15], in order to reduce the stress on distribution networks, as well as, reducing the total charging cost to EV owners, thereby maximizing consumer welfare. Moreover, to enhance the applicability of our optimization problem to real-world scenarios, we will take into account practical considerations, such as real public charging data and unbalanced three-phase infrastructure constraints. By doing so, we aim for our simulations to more accurately capture the behavior and complexities of real-world EV charging systems.

The overall project framework graphically depicting the two stages of our project is shown in Figure 1.1.

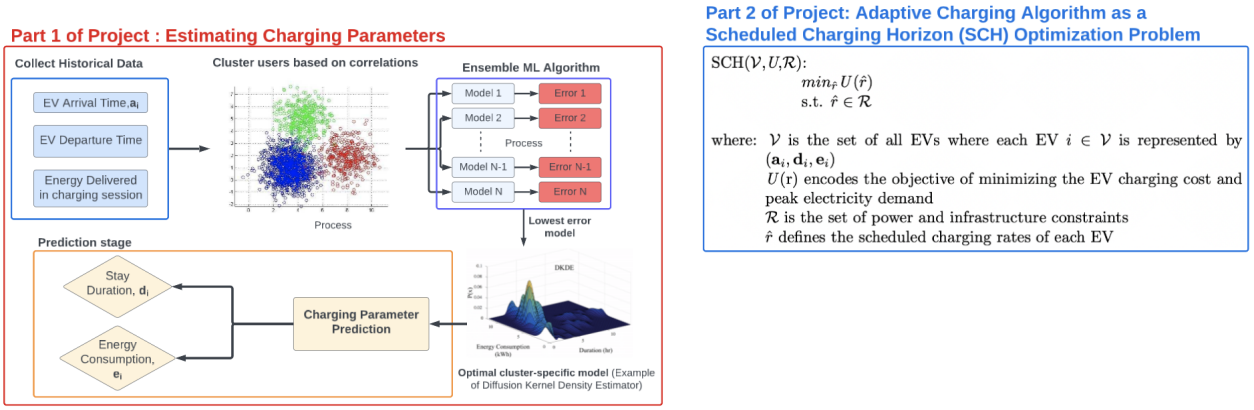


Figure 1.1: Proposed project framework for smart EV charging scheduling.

Furthermore, whilst our primary research focuses on predicting the EV user's charging behavior based on a given arrival time - suitable for individual charging stations and small-scale charging facilities - we also recognize the critical need for accurate arrival time predictions at

large-scale, high-density (LSHD) charging facilities. Such predictions are essential for day-ahead scheduling, where utilities require accurate estimates of aggregate demand to manage the high electricity loads that significantly impact the distribution grid [16]. This need is increasingly important as a growing number of EV drivers, lacking access to home charging solutions, turn to workplace and public use facilities for charging, making LSHD charging facilities vital in the transition to EVs [17]. To address this challenge, our research extends to developing sophisticated time-series forecasting models using a blend of statistical and deep learning techniques. These models aim not only to predict the EV owner's arrival times accurately but also to serve a wide array of practical applications, including weather, climate, finance, and retail forecasting [18]. This broad applicability underscores the versatility and potential impact of our work beyond the immediate context of EV charging.

1.2 Project Structure

This project is structured into several chapters:

- **Chapter 2 - Literature Review**: This chapter introduces previous studies on smart electric vehicle charging and time series forecasting. It provides a high-level overview of the various approaches employed in the past, detailing their disadvantages and limitations. This analysis identifies gaps in the literature and justifies the research direction of our project.

- **Chapter 3 - Prediction of EV User Behavior**: This chapter outlines the background theory behind the predictive models implemented and describes the methodology followed in our work, including the adoption of user correlations and our ensemble machine learning algorithm. A detailed comparison of our model's performance in estimating EV user behavior, particularly in predicting stay duration and energy consumption, against state-of-the-art (SOTA) models present in the literature, is then presented.

- **Chapter 4 - EV Charging Scheduling**: This chapter provides the theoretical back-

ground for the online charging scheduling algorithms utilized and details the creation of a realistic simulation environment that aligns with real-world scenarios. It then evaluates the results of our charging algorithm, which uses the predicted values from Chapter 3 as inputs, in achieving our objectives of minimizing peak load demand and charging costs to EV owners compared to uncontrolled charging, the current charging method.

- **Chapter 5 - Time Series Forecasting**: This chapter provides an overview of the leading statistical and deep learning time series models available in the literature. It details the methodology behind our proposed hybrid time series model, explaining the model's architecture and training implementation. The prediction errors of our hybrid model in two distinct time series forecasting tasks are then compared with those of the SOTA models.

- **Chapter 6 - Conclusion**: This chapter concludes the project by providing a summary of the project's achievements and proposing potential directions for future research.

2 Literature Review

2.1 Smart Electric Vehicle Charging

Previous studies have developed various strategies for optimizing EV charging, targeting several key objectives. These include flattening charging profiles to mitigate demand spikes in the distribution network [13], minimizing power losses during the charging process [19][20], maximizing the main grid load factor to ensure a balanced power supply [21] and reducing charging costs to EV owners [22][23]. In general, studies on EV charging control fall intro three categories: effect of time-of-use (TOU) prices [24][25], centralized charging scheduling [19][26] and decentralized charging scheduling [27] [28].

On TOU prices, customers are charged a higher electric rate when there is larger demand for electricity. This is used to encourage users to charge their EVs during the night, when electricity demand is lower. Although this approach can shift demand and reduce demand spikes, recent studies [6] have shown that this can damage critical infrastructure such as inverters, which rely on the low night demand to cool down. Centralized scheduling requires a centralized structure to collect the complete information of all EVs and centrally optimize their charging schedule. Although this makes the optimization problem simpler as all information is available, the size of this centralized optimization increases with the increasing number of EVs, whilst having complete information of all EVs is often not feasible in practice [29]. Hence, in our research, a decentralized coordination scheme will be developed, a more realistic approach especially in the case where there is a high-penetration of EVs (large-scale, high-density EV charging).

Despite previous research in decentralized charging algorithms [30][31][32], existing EV scheduling algorithms assume charging parameters, such as the EV stay duration and user electricity demand for each session, are either perfectly known [31] in advance or directly provided by the user [17] [33]. However, as indicated by Wang et al., the randomness of user behaviours plays a significant role in EV scheduling problems [34], whilst estimating charging parameters has been shown to improve the robustness and accuracy of predictions

compared to using user inputs directly [14]. Building on this insight, our study leverages statistical and deep learning models to forecast these key charging parameters, aiming to enhance predictions of EV owners' charging behavior.

To the best of our knowledge, only a limited number of studies have employed this method in the past. Wang et al. [34] simplified the prediction process by developing aggregate joint probability distributions based on historical data from 100 users, focusing on arrival times, departure times, and energy demands. Although this approach streamlines model creation to just two distributions - arrival time versus stay duration and stay duration versus energy demand - it sacrifices prediction accuracy due to the high variance in user behavior, resulting in fat-tailed distributions. Conversely, Lee et al. [14] employed Gaussian Mixture Models (GMMs) to estimate individual charging parameters, but their reliance on a single model architecture for all users failed to account for varied charging behaviors, reducing prediction accuracy. Chung et al. [8] sought to mitigate this by proposing an ensemble machine learning algorithm aimed at recognizing distinct charging patterns among users. However, their dependence on individual historical data made their predictions vulnerable to noise and unforeseen events. Each of these methodologies presents unique drawbacks, suggesting the need for a more adaptive and nuanced approach to accurately predict EV owners' charging behavior.

2.2 Time Series Forecasting

A time series is a sequence of data points recorded at successive points in time, typically reflecting the observations of a specific process or phenomenon at regular time intervals. This arrangement allows for the analysis of the observed object's behavior and state over time [35]. Time series forecasting (TSF) is a classical forecasting task that predicts the future trend changes of a time series, using historical data and the a priori knowledge gained in the area, and has been widely used in real-world applications such as energy [36], transportation [37], meteorology [38], medicine [39] and finance [40].

Traditionally, statistical methods have played a pivotal role in forecasting time series data.

The autoregressive (AR) model, introduced by Yule in 1927 [41], marked the beginning of using statistical models for predicting univariate, smoothly varying time series. Subsequently, the moving average (MA) model [42], developed by Walker, and the autoregressive moving average (ARMA) model [43], which combines AR and MA models, were employed to solve univariate TSF problems. Despite their initial success, these traditional methods come with strict assumptions about the time series data, such as stationarity, linear correlations, normality, and independence, limiting their effectiveness in real-world applications.

Furthermore, time series data in practical applications frequently exhibit nonlinear characteristics, due to the combined effects of internal and external factors. Traditional statistical models, however, lack the capability to learn and represent these nonlinear dynamics effectively, leading to suboptimal forecasting accuracy in real-life applications [44]. This gap has prompted the exploration of artificial intelligence (AI) techniques for TSF, due to their superior ability to discover intricate features implied in time series data and their more powerful learning capability.

Traditional machine learning (ML) algorithms are effective solutions. Two types of classic ML algorithms, Support Vector Machines (SVMs) [45] and adaptive boosting (AdaBoost) [46], were developed in 1995 and have achieved great success in the field of TSF. However, these models suffer from poor generalization, exhibiting limited predictive accuracy that can often only be improved with complex manual feature engineering.

In recent years, the field of deep learning has developed rapidly and has become a powerful branch of machine learning. Many scholars have leveraged deep learning methods for time series forecasting, achieving high accuracy by adeptly capturing complex temporal dynamics and dependencies in data. Notable methodologies include the use of Convolutional Neural Networks (CNN) [47], Recurrent Neural Networks (RNN) [48], Long Short-term Memory (LSTM) Networks [49], Deep Belief Networks (DBN) [50], and, most recently, Tranformer-architecture models [51] and their variants [52][53]. However, all individual models suffer from deficiencies in certain aspects, including overfitting, high computational requirements, the need for large datasets and difficulties in capturing long-term dependencies.

As such, in our research, a combinatorial model approach will be adopted to improve the accuracy of predictions. In this approach, a number of models, ranging from traditional statistical methods to machine and deep learning models are combined together, aiming to produce an aggregate final prediction by merging individual forecasts from each model. Through meaningful signal processing techniques, this strategy takes advantage of the strengths of each model, enabling the combinatorial model to have more diverse data pattern recognition, reduced bias and variance and be less prone to overfitting, resulting in more accurate, reliable and robust forecasting.

3 Prediction of EV User Behavior

By examining the EV charging data of each user, the data patterns can be classified into four categories: linear, non-linear, clustered and scattered patterns [8]. This categorization acknowledges the inherent diversity and complexity within charging behaviors, underscoring the challenge that no single predictive model can effectively address all patterns. Moreover, while machine learning algorithms have shown impressive capabilities in predictive tasks, particularly in capturing complex non-linear patterns [54], they are often prone to overfitting, suffer from low-data efficiency [55], have high computational requirements and costs, are non-interpretable [56] and require significant hyperparameter tuning to perform well [57].

In response to these challenges, in our work, the EV users were classified according to their charging patterns and an ensemble machine learning algorithm was employed to choose the best model for each class out of seven different statistical and machine learning algorithms implemented. The models implemented include:

- Statistical Models
  - Mode with linear interpolation
  - Mode with repetition
- Machine Learning Models
  - Multiple Linear Regression (MLR)
  - K-Nearest Neighbours (KNN) Regression
  - Decision Tree (DT)
  - Random Forest (RF)
  - Support Vector Regression (SVR) with Radial Basis Function (RBF) Kernel
  - Diffusion-Based Kernel Density Estimation (DKDE)

This selection of models was deliberate, reflecting the distinct characteristics and strengths of each model in handling various EV charging pattern classes. Statistical models were chosen

for their simplicity, understandability, and robustness, as well as to provide a baseline against which to evaluate the machine learning models.

For linear patterns, Multiple Linear Regression (MLR) was selected for its direct applicability and the interpretability of its coefficients, which allow us to understand the effect of each independent variable on the dependent variable [58]. However, its limitations in addressing non-linear patterns necessitated the inclusion of Support Vector Regression (SVR). SVR enhances our ability to model both linear and non-linear phenomena, offering robustness to outliers and strong generalization capabilities [59].

For clustered patterns, Decision Trees (DT) and Random Forest (RF) regression present robust solutions as they are non-parametric models with hierarchical structures, which allows them to naturally adapt to the data's structure [60]. While DT offer a straightforward modeling approach, their susceptibility to overfitting is mitigated by RF models. RF improves accuracy by combining the outputs of multiple DT through ensemble learning, although its effectiveness depends on complex hyperparameter tuning, such as selecting the optimal number of trees. Moreover, this approach, may lead to slower predictions and higher computational costs. K-Nearest Neighbors (KNN) regression is also effective for clustered data [61], leveraging the proximity of data points to predict outcomes, showing excellent performance with small datasets and adaptability to new data.

Scattered patterns pose a significant challenge due to their lack of clear structure. Kernel estimation, particularly Diffusion-Based Kernel Density Estimation (DKDE), is employed to address this challenge [62]. This is because, DKDE does not assume any underlying distribution for the data generation process and can effectively model the probability density function in sparse areas, facilitating predictions based on the expected value.

3.1 Background Theory

3.1.1 Statistical Models

In statistics, the mode [63] is the value that appears most often in a set of data values. For a discrete random variable $\mathbf{X}$, the mode is the value x at which the probability mass

function (PMF) takes its maximum value. This can be described mathematically as shown in Equation 1.

$$x = \arg\max_{x_i} P(\mathbf{X} = x_i) \tag{1}$$

3.1.2 Multiple Linear Regression (MLR)

Multiple linear regression is a statistical technique used to model the mathematical relationship between two or more explanatory/independent variables and a response/dependent variable by fitting a linear equation to the observed data. The model of MLR with k explanatory variables and n observations is given in Equation 2

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + \epsilon_i \text{ for } i = 1, 2, ...n \tag{2}$$

where y_i is the dependent variable, $[x_{i1}, x_{i2}, .., , x_{ik}]$ are the independent variables used to predict y, β_0 is the bias, $\boldsymbol{\beta} = [\beta_1, ..., \beta_k]^T$ are the regression coefficients and ϵ_i is error term, which is also known as the residual that is used to account for the difference between the actual outcome and the prediction.

In order to calculate the regression coefficients, the ordinary least squares (OLS) algorithm which aims to minimize the sum of the squared residuals is employed. Mathematically, this can be formulated as an unconstrained quadratic minimization problem

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} S(\beta) \tag{3}$$

where the objective function S is given by

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{j=1}^{k} x_{ij}\beta_j \right|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 . \tag{4}$$

and matrix $\boldsymbol{X}$ is equal to

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} \tag{5}$$

This minimization problem has a unique solution, provided that the k columns of the matrix $\boldsymbol{X}$ are linearly independent, given by solving the so-called normal equations:

$$\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)\hat{\boldsymbol{\beta}} = \mathbf{X}^{\mathsf{T}}\mathbf{y}. \tag{6}$$

In Equation 6, the matrix $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ is known as the normal matrix or Gram matrix [64], and the matrix $\mathbf{X}^{\mathsf{T}}\mathbf{y}$ is known as the moment matrix of regressand by regressors [65]. Finally, $\hat{\boldsymbol{\beta}}$ is the coefficient vector of the least-squares hyperplane, expressed as

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}$$

A formal proof of the analytical solution to the OLS algorithm is provided in Appendix A.

When an analytical solution becomes impractical due to the large size of the dataset or the presence of linear dependence among the columns of the matrix $\boldsymbol{X}$, iterative optimization algorithms like gradient descent [66] can be employed to estimate $\hat{\boldsymbol{\beta}}$. These methods iteratively adjust the coefficients to minimize the cost function, offering a practical alternative for large-scale problems or when facing issues of multicollinearity.

Once the regression coefficients are calculated, they are combined with the independent variable values to predict the dependent variable's outcome as outlined in Equation 2.

3.1.3 K-Nearest Neighbors (KNN) Regression

K-Nearest Neighbours (KNN) [67] is a non-parametric supervised learning method used for both classification and regression. In our research, the regression model is used since the data labels, whether for stay duration or energy consumption prediction, are continuous instead of discrete variables.

In KNN Regression, the value of K, which is the number of nearest neighbors for making predictions, is first determined. This is because, K is a crucial hyperparameter that significantly influences the model's performance. A smaller value of K decreases bias and captures complex patterns, whilst a larger value of K improves generalization and reduces sensitivity to noise, highlighting a trade-off between capturing complex patterns (low K) and generalizing well (high K). To identify the closest neighbors to a new input data point, various distance metrics can be employed, includean the Euclidean, Manhattan and Hamming distances. Given the continuous nature of our data, the Euclidean distance is chosen, as defined in Equation 7, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are n-dimensional points in Euclidean space.

$$d(\boldsymbol{p}, \boldsymbol{q}) = \|\boldsymbol{p} - \boldsymbol{q}\| = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{7}$$

For predicting a new query point's value, KNN computes the distances between this point and all others in the dataset, selecting the K closest points. The prediction is then the average of the labels of these nearest neighbors, assuming uniform contribution from each neighbor, as indicated in Equation 8.

$$\hat{y}_{query} = \frac{1}{K} \sum_{i=1}^{K} y_i \tag{8}$$

It is important to note that although KNN regression is straightforward to implement and inherently non-parametric - eliminating the need for training by performing all calculations at the inference stage - it may become computationally intensive with large datasets. This is because, to make a prediction, it is essential to calculate distances between the query point and every other point in the dataset.

3.1.4 Decision Tree (DT)

Decision tree (DT) regression [68] is a non-parametric, supervised learning method capable of modeling complex data with nonlinear relationships between features and a continuous target variable. Unlike its classification counterpart, regression trees predict continuous outcomes by splitting the data into homogeneous regions that minimize a specific error criterion, in this case the mean square error (MSE). Each node in the tree represents a decision point where the dataset is split, and the leaves represent the output values based on the mean of the target variable within that region.

The core of the regression decision tree algorithm involves partitioning the input space into M distinct regions $(R_1, R_2, \ldots, R_M)$, each associated with a unique output value (O_m). This partitioning is governed by a function π, mapping each element in the input space to one and only one region. If we already knew a division of the space into regions, we would set O_m, the constant output for region R_m, to be the average of the training output values in that region. Mathematically, for a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, and indicator set I_m identifying samples within R_m, such that $I_m = \{i \,|\, x^{(i)} \in R_m\}$ the output value is given by

$$O_m = \operatorname*{average}_{i \in I_m} y^{(i)} \tag{9}$$

The error within a region, E_m, can be articulated as the sum of squared differences between the observed values and O_m, indicating the model's fit within that region.

$$E_m = \sum_{i \in I_m} (y^{(i)} - O_m)^2 \tag{10}$$

The algorithm aims to select partitions that minimize the overall error across all regions, incorporating a regularization term (λ) to penalize complexity and prevent overfitting, thereby ensuring the model's generalizability and predictive robustness.

$$\lambda M + \sum_{m=1}^{M} E_m \tag{11}$$

Given the NP-complete nature of optimizing the partition across the input space [69] defined in Equation 11 , a greedy approach is employed to construct the tree. Starting from the root, the dataset is recursively split by choosing the best feature and split point, s, that minimize the sum of squared errors post-split. This process is known as binary recursive partitioning and is described in Algorithm 3.1 [70].

Algorithm 3.1 BuildTree Algorithm (I, k)

if $|I| \leq k$ **then**
  $\hat{y} \leftarrow \text{average}_{i \in I} y(i)$
  return LEAF(value $= \hat{y}$)
else
  for each split dimension j and split value s **do**
    $I_{j,s}^{+} \leftarrow \{i \in I | x_j(i) \geq s\}$
    $I_{j,s}^{-} \leftarrow \{i \in I | x_j(i) < s\}$
    $\hat{y}_{j,s}^{+} \leftarrow \text{average}_{i \in I_{j,s}^{+}} y(i)$
    $\hat{y}_{j,s}^{-} \leftarrow \text{average}_{i \in I_{j,s}^{-}} y(i)$
    $E_{j,s} \leftarrow \sum_{i \in I_{j,s}^{+}} (y(i) - \hat{y}_{j,s}^{+})^2 + \sum_{i \in I_{j,s}^{-}} (y(i) - \hat{y}_{j,s}^{-})^2$
  end for
  $(j^*, s^*) \leftarrow \arg\min_{j,s} E_{j,s}$
  return NODE$(j^*, s^*, \text{BuildTree}(I_{j^*,s^*}^{+}, k), \text{BuildTree}(I_{j^*,s^*}^{-}, k))$
end if

where

- k is a hyperparameter denoting the maximum allowed leaf size
- $I_{j,s}^{+}$ and $I_{j,s}^{-}$ partition I into points whose feature value in dimension j is greater than or equal to s, and less than s, respectively
- $\hat{y}_{j,s}^{+}$ and $\hat{y}_{j,s}^{-}$ are the average target values for the subsets $I_{j,s}^{+}$ and $I_{j,s}^{-}$, respectively
- j^*, s^* are the feature and split point that result in the minimum error $E_{j,s}$

In practice, the 'BuildTree' function is initially called on the entire dataset ($I = \{1, \ldots, n\}$), and this initial call can lead to many recursive calls to itself. Each call of the 'BuildTree' function evaluates all possible splits across each of the d dimensions. However, in each dimension, only splits between two data points need to be considered (as any other split will give the same error on the training data), resulting in a total of $O(dn)$ splits per call.

3.1.5 Random Forest (RF)

Random Forest (RF) regression [71] is an ensemble learning approach that operates by constructing a multitude of decision trees at training time. For regression tasks, the mean or average prediction of the individual trees is returned. Formally defined, the RF model can be expressed as follows:

$$g(x) = \frac{1}{N_{\text{tree}}} \sum_{i=1}^{N_{\text{tree}}} f_i(x) \tag{12}$$

where $g(x)$ signifies the RF model, $f_i(x)$ denotes the i^{th} DT model, and N_{tree} represents the total number of decision trees within the ensemble. Each decision tree, $f_i(x)$, is independently constructed from a bootstrap sample, a dataset drawn with replacement from the original training data, ensuring diversity among the trees. The RF algorithm further introduces an additional layer of randomness by selecting a random subset of features for node splitting across the trees. This dual strategy of bootstrapping and feature randomness effectively decorrelates the trees, thereby significantly enhancing the model's generalization capabilities.

By leveraging the average output of multiple DT models, each trained on different subsets of the dataset, RF regression substantially improves predictive accuracy. Moreover, the inherent randomness in feature selection across trees ensures that the ensemble model benefits from a diversified set of predictors, reducing the likelihood of overfitting.

The construction of a Random Forest model involves several hyperparameters: B, the number of trees; m, the number of variables considered for splitting at each node; and n, the size of the bootstrap sample. The algorithm for constructing a Random Forest model [71] is outlined in Algorithm 3.2.

Algorithm 3.2 RandomForest Algorithm $(D; B, m, n)$

1: **for** $b = 1,...,B$ **do**
2: Draw a bootstrap sample D_b of size n from D
3: Grow a tree T_b on data D_b by recursively:
4: Select m variables at random from the d variables
5: Pick the best variable and split point among the m variables
6: Split the node
7: **end for**
8: **return** tree T_b

3.1.6 Support Vector Regression (SVR) with Radial Basis Function (RBF) Kernel

SVR [72] is a type of support vector machine that supports linear and non-linear regression. SVR operates by constructing a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for regression, while maintaining all the main features that characterize the maximal margin algorithm of a support vector machine.

Unlike traditional regression models, which attempt to minimize the residual sum of squares between the observed responses in the dataset and the responses predicted by the linear approximation, SVR seeks to fit the error within a certain threshold. Specifically, in $\epsilon-$SVR [73], the goal is to find a function $\hat{y}(x)$ that has at most ϵ deviation from the obtained targets y_i for the training data, ignoring the outliers that locate outside of the ϵ-tolerence band. Given a training dataset

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X},$$

where $\mathcal{X}$ denotes the space of the input data, SVR can be expressed as

$$\hat{y}(\mathbf{x}) = \langle \boldsymbol{\omega}, \mathbf{x} \rangle + b \quad \text{with} \quad \boldsymbol{\omega} \in \mathcal{X}, b \in \mathbb{R}, \tag{13}$$

where $\boldsymbol{\omega}$ and b are the solution to the SVR optimization problem outlined in Equations 14 and 15.

$$\min_{\boldsymbol{\omega}, b, \xi, \xi^*} \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \tag{14}$$

subject to:

$$\begin{aligned} y_i - \langle \boldsymbol{\omega}, \mathbf{x}_i \rangle - b &\leq \epsilon + \xi_i, \\ \langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b - y_i &\leq \epsilon + \xi_i^*, \\ \xi_i, \xi_i^* &\geq 0. \end{aligned} \tag{15}$$

In this formulation, $\boldsymbol{\omega}$ represents the weight vector, b is the bias term, and ξ_i, ξ_i^* are slack variables introduced to soften the margin in order to handle the problem of infeasible e-

precision constraints. The constant $C > 0$ is a regularization parameter which controls the trade-off between the flatness of $\hat{y}(\mathbf{x})$ (which is $||\boldsymbol{\omega}||^2$) and the number of training data points up to which deviations larger than ϵ are tolerated.

The SVR optimization problem can be solved by the Lagrange multipliers method [74], often using quadratic programming [75], and the solution is given by

$$\begin{aligned} \boldsymbol{\omega} &= \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \Phi(\mathbf{x}_i), \\ f(\mathbf{x}) &= \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}) + b, \end{aligned} \tag{16}$$

where α_i, α_i^* are the Lagrange multipliers in which $\alpha_i, \alpha_i^* \in [0, C]$, $\Phi(\mathbf{x}_i)$ is a transformation function mapping input data into a higher-dimensional space, and $k(\mathbf{x}_i, \mathbf{x}) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle$ is the RBF kernel function

$$k(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2} \right), \tag{17}$$

with $\sigma \in \mathbb{R}$ being the kernel parameter, which determines the width of the RBF kernel. This parameter controls the smoothness of the decision boundary and the model's sensitivity to the distance between data points. Specifically, a smaller σ results in more complex, tightly fitted decision boundaries prone to overfitting, while a larger σ yields smoother, more generalized decision boundaries that may underfit the data.

The RBF kernel effectively enables the performance of linear regression in the transformed feature space by implicitly mapping input vectors into a higher-dimensional space where a linear regressor is then applied. This enables the model to capture complex, non-linear patterns without requiring explicit computations in the high-dimensional space. The full details of the SVR formulation with an RBF kernel can be found in [76].

3.1.7 Diffusion-Based Kernel Density Estimation (DKDE)

Kernel Density Estimation (KDE) is a nonparametric approach for estimating the probability density function (PDF) of a random variable. This estimator circumvents the need for a pre-specified parametric model, accommodating the multimodal and asymmetric nature of empirical distributions.

Given an observed dataset $X = [X_1, X_2, ..., X_N]$, the probability density function can be estimated according to KDE as:

$$\hat{P}(x) = \frac{1}{N}\sum_{i=1}^{N} K_h(x - X_i) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - X_i}{h}\right) \tag{18}$$

where N is the number of observed data points, K denotes the kernel function and h is the bandwidth of the kernel [77].

Gaussian Kernel Density Estimation (GKDE) is a specific type of Kernel Density Estimation (KDE) where the kernel function K is chosen to be a Gaussian (or normal) distribution. This choice is popular due to the Gaussian distribution's smooth, bell-shaped curve, which is symmetric around the mean and decreases towards the tails, as well as, its mathematical properties, such as being infinitely differentiable. The Gaussian kernel K$(\cdot)$ is defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}u^2) \tag{19}$$

Bandwidth h defines the shape of the kernel function, thus it is a deterministic factor to the performance of the estimator. A large h oversmooths the density function that masks the structure of data whilst a small h generates a spiky one that makes the interpretation difficult. Finding an optimal h is essential to balance between underfitting and overfitting, aiming to minimize the error between the estimated and actual density functions. However, there is a bias-variance trade-off for the bandwidth selection. A larger bandwidth reduces the variance of the estimated probability density function, $\hat{P}_{GKDE}(x)$ at the cost of increased bias, meaning the estimate may be smooth but not accurately reflect the true distribution. In contrast, a smaller bandwidth reduces the bias, making the estimate closer to the data's

actual distribution, but increases variance, introducing noise and potentially overfitting to the data. Silverman's rule of thumb [78], also known as the normal reference rule, provides a simple solution for the optimal bandwidth, with the assumption that the actual density has Gaussian normal distribution. However, this method usually leads to an oversmoothed result in multimodal models such as EV user charging behaviors.

The DKDE [79] method extends GKDE by deriving the optimal bandwidth through the diffusion process, linking the estimation to the heat equation in physics. This is becuase, the KDE in Equation 18 can be expressed in an alternative form

$$\hat{P}_{KDE}(x; t_{diff}) = \frac{1}{N}\sum_{i=1}^{N}\phi(x, X_i; t_{diff}) \tag{20}$$

where

$$\phi(x, X_i; t_{diff}) = \frac{1}{\sqrt{2\pi t_{diff}}}\exp\left(-\frac{(x - X_i)^2}{2t_{diff}}\right) \tag{21}$$

in which $\sqrt{t_{diff}} = h$ is defined as in Equation 18.

It is interesting to note that GKDE in Equation 20 is the unique solution to the Fourier heat equation [79]

$$\frac{\partial \hat{P}(x; t_{diff})}{\partial t} = \frac{1}{2}\frac{\partial^2 \hat{P}(x; t_{diff})}{\partial x^2}, \quad x \in \mathbb{E},\ t_{diff} > 0,\ \Delta \in \mathbb{R} \tag{22}$$

with initial condition

$$\hat{P}(x; 0) = \frac{1}{N}\sum_{i=1}^{N}\delta(x - X_i) \tag{23}$$

where $\hat{P}(x; 0)$ represents the empirical density of X, and $\delta(x - X_i)$ is the Dirac measure at X_i.

The Neumann boundary condition is defined as:

$$\left.\frac{\partial \hat{P}(x; t_{diff})}{\partial t}\right|_{x=1} = \left.\frac{\partial \hat{P}(x; t_{diff})}{\partial t}\right|_{x=0} = 0 \tag{24}$$

Making use of the link between GKDE and Fourier heat equation, finding the optimal bandwidth of Equation 20 is equivalent to finding the optimal mixing time t^* of the diffusion process governed by Equation 22. Considering all these conditions and the finite domain [0, 1], the analytical solution of Equation 22 is obtained by

$$\hat{P}_{DKDE}(x; t^*_{diff}) = \frac{1}{N} \sum_{i=1}^{N} \kappa(x, X_i; t^*_{diff}) \quad x \in [0, 1] \tag{25}$$

in which the kernel function is given by

$$\kappa(x, X_i; t^*_{diff}) = \sum_{k=-\infty}^{\infty} \left[\phi(x, 2k + X_i; t^*_{diff}) + \phi(x, 2k - X_i; t^*_{diff}) \right] \quad \text{for } x \in [0, 1] \tag{26}$$

The full details of the DKDE formulation can be found in [79].

3.2 Methodology

3.2.1 Problem Formulation

As stated in Section 1.1, in the first stage of the project we aim to predict the behaviour of the EV user. As such, the objective is to predict each EV user's stay duration and energy demand based on their historical charging data. For each charging session, a 5-tuple of parameters is used to describe a charging behavior:

$$s \triangleq (u_{id}, t_s, t_d, d_w, e) \tag{27}$$

where u_{id} is the unique identifier (user ID) for each user in our system; t_s and t_d denote start time and stay duration, respectively; d_w denotes day of week; and e denotes energy consumption. The predicted charging parameters are essential for the online EV charging scheduling algorithms we will use to determine an optimal solution/charging schedule. This is because, as shown in [80], algorithms that use information about departure times and energy demands perform significantly better in optimizing EV charging than those that do not. Thus, once a user initiates a charging session, the predictions of stay duration and energy consumption are required for the scheduling services to determine the energy allocation schedule.

In order to predict *stay duration* the *start time* and *day of the week* will be used since users may have their fixed weekly working schedules. Therefore, the prediction of stay duration ($\hat{t}_d$) can be expressed as follows:

$$\hat{t}_d = f_d(t_s, d_w) \tag{28}$$

On the other hand, energy consumption is related to *start time, day of week, and stay duration*, such that:

$$\hat{e} = f_e(t_s, d_w, \hat{t}_d) \tag{29}$$

As shown in 29, when predicting energy consumption, the predicted stay duration is utilized as the actual stay duration is assumed to be unknown, indicating that our algorithm requires

only the EV arrival time to make predictions.

3.2.2 Data Collection and Processing

In order to enhance the practical application of our methodology, we will utilize real charging data instead of synthetic datasets. Moreover, because public charging data is more unpredictable compared to residential charging data – which tends to be more consistent, as noted in [81] – we will use a public charging dataset. Specifically, we employed the Adaptive Charging Network (ACN) dataset for this research, collected from a charging station at the JPL national research lab in La Cañada, CA. This site was selected as it provides access to a total of 50 EV chargers, offering a substantial dataset with real data for large-scale EV charging analysis. Additionally, it is representative of a typical workplace charging environment. Moreveor, as the ACN dataset is a trustworthy open-source resource that has been widely used in prior studies [14, 17, 33, 82–84], this enables us to benchmark our algorithms against established literature.

Data spanning three months, from October 1, 2018, to December 31, 2018, were selected to allow for a direct comparison with the best predictive algotithm for this dataset, to our knowledge [14]. As not every user in the dataset has a charging history that is long enough for data analysis and prediction, only users with a minimum of 20 charging sessions were included. The dataset was split into training and validation sets covering the first two months (October 1, 2018, to November 30, 2018), with the remaining month (December 2018) serving as the test set. This division aims to rigorously evaluate our algorithm's predictive accuracy against historical data.

For the analysis, charging records that did not include a user ID were removed. The start time and stay duration of each charging session were converted to hours, and the days of the week were encoded as discrete values, ranging from 1 (Monday) to 7 (Sunday). Additionally, in instances where energy consumption was erroneously recorded beyond the physical capacity of the charging device, the affected record was corrected by substituting it with the maximum observed value from its historical data.

3.2.3 Model Training

Our ensemble machine learning algorithm, designed to choose the best model for each user from the seven statistical and machine learning models detailed in Section 3.1, required the initial implementation of these models. Given that most of these models are available through Python's built-in open-source `scikit-learn` machine learning library [85] or other open-source packages our efforts primarily focused on optimizing their hyperparameters for improved performance. We employed cross-validation and grid search techniques for this purpose, enabling precise model fine-tuning to enhance predictive accuracy. The implemented models, along with their respective hyperparameters and grid search configurations used, are detailed in Table 3.1.

Predictive Model	Hyperparameters	Grid Search Values
Mode	None	–
MLR	None	–
KNN Regression	K = Number of nearest data points to use for prediction	$K \in (1, 5)$
DT	D = Maximum Tree Depth, S = Minimum number of samples to split node	$D \in (1, 21)$, $S \in (2, 11)$
RF	T = Number of trees, S = Minimum number of samples to split node,	$T \in \{10, 20, 50\}$, $S \in (2, 11)$
	D = Maximum Tree Depth, F = Features to consider for best split	$D \in \{2, 5, 7, 10, 12\}$, $F \in \{1.0, sqrt\}$
SVR with RBF Kernel	C = Controls training dataset misclassification level,	$C \in \{0.1, 1, 10, 100\}$
	γ = Controls flexibility of model, ϵ = Sets a penalty-free error tolerance margin	$\gamma \in \{0.1, 1, 10\}, \epsilon \in \{0.001, 0.01, 0.1\}$
DKDE	N = Number of discrete points used to construct the probability density function	$C \in \{32, 64, 128, 256, 512\}$

Table 3.1: List of predictive models implemented along with their respective hyperparameters and grid search values.

To enhance reproducibility and for future reference, the packages used to implement each model are listed in Table 3.2.

Predictive Model	Python Package
Mode	None
MLR	`sklearn.linear_model.LinearRegression`
KNN Regression	`sklearn.neighbors.KNeighborsRegressor`
DT	`sklearn.tree.DecisionTreeRegressor`
RF	`sklearn.ensemble.RandomForestRegressor`
SVR with RBF Kernel	`sklearn.svm.svr`
DKDE	`KDE-diffusion` package from `PyPI`

Table 3.2: List of predictive models and the corresponding python packages used for their implementation.

3.2.4 Integrating Correlations in Our Predictive Algorithm

In order to mitigate the impact of data variability and unpredictable events in an individual EV owner's historical data, we will utilize correlations between users. To analyze

these correlations based on arrival times, we discretized the times into 30-minute and 1-hour intervals, encoding them as integer sets ranging from 0 to 47 for the 30-minute bins and 0 to 23 for the 1-hour bins, respectively. This discretization helps in managing the variability in arrival times by grouping similar behaviors. Two distinct correlation metrics were tested to capture different aspects of user relationships: the Pearson Correlation Coefficient (PCC) [86] and the Cosine Similarity Measure [87]. The PCC, r_{xy}, defined as

$$r_{xy} = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}} \tag{30}$$

where x_i and y_i are individual sample points from random variables (X, Y) representing the discretized arrival times of two users, and $\overline{x}$ and $\overline{y}$ are their respective means, was chosen for its ability to measure both the direction and magnitude of linear relationships between users. The PCC value ranges between -1 and 1, with an absolute value of 1 implying a perfect linear relationship, and a value of 0 indicating no linear dependency between the variables.

In contrast, the Cosine Similarity Measure, S_c, defined as

$$S_c(\mathbf{A}, \mathbf{B}) := cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{31}$$

where $\mathbf{A}$ and $\mathbf{B}$ are vectors representing the discretized arrival times of two users and A_i and B_i are the components of these vectors corresponding to the number of arrivals in each discretized time, was selected for its adeptness in handling sparse vectors. The cosine similarity always belongs to the interval $[-1, 1]$ as two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite vectors have a similarity of -1.

Consequently, this approach provided us with four distinct options for assessing user correlations: Pearson and Cosine Similarity Measures applied to both 30-minute and 1-hour intervals. The four correlation options were integrated into each of the seven predictive algorithms implemented, as listed in Table 3.1. To identify the most effective combination, cross-validation was employed on the two month training data. Moreover, to determine the

optimal number of correlated users to include in our prediction calculation, a threshold was established for the absolute value of the correlation, ranging between 0 and 1. Setting a higher threshold ensures the inclusion of only highly correlated users, minimizing the risk of incorporating inaccurate information. However, this approach has a trade-off, as it potentially reduces the pool of users considered, which might affect the predictions' ability to withstand data variability and unexpected daily fluctuations due to unpredictable events. As such, the correlation threshold was varied between 0.55 and 0.85, with adjustments made in 0.05 increments, allowing us to determine the optimal value through cross-validation with grid search.

The overall predictions for each user's stay duration and energy consumption were then calculated by summing the predictions of the individual and their most correlated users. This sum was weighted by the correlation values, effectively using a weighted average approach, as shown in Equations 32 and 33 respectively

$$\bar{t}_{di} = \hat{t}_{di} + c_{ij} \times \hat{t}_{dj} + \cdots + c_{ik} \times \hat{t}_{dk} \tag{32}$$

$$\bar{e}_i = \hat{e}_i + c_{ij} \times \hat{e}_j + \cdots + c_{ik} \times \hat{e}_k \tag{33}$$

where c_{ij} represents the correlation between user i and user j.

3.2.5 Ensemble Machine Learning Algorithm

By examining the EV charging data of each user, the data patterns can be classified into four categories: linear, non-linear, clustered and scattered patterns. Due to the inherent diversity and complexity within charging behaviors, there is no single predictive model for all users. Consequently, an ensemble machine learning algorithm was employed to choose the best model for each class out of the seven different statistical and machine learning models implemented, listed in Table 3.1. This selection of models was deliberate, reflecting the distinct characteristics and strengths of each model in handling the various EV charging pattern classes, as highlighted in the introduction of Section 3.

To underpin our model selection, we began by quantifying the entropy and sparsity of each dataset. This approach is justified by our observations from Figure 3.1 which indicate a positive correlation between the Symmetric Mean Absolute Percentage Error (SMAPE) – our chosen metric for evaluating model performance – and data entropy, alongside a negative correlation with data sparsity. These correlations were consistent across both datasets analyzed, specifically the arrival time versus duration data, which will be used for stay duration prediction, and duration versus energy consumption data, which will be used for energy consumption prediction, underscoring the importance of these data characteristics in influencing model effectiveness.

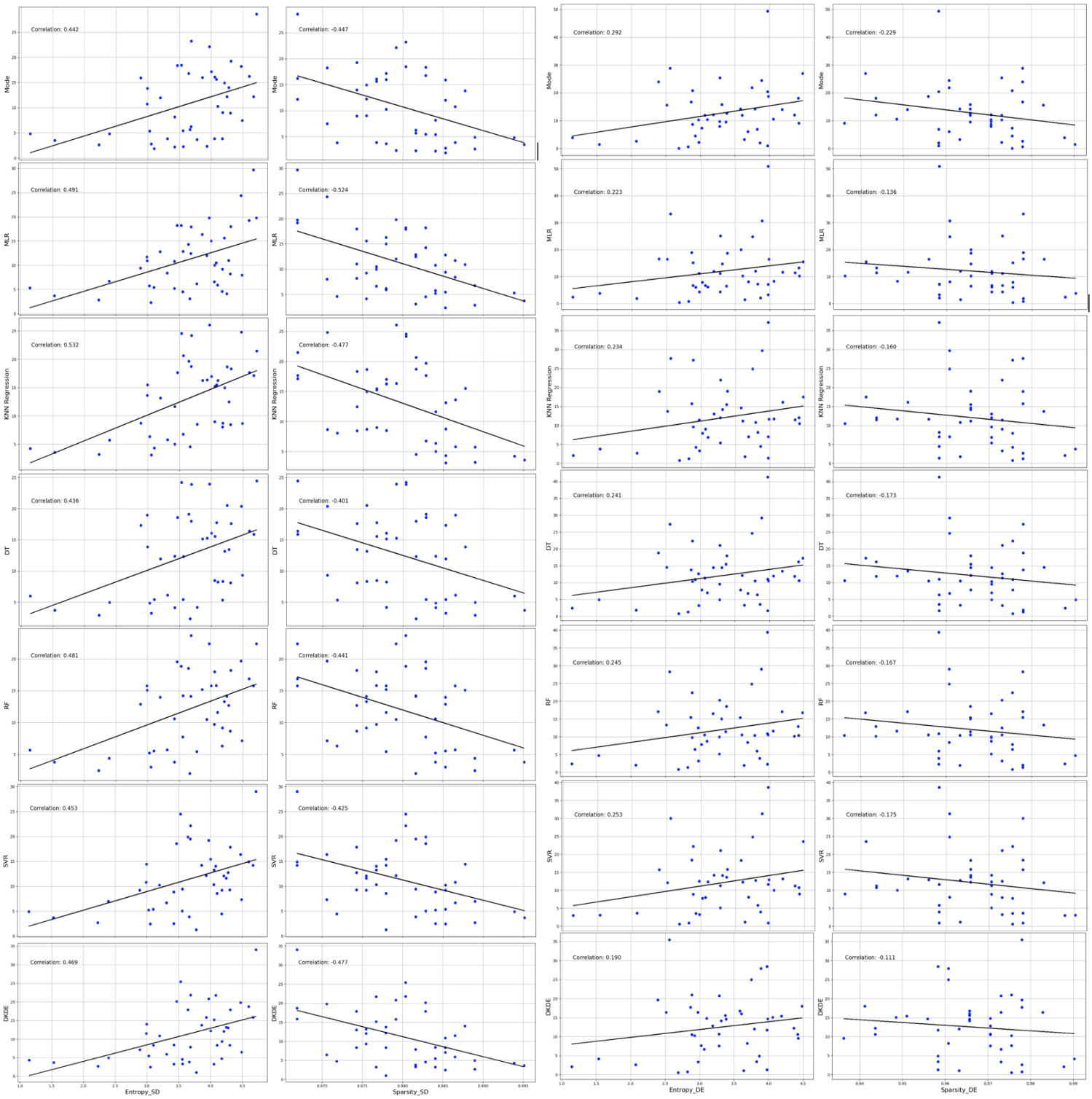
Consequently, in our study, we account for the effects of both entropy and sparsity by basing the ensemble decision on a defined threshold ratio, $R = entropy/sparsity$. The algorithm is designed to select one optimal model for users with a ratio below this threshold (low-ratio users) and another optimal model for users with a ratio above this threshold (high-ratio users), thereby tailoring the approach to the specific data characteristics of each user group.

In order to calculate the data entropy, the joint entropy was used as it characterizes the uncertainty of a set of variables. The start time and duration were rounded to the closest half hour and mapped into a set of integers $\in$ [0,47], and the energy consumption was rounded to the closest integer. The joint entropy was then calculated as shown in Equation 34

$$E(X,Y) = -\sum_{x\in X}\sum_{y\in Y} P(x,y)\log_2\left[P(x,y)\right] \tag{34}$$

where x and y are the two variables in dataset X and Y, respectively, and $P(x,y)$ is the joint probability of the two variables. Here, X and Y refer to either arrival time and stay duration when predicting stay duration, or to stay duration and energy consumption when predicting energy consumption.

Sparsity is defined as the number of zero entries divided by the total number of entries. Intuitively, if sparsity is high the data is less variant because most entries are repeated. On the other hand, for low sparsity, the data is more scattered. As with entropy, start time



(a) Dataset of Arrival time vs Stay Duration (b) Dataset of Duration vs Energy Consumption

Figure 3.1: Comparative analysis of SMAPE (%) against entropy and sparsity across the seven predictive models for both the stay duration and energy consumption datasets. For both datasets, SMAPE increases with entropy, illustrated by the upward-sloping lines of best fit in the left column, and decreases with sparsity, evident from the downward-sloping lines of best fit in the right column of the subfigures.

vs. duration data and duration vs. energy consumption data were analyzed. Each dataset was arranged in a 2D matrix for each user and each charging session was assigned to its corresponding entry (with some entries having multiple sessions). The sparsity equation for each user is provided in Equation 35.

$$Sparsity = \frac{Number\ of\ Zero\ Entries}{Number\ of\ Total\ Entries} \tag{35}$$

To determine the optimal value of R to use as a threshold in our ensemble algorithm, we varied the threshold from 3 to 4.5 in increments of 0.5. This range was selected because it includes all possible values of the ratio observed among users for both datasets. For each threshold setting, we evaluated all models incorporating correlation, identifying the best model for users with a small R and users with a large R for each dataset. Both the optimal threshold and corresponding models for each user group were obtained through cross-validation.

3.3 Experimental Results

In assessing the predictive performance of the models, the Symmetric Mean Absolute Percentage Error (SMAPE) will be used for both stay duration and energy consumption predictions. SMAPE is defined as follows:

$$\text{SMAPE}(x) := \sum_{j \in \mathbf{U}} \frac{1}{|\mathbf{U}|} \sum_{i \in \mathbf{A}_j} \frac{1}{|\mathbf{A}_j|} \left| \frac{x_{i,j} - \hat{x}_{i,j}}{x_{i,j} + \hat{x}_{i,j}} \right| \times 100\% \tag{36}$$

where $\mathbf{U}$ is the set of all users in the testing dataset X_{test} and $\mathbf{A}_j$ denotes the set of charging sessions for user j, where $j \in \mathbf{U}$.

The choice of SMAPE as a metric is motivated by several factors:

i) SMAPE is a unit free percentage error which facilitates straightforward comparisons across different types and scales of data.

ii) It mitigates the influence of small-valued data points, which can disproportionately

affect the overall error measurement.

iii) SMAPE is widely used in evaluating EV charging prediction accuracy, making it easier for comparisons.

To evaluate how i) correlations and ii) our ensemble strategy enhance the predictive performance of our models for stay duration and energy consumption, we obtained three distinct sets of results. Initially, we calculated the SMAPE for each individual model. Subsequently, we integrated correlations between users into these individual models. Finally, we combined the enhanced models that incorporate user correlations into our ensemble algorithm.

3.3.1 Results of Individual Models

The prediction errors for each of the seven predictive models, as well as for direct user inputs, are summarized in Table 3.3. As observed from Table 3.3, the DKDE model performs best in predicting stay duration. Surprisingly, despite its simplicity, the mode achieves the best results for predicting energy consumption. Moreover, the prediction error for energy consumption is generally higher than that for stay duration. This occurs because the predictions for energy consumption rely on the **predicted** stay duration, thereby propagating some of the initial error from stay duration estimates into the energy consumption predictions. In contrast, stay duration predictions utilize the actual arrival times, thus eliminating this source of error. Finally, it is important to note that all predictive models demonstrate lower SMAPE values compared to direct user inputs, underscoring the advantages of using predictive methods over reliance on potentially inaccurate user inputs, as explained in Section 1.1.

3.3.2 Results of Individual Models with Correlations

In order to observe the effect of integrating correlations on the predictive performance of each of our models, we first utilized cross-validation to select between using the Pearson Correlation Coefficient and cosine similarity, and to determine the optimal threshold value, as explained in Section 3.2.4. The prediction errors for the seven predictive models **with**

Model	SMAPE(d)%	SMAPE(e)%
User Input	18.60	26.90
Individual Mode	11.23	**11.88**
Population Mode	13.45	35.54
MLR	11.10	12.03
K-NN Regression	13.03	12.04
Decision Tree	12.50	12.07
Random Forest	11.99	12.02
SVR with RBF kernel	11.28	12.14
DKDE	**11.18**	12.53

Table 3.3: Comparative SMAPE performance of user inputs and predictive models for predictions without correlations. Note that the SMAPE scores of user inputs are reported directly from [14].

correlations, along with the selected correlation metric and optimal correlation threshold value, are detailed in Table 3.4.

By comparing the prediction errors between Table 3.3 and Table 3.4 , we can conclude that the integration of correlations consistently enhances prediction accuracy for both stay duration and energy consumption across **all** models. Specifically, the mode exhibits the largest decrease in prediction error for stay duration, with the SMAPE reduced by 12.4%, reaching a value of 9.83%. Conversely, for energy consumption, the most significant reduction occurs in the MLR model, where employing correlations leads to a 16.4% decrease in SMAPE, achieving a value of 10.06%. These results highlight the substantial impact incorporating correlations can have on the predictive performance of models. By utilizing data from correlated users instead of relying solely on the individual's historical data, the models can more effectively capture complex patterns and mitigate the effects of unpredictable events and data variability in individual EV charging behaviors.

Furthermore, as observed from Table 3.4, the correlation threshold value for all models is high. This indicates that optimal performance is achieved when only including highly correlated users, as a strong correlation is defined to be greater than or equal to a magnitude of 0.7 [88]. This is because incorporating correlations that are not strong can severely harm the predictions due to the introduction of inaccurate and irrelevant information for the specific user of interest.

Model	SMAPE(d)%	SMAPE(e)%	Corr. Metric	Corr. Value
Individual Mode	**9.83** (↓ 12.4%)	10.13 (↓ 14.7%)	Cosine Similarity	0.75
MLR	10.90 (↓ 1.80%)	**10.06** (↓ 19.6%)	PCC	0.80
K-NN Regression	11.24 (↓ 13.7%)	11.05 (↓ 8.23%)	Cosine Similarity	0.80
Decision Tree	10.89 (↓ 12.9%)	10.67 (↓ 11.6%)	Cosine Similarity	0.80
Random Forest	10.51 (↓ 14.1%)	10.63 (↓ 11.6%)	Cosine Similarity	0.85
SVR with RBF kernel	10.49 (↓ 7.00%)	10.68 (↓ 12.0%)	Cosine Similarity	0.75
DKDE	10.54 (↓ 5.72%)	12.12 (↓ 3.27%)	PCC	0.70

Table 3.4: Comparative SMAPE performance of predictive models with correlations. The percentage decrease in error for each model when using correlations compared to not is enclosed in brackets.

3.3.3 Results of Ensemble Machine Learning Algorithm

Given the diverse behaviors of different users, a 'one-size-fits-all' model is not effective, prompting us to adopt an ensemble learning approach. As detailed in Section 3.2.5, the first step involves identifying the optimal threshold for the entropy-sparsity ratio, R, for each of the two datasets: arrival time versus stay duration and stay duration versus energy consumption. Additionally, it is crucial to select the most appropriate model for the two distinct groups of EV users: those with low R and those with high R.

For predicting stay duration (arrival time vs stay duration dataset), the optimal threshold was found to be $R_{sd} = 3$. For users with R_{sd} below this threshold, the DKDE model is applied, while for those with R_{sd} above it, the mode is utilized. For energy consumption prediction (stay duration vs energy consumption dataset), the optimal threshold was set at $R_{de} = 3.5$. Here, the MLR model is used for users with an R_{de} below the threshold, and the DKDE model is applied for those exceeding it.

The flowchart for the proposed ensemble predicting algorithm, utilizing the optimal models and thresholds identified above, is depicted in Figure 3.2.

By employing our ensemble machine learning algorithm, detailed in Section 3.2.5 and graphically depicted in Figure 3.2, we further reduced the SMAPE for stay duration and energy consumption predictions to **9.32**% and **9.59**% respectively. These values represent percentage decreases of 5.5% and 4.7% compared to our best single-model predictions.

The results of our work, in comparison to those achieved in previous studies, are sum-

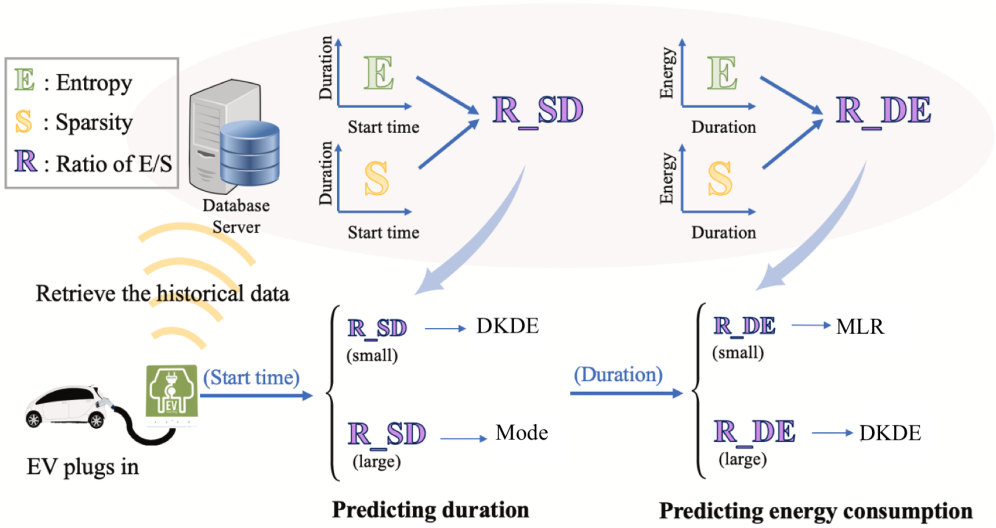


Figure 3.2: Flowchart of the proposed ensemble prediction algorithm. Historical data is used to calculate the entropy-sparsity ratios for stay duration (R_{sd}) and energy consumption (R_{de}) for each user. Upon their arrival, the algorithm utilizes these ratios to determine the appropriate model for predicting stay duration based on arrival time and energy consumption based on predicted stay duration for each user.

marized in Table 3.5. As observed in Table 3.5, our results surpass those of all previous works that employed similar evaluation metrics. Compared to [8], our work achieves a lower prediction error for energy consumption, but does not show improvement in stay duration predictions. This discrepancy is most likely attributed to the inclusion of both residential and non-residential data in [8], given that residential charging behavior tends to be more consistent, making predictions easier. It is important to note, however, that all studies except [14] utilized different datasets, which makes direct comparisons somewhat inappropriate.

Source	Dataset Used	Models Used	SMAPE of Stay Duration (%)	SMAPE of Energy Consumption (%)
[8]	UCLA Campus (historical charging data) & Residential UK charging data	Ensemble learning of SVM, RF, DKDE	7.5	10.4
[89]	UCLA Campus (historical energy data)	K-NN (K=1)	—	15.3
[90]	UCLA Campus (historical energy data)	PSF	—	14.1
[14]	**ACN** (historical charging data)	GMMs	12.3	12.7
[91]	**ACN** (historical charging data) with weather and campus events	SVM,RF,XGBoost,ANN, Ensemble Learning	12.7	11.0
[81]	**ACN** (historical charging data)	SVM,RF,XGBoost,ANN, Ensemble Learning	9.92	11.6
Ours	**ACN** (historical charging data)	Ensemble learning of mode, MLR, DKDE with correlations	**9.32**	**9.59**

Table 3.5: Performance comparison for stay duration and energy consumption predictions, in terms of SMAPE, with previous works.

Therefore, when maintaining the comparison across the same dataset, it is evident that the incorporation of correlations along with an ensemble machine learning strategy has resulted in an improvement in the EV charging behavior predictions. Specifically, our work has achieved reductions in SMAPE of **24.2**% for stay duration and **24.5**% for energy consumption predictions, respectively.

4 EV Charging Scheduling

4.1 ACN-Simulator

To evaluate the impact of our predictions for EV charging parameters and the implementation of smart charging algorithms on achieving our objectives – reducing peak electricity demand and lowering charging costs for EV owners – comprehensive simulations based on realistic models are essential. This necessity arises from the challenges of translating these algorithms from theoretical constructs to practical applications. Practical systems introduce complexities often absent in simplified theoretical models. While these simpler models facilitate analysis, they can create significant discrepancies between theoretical predictions and the actual performance of implemented algorithms, affecting their robustness and effectiveness.

To this end, ACN-Sim [80], a data-driven, open-source simulation environment designed to evaluate online and closed-loop control strategies, will be employed. Despite the availability of numerous open-source tools capable of supporting smart grid research — such as MATPOWER [92], PandaPower [93], and PowerModels.jl [94], which facilitate solving optimal power flow problems in MATLAB, Python, and Julia respectively, along with other significant simulators like OpenDSS [95] and GridLab-D [96] that enable large-scale studies of the distribution system — ACN-Sim is particularly well-suited for our purposes. ACN-Sim integrates with many of these tools, including MATPOWER, PandaPower, and OpenDSS, enabling detailed studies of the effects of EV charging on the distribution grid. Moreover, its realistic models, derived from actual charging system data, unique unbalanced infrastructure modeling capabilities, and simple interfaces for defining new control algorithms set it apart from other simulators. At the same time, ACN-Sim includes functions to generate network models that match the physical infrastructure of the JPL site from which we obtain our charging data, thereby simplifying the linkage between the simulation environment and real-world data, a task that would otherwise be challenging.

4.2 Online Charging Algorithms

4.2.1 Uncontrolled Charging and Round Robin Algorithm

Uncontrolled Charging leads each EV to charge at its maximum allowable rate without considering infrastructure constraints. This approach often leads to significant challenges, such as grid overload during peak energy usage, which can trigger power outages and necessitate costly infrastructure upgrades. It also encourages inefficient energy use, as it disregards opportunities to utilize cheaper, sustainable energy sources available during off-peak hours. Most charging systems currently do not manage charging.

The Round Robin (RR) algorithm is a straightforward method designed to distribute charging capacity equally among all active EVs. In this context, 'active' refers to EVs that are currently plugged in, have a non-zero remaining energy demand, and are not scheduled to depart. The algorithm organizes these EVs into a queue and processes each one sequentially. For each EV, it assesses whether it is feasible to increase the charging rate by one unit. If an increase is feasible, the charging rate is incremented, and the EV is moved to the end of the queue. If it is not, the charging rate remains unchanged, and the EV is not returned to the queue. This process is repeated until the queue is empty.

4.2.2 Sorting Based Algorithms

Sorting based algorithms are commonly used in deadline scheduling tasks such as job scheduling in servers due to their simplicity [97]. Despite the wide array of sorting based algorithms available, each defined by specific metrics, our research specifically focuses on three: First-Come First-Served (FCFS), Earliest-Deadline First (EDF), and Least-Laxity First (LLF).

In the FCFS algorithm, EVs are processed in the order in which they arrive, without regard to their priority or deadlines. This approach is straightforward to implement and is fair in the sense that it strictly adheres to the order of requests, ensuring that no EV is overlooked.

On the other hand, in the EDF algorithm, EVs are prioritized based on their deadlines, with the EV closest to its deadline processed first. This priority is recalculated whenever a new EV is added to the queue or an EV's charging is completed, allowing the system to dynamically adapt to the charging workload.

The LLF algorithm prioritizes vehicles based on the driver's laxity, defined as the slack time remaining before the charging deadline becomes critical. Even though there are different notions of driver laxity, we use the following definition that has been applied to EV charging in [84]. Here, the initial laxity of an EV charging session i is defined as

$$LAX(i) = d_i - \frac{e_i}{r_i} \tag{37}$$

where d_i is the session duration, e_i is the energy demand and r_i is the charging rate for EV i. $LAX(i) = 0$ means that the EV must be charged at its maximum charging rate over the entire session duration to meet its energy demand. A higher value of $LAX(i)$ means there is more flexibility in satisfying its energy demand. In applying the LLF algorithm, EVs are dynamically prioritized and queued by their laxity, with those having the least laxity (least slack time) receiving priority in charging. This ensures that vehicles with the most urgent energy needs are charged promptly, enhancing the reliability of the charging station.

Overall, for each of the three sorting based algorithms, the active EVs are initially sorted based on the respective metric specific to each algorithm. After sorting, the EVs are processed in order, with each EV assigned its maximum feasible charging rate, given that the assignments to all previous EVs are fixed. This process continues sequentially until all EVs have been processed.

4.2.3 Model Predictive Control

Many approaches to the EV scheduling problem rely on model predictive control (MPC). The adacharge package [98] is based on CVXPY [99] [100], an open source Python-embedded modeling language for convex optimization problems, and makes it easy to use algorithms that aim to optimize EV charging with ACN-Sim. As such, our Adaptive Charging Algorithm

(ACA) uses MPC as described in Algorithm 4.1.

Algorithm 4.1 Adaptive Charging Algorithm (ACA)

1: **for** $k \in \mathcal{K}$ **do**
2: $\mathcal{V}_k := \{i \in \hat{\mathcal{V}}_k \mid e_i(k) > 0$ **AND** $d_i(k) > 0\}$
3: **if** *event fired* **OR** *time since last computation* $> P$ **then**
4: $(r_i^*(1), \ldots, r_i^*(T), i \in \mathcal{V}_k) := \mathbf{OPT}(\mathcal{V}_k, \mathcal{U}, \mathcal{R})$
5: $r_i(k+t) := r_i^*(1+t), \quad t = 0, \ldots, T-1$
6: **end if**
7: set the pilot signal of EV i to $r_i(k), \forall i \in \mathcal{V}_k$
8: $e_i(k+1) := e_i(k) - \hat{e}_i(k), \forall i \in \mathcal{V}_k$
9: $d_i(k+1) := d_i(k) - 1, \forall i \in \mathcal{V}_k$
10: **end for**

In our simulations, a discrete time model is utilized, with each time interval, indexed by $k \in \mathcal{K} := \{1, 2, 3, \ldots\}$, lasting δ minutes ($\delta = 5$ in our simulation). At time k, $\hat{\mathcal{V}}_k$ is the set of all EVs present at the JPL station and $\mathcal{V}_k \subseteq \hat{\mathcal{V}}_k$ is the subset of active EVs. The state of EV $i \in \mathcal{V}_k$ at time k is described by a tuple $(e_i(k), d_i(k), \bar{r}(k))$, where $e_i(k)$ is the remaining energy demand of the EV at the beginning of the period, $d_i(k)$ is the remaining duration of the session, and $\bar{r}(k)$ is the maximum charging rate for EV i. Moreover, $\hat{e}_i(k)$ is defined to be the measured energy delivered to EV i over time interval k. For simplicity, we express $r_i(t)$ in amps and $e_i(t)$ and $\hat{e}_i(t)$ in $\delta\times$ amps assuming nominal voltage.

We now describe the ACA algorithm. In line 2, the active EV set $\mathcal{V}_k$ is first computed by looking for all EVs currently plugged in which have non-zero remaining energy demand and are not already scheduled to depart. We then check, in line 3, if we should compute a new optimal schedule. This is done whenever an event-fired flag is `True`, or when the time since the last computed schedule exceeds P periods ($P = 3$ in our implementation).

If a new schedule is required, the optimal scheduling algorithm $\mathbf{OPT}(\mathcal{V}_k, \mathcal{U}, \mathcal{R})$ is called in line 4 that takes the form

$$\begin{aligned} \min_{\hat{r}} \quad & \mathcal{U}(\hat{r}) \\ \text{s.t.} \quad & \hat{r} \in \mathcal{R} \end{aligned} \tag{38}$$

where $\mathcal{U}$ is the objective function and $\mathcal{R}$ encodes a set of power and infrastructure constraints.

In our study, our objective is twofold: to minimize the peak charging load and reduce the charging costs for EV owners. Consequently, the objective function which we aim to

minimize is a weighted sum of two distinct utility functions, $u^v(r)$, and can be expressed as

$$U(r) := \alpha^{CM}\ u^{CM}(r) + \alpha^{LF}\ u^{LF}(r) \tag{39}$$

In Equation 39, $u^{CM}(r)$ can be used to simultaneously minimize costs for EV owners and maximize profits for system operators. It is defined as

$$u^{CM}(r) := \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{V}} r_i(t)(c(t) - \pi) \tag{40}$$

where $c(t)$ is the time-varying cost of one unit of energy, π is the per unit revenue from charging and $r_i(t)$ is the charging rate of EV i at time index t for $i \in \mathcal{V} := \{1, ..., V\}$ and $t \in \mathcal{T} := \{1, ..., T\}$. Setting π to a very large value (greater than the maximum marginal cost of energy) makes the utility function act as a non-completion penalty. This results in the algorithm attempting to minimize costs while meeting all energy demands.

On the other hand, $u^{LF}(r)$ is designed to minimize the peak load, which is equivalent to load variance minimization, a strategy known as load profile flattening. As such, $u^{LF}(r)$ can be formulated as

$$u^{LF}(r) := \sum_{t\in\mathcal{T}} \left(D(t) + \sum_{i\in V} r_i(t) \right)^2 \tag{41}$$

where $D(t)$ denotes the base load in slot t. In our implementation, both α^{CM} and α^{LF} are set to 1, as the two objectives are weighted equally, and π is set to 1000.

Moreover, the set of power and infrastructure constraints takes the form

$$0 \le r_i(t) \le \bar{r}_i(t) \qquad t \le d_i,\ i \in \mathcal{V} \tag{42a}$$

$$r_i(t) = 0 \qquad t > d_i,\ i \in \mathcal{V} \tag{42b}$$

$$\sum_{t\in\mathcal{T}} r_i(t) \le e_i \qquad i \in \mathcal{V} \tag{42c}$$

$$\left| \sum_{i\in\mathcal{V}} A_{li} r_i(t) e^{j\phi_i} \right| \le c_{lt}(t) \qquad t \in \mathcal{T},\ l \in \mathcal{L} \tag{42d}$$

Constraint (42a) ensures that the charging rate for all EVs at each time period is non-

negative, as vehicle-to-grid (V2G) capabilities [101] are not considered in this study, and does not exceed the upper limit set by each EV's battery management system (BMS). Constraint (42b) guarantees that no EV is charged beyond its departure time, while constraint (42c) limits the total energy delivered to each EV to at most e_i. To ensure that the optimization algorithm always returns a feasible solution – a critical requirement for practical applications – constraint (42c) does not require equality, acknowledging that the zero vector is always a feasible solution. Finally, constraint (42d) addresses the limits set by the unbalanced three-phase electrical network, as detailed below. It is important to note that a three-phase infrastructure, rather than the single-phase typically discussed in the literature, is modelled. This allows our simulation to more accurately reflect the behavior and complexities of real-world power distribution networks, thereby ensuring that our results are more closely aligned with realistic and practical scenarios.

For the infrastructure constraint (42d), we consider that the electric vehicle supply equipment (EVSEs) are connected line-to-line, as shown in Figure 4.1, given that this is the arrangement in the JPL station and most large charging systems in the United States.

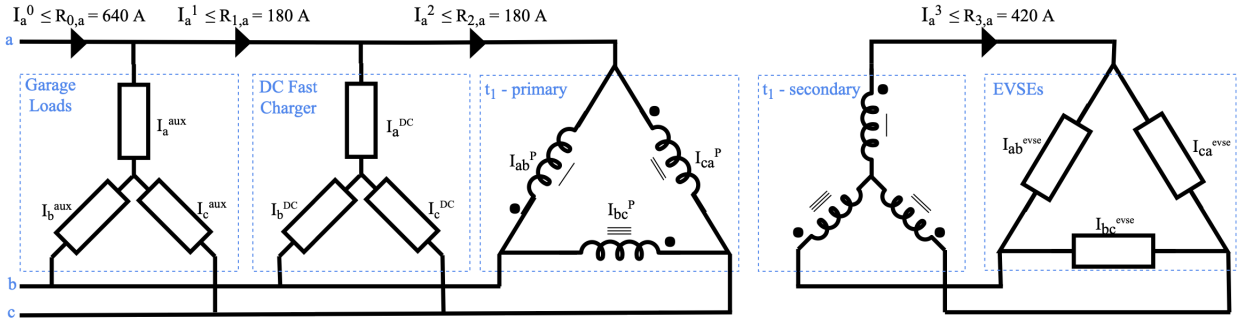


Figure 4.1: Circuit diagram depicting the connection loads within the JPL Parking Station. For simplicity, transformer t_2 is omitted and all EVSEs between phases A and B have been lumped together as I_{ab}^{evse}, and so forth for BC and CA.

In general, infrastructure constraints can be expressed as upper bounds on the magnitudes of currents within the system. By Kirchhoff's Current Law, we can express any current within the system as the sum of load currents. Let $\mathcal{V}(t)$ denote the set of EVs which are available to be charged at time t. The current draw of EV i at time t can be expressed as a phasor in the form $r_i(t)e^{j\phi_i}$, where $r_i(t)$ is the charging rate of the EV in amps, and ϕ_i is the phase

angle of the current sinusoid. We assume in this model that each charging EV has a unity power factor, so ϕ_i is determined based on how the EVSE is connected and the voltage phase angles (which we assume are separated by $\pm 120°$). [1] We can then model constraints within the electrical system as

$$\left| \sum_{i \in \mathcal{V}} A_{li} r_i(t) e^{j\phi_i} + L_l(t) \right| \leq c_{lt}, \quad t \in \mathcal{T}, \quad l \in \mathcal{L} \tag{43}$$

Infrastructure limits of the network are indexed by resources $l \in \mathcal{L}$. For example l may refer to a transformer or a breaker on a phase. For each constraint l, c_{lt} is a given capacity limit for time t and $L_l(t)$ is the aggregate current draw from uncontrollable loads through resource l. The matrix $\mathbf{A} = (A_{li}) \in \mathbb{R}^{|\mathcal{L}| \times |\mathcal{V}|}$ maps individual EVSE currents to aggregate currents within the network. This matrix can account for both the connection of loads and lines as well as the effect of transformers, such as the delta-wye transformers in the JPL station. The constraints in Equation 43 are second-order cone constraints, which are convex and can be handled by many off-the-shelf solvers such as ECOS, MOSEK, and Gurobi. However, in some applications, these constraints could be too computationally expensive or difficult to analyze. Hence, simpler, but more conservative constraints can be derived and employed to solve our optimization problem by observing

$$\left| \sum_{i \in \mathcal{V}} A_{li} r_i(t) e^{j\phi_i} \right| \leq \sum_{i \in \mathcal{V}} |A_{li}| r_i(t) \tag{44}$$

This yields conservative affine constraints in the form:

$$\sum_{i \in \mathcal{V}} |A_{li}| r_i(t) + |L_l(t)| \leq c_{lt}, \quad t \in \mathcal{T}, l \in \mathcal{L} \tag{45}$$

Using convex optimization through the CVXPY package, the optimization problem defined in Equation 38 aims to find the optimal solution for the charging rate $\hat{r} := (\hat{r}_i(1), \ldots, \hat{r}_i(T), i \in \mathcal{V}_k)$, for every active EV $i \in \mathcal{V}_k$ over the optimization horizon $\mathcal{T} := \{1, \ldots, T\}$. In words,

[1] This is reasonable since EVs' onboard charger generally includes power factor correction and voltage phase angles can be easily measured.

the algorithm aims to determine the optimal charging rate for each active EV i at every time slot t. It is important to note that **OPT** does not have a notion of the current time index k and returns an optimal solution $r_i^* := (r_i^*(1), \ldots, r_i^*(T))$ for Equation 38 as a T-dimensional vector for each active EV i. The algorithm then adjusts the indexing and sets the scheduled charging rates of EV i at time index k as $r_i(k+t) := r_i^*(1+t)$, for $t = 0, \ldots, T-1$ in line 5. At every time index k, regardless of whether a new schedule was produced, the pilot signal of each EV i is set to $r_i(k)$ (line 7) and the system state is updated for the next time period (lines 8,9).

4.3 Simulation Experiments

To assess the performance of the various online charging algorithms mentioned in Section 4.2, specifically their effectiveness in minimizing peak load demand and charging costs to EV owners, several steps must be considered within the ACN-Sim framework.

4.3.1 Define Experimental Parameters

First, several global parameters must be defined to establish the simulation environment. These include the site name, start and end dates, the duration of each discrete time interval, the network voltage, and the maximum charging rate for each EV battery. For our simulations, the site was set to the JPL station, reflecting the source of our charging data. The network voltage was set at 208V, and the maximum charging rate for each EV battery was capped at 6.6 kW. Furthermore, to assess the performance of different algorithms under varied conditions, we conducted multiple 24-hour simulations on specific days selected from our testing dataset collected in December 2018, adjusting the start and end dates accordingly.

4.3.2 Define EV and Battery Objects

In order to accurately represent the vehicles and their requirements for the simulation, the next step involves defining an EV object along with its corresponding battery characteristics. The `EV` object contains relevant information for a single charging session, including arrival

time, departure time, station ID, estimated departure time, and requested energy. In our simulations, the estimated departure time and requested energy are derived either from direct user inputs or predicted by our ensemble algorithm, as described in Section 3. By doing this, we assess how predicting these parameters – shown to be more accurate and reliable than direct user inputs in Section 3.3 – affects our overall objectives of minimizing peak charging demand and charging costs.

For each EV, a corresponding `Battery` object is defined, which includes characteristics such as the battery's capacity and initial charge. For our simulations, we set the initial charge to 0 Coulombs for all EVs to model the worst-case scenario. Moreover, an ideal battery model is utilized, where EVs are assumed to follow the given pilot signal exactly. In this idealized model, the actual charging rate of the battery, $\hat{r}(t)$, is described by

$$\hat{r}(t) := \min\{r(t), \bar{r}, \hat{e}(t)\} \tag{46}$$

where $r(t)$ is the pilot signal passed to the battery, $\bar{r}$ is the maximum charging rate of the on-board charger, and $\hat{e}(t)$ is the difference between the capacity of the battery and the energy stored in it at time t in the units of A $\times$ periods. We do not consider discharging batteries, so all charging rates are positive.

4.3.3 Create Simulation Environment

ACN-Sim uses events to describe actions in the simulation. There are two types of events currently supported:

- `PluginEvent` signals when a new EV arrives at the system. A `PluginEvent` also contains a reference to the EV object and its corresponding battery object, detailed in Section 4.3.2, which represents the new session.
- `UnplugEvent` signals when an EV leaves the system at the end of its charging session.

Each event has a timestamp describing when the event should occur. Events are stored in a queue (which we define as `EventQueue`) sorted by their timestamp. Since multiple events

could occur at the same timestep, we further sort by event type, first executing `UnplugEvents`, then `PluginEvents`. At each timestep, the `Simulator` executes all events left in the queue with timestamps on or before the current timestep. After any event, the charging scheduling algorithm is called to adapt to the new system state.

The `Simulator` object forms the base of any ACN-Sim simulation and defines a discrete-time, event-based simulation model. As such, this `Simulator` holds models of the hardware components in the simulated environment and a queue of events that define when actions occur in the system. Figure 4.2 illustrates the operation of the `Simulator`. During a simulation, the `Simulator` stores essential data such as event history, EV history, and time series for the pilot signal and charging current for each EVSE, enabling comprehensive analysis. This setup allows us to not only compare the effectiveness of the various scheduling algorithms defined in Section 4.2, but also to assess the impact of using predicted versus direct user inputs on achieving our objectives.

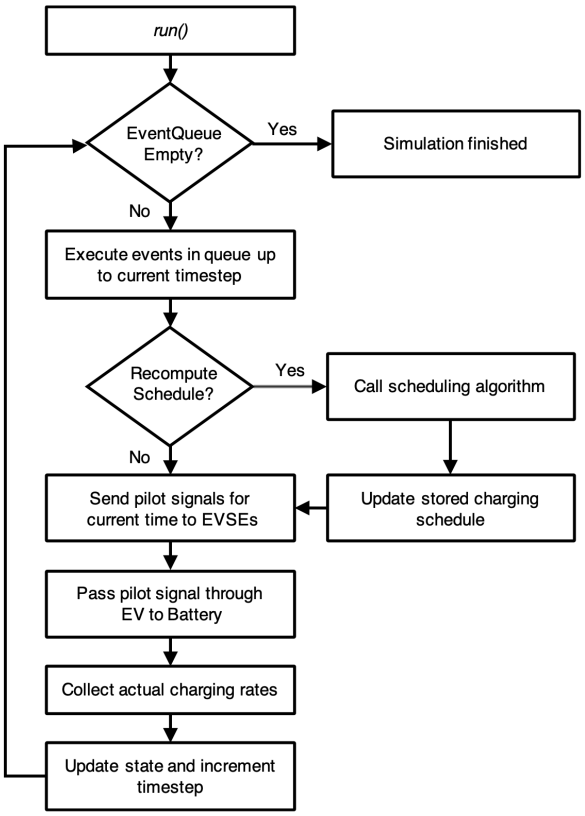


Figure 4.2: Flow chart describing the simulator's `run()` function. Each timestep consists of a single iteration of this loop. The simulation ends when the last event from the `EventQueue` is executed at which time the user can analyze the results of the simulation

4.4 Experimental Results

We aim to conclude two different outcomes from our experimental results. First, we assess whether different charging scheduling algorithms, particularly our ACA algorithm based on MPC with our explicit goals in its objective function, are capable of reducing peak load and charging costs for EV owners. Second, we evaluate whether the predicted inputs for stay duration and energy consumption from our ensemble machine learning algorithm, which were shown in Section 3.3 to be more accurate and robust in estimating EV user behavior compared to direct user inputs, are capable of further enhancing these reductions.

To obtain a representative sample of results, a one-week simulation was performed using the test dataset from December 10, 2018, to December 14, 2018. This week was selected because it had the highest average daily flow of cars compared to other weeks in the testing dataset, which spans the entirety of December 2018. It is important to note that weekends were not included in the simulation, as this is a workplace parking scenario with no cars parked on Saturdays or Sundays.

The peak loads for each of the five scheduling algorithms, listed in Section 4.2, as well as for uncontrolled charging, are summarized in Table 4.1. For each method, the peak load is presented for both the predicted inputs and the direct user inputs, with the latter values enclosed in brackets.

	Mon., 10/12	**Tue.,11/12**	**Wed., 12/12**	**Thu., 13/12**	**Fri., 14/12**
Uncontrolled	46.20 (46.20)	33.00 (33.00)	46.03 (46.03)	39.60 (39.60)	46.20 (46.20)
RR	43.76 (46.20)	33.00 (33.00)	46.03 (46.03)	37.09 (39.60)	46.20 (46.20)
FCFS	43.77 (46.20)	33.00 (33.00)	46.03 (46.03)	37.10 (39.60)	46.20 (46.20)
EDF	43.77 (46.20)	33.00 (33.00)	46.03 (46.03)	37.10 (39.60)	46.20 (46.20)
LLF	43.77 (46.20)	33.00 (33.00)	46.03 (46.03)	37.10 (39.60)	46.20 (46.20)
ACA	**8.82** (18.52)	**9.37** (19.17)	**8.53** (15.89)	**13.78** (14.34)	**9.50** (25.34)

Table 4.1: Peak demand loads in kW of uncontrolled charging and scheduling algorithms, with predicted and direct user inputs. The peak loads for direct user inputs are enclosed in brackets.

For direct user inputs, we observe that both the RR algorithm and all sorting based algorithms fail to reduce peak load compared to uncontrolled charging. This is due to users overestimating their energy requirements, with many declaring a demand larger than their

battery capacity, and wanting their cars charged as quickly as possible, thereby reducing the effectiveness of sorting-based algorithms. On the other hand, when using predicted inputs, these algorithms can reduce peak demand for two out of the five days, demonstrating some effectiveness of using predicted inputs over direct user inputs. However, since both RR and sorting algorithms do not optimize charging specifically to minimize peak load but rather follow their individual objectives, they are not highly effective in this task.

Conversely, the ACA algorithm manages to reduce peak load demand significantly relative to uncontrolled charging due to its explicit objective function, which incorporates a utility function for minimizing peak load as detailed in Section 4.2.3. For direct user inputs, the percentage decrease in peak load ranged from 41.9% on Tuesday, December 11, to 65.5% on Wednesday, December 12. This signifies a significant decrease in peak load demand, demonstrating the effectiveness of our ACA algorithm in achieving our objective.

At the same time, predicted inputs further reduced peak load demand for all five days. The smallest reduction occurred on Wednesday, where using predicted inputs for stay duration and energy consumption led to an additional 3.9% decrease in peak load, while the largest reduction occurred on Friday, with a further 62.5% decrease. As a result, using our ACA algorithm with predicted stay duration and energy consumption inputs led to overall decreases in peak load demand ranging from 65.2% to 81.5%, with an average decrease of **76.3**% when summing the loads across the entire week.

The total charging profiles of the JPL station for each day, under both uncontrolled charging and optimized charging conditions using the ACA algorithm, for both direct user inputs and predicted inputs, are depicted in Figure 4.3. By observing the load profiles, we see that for uncontrolled charging, where the EV starts charging immediately upon being plugged in, there is a significant surge in demand around 8 am, which remains high for approximately 2-5 hours. This pattern aligns with our expectations, given that most workers arrive at the office between 7-9 am, and the EVs tested require approximately 1-4 hours to fully charge from an empty battery. This is because, the JPL station is equipped with Level 2 chargers delivering power between 6 kW to 10 kW [102, 103], while the capacities of the

cars ranged from 7 kWh to 38 kWh, with the majority having a capacity between 15-20 kWh. Consequently, the load profile is very low or even zero for much of the day, with some small demand observed later at night on certain days.

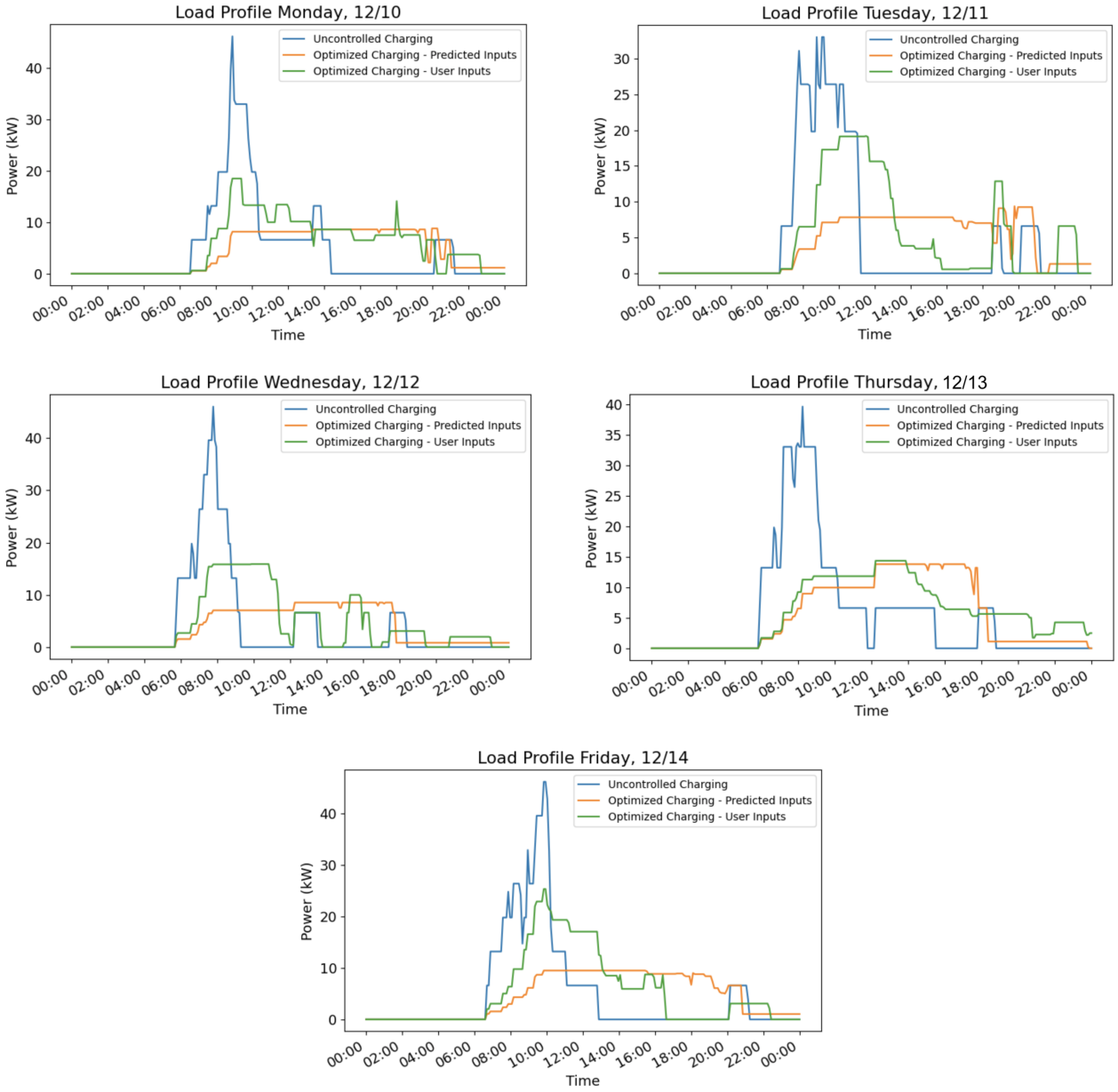


Figure 4.3: Daily charging load profile of the JPL station for uncontrolled and optimized charging using both direct user inputs and predicted inputs. The simulation week spans from Monday, December 10, 2018, to Friday, December 14, 2018.

In contrast, our controlled charging algorithm distributes the demand more evenly between 8 am and 6 pm, when most workers are at the office. This allows the optimized charging algorithm to minimize peak demand while meeting all consumer needs. The differ-

ence between predicted and direct user inputs arises because EV users tend to input a faster departure time than they actually require, forcing the algorithm to provide a higher charging rate during the early working hours. This leads to a higher peak demand during those hours and lower demand later in the day, as the cars are already charged. This discrepancy is particularly noticeable on Tuesday, December 11, where between 9:00 and 13:00, the peak load demand with direct user inputs is significantly higher compared to predicted inputs. However, during the rest of the working hours from 13:00 to 18:00, the demand with direct user inputs drops to nearly zero and is significantly lower than the demand with predicted inputs, which maintains a constant level. This is because the predicted inputs, informed by historical data, recognize that most EVs need to be available by 6 pm, aligning with the typical departure time of many workers.

As a result, we observe that the optimized charging algorithm effectively flattens the total load profile, resulting in peak load shaving during high demand periods and valley filling during low demand periods. It is important to note that in all scenarios tested, for both uncontrolled charging and the ACA algorithm with predicted and direct user inputs, we are satisfying 100% of user demands, even in the worst-case scenario of starting from an empty tank and requiring a full charge, given that we have knowledge of each car's capacity and set the initial charge to zero. Moreover, using predicted inputs rather than direct user inputs leads to further significant decreases in peak loads. The algorithms can better flatten the load curve by relying on the ensemble machine learning strategy, which uses both the individual's historical data and the data of their most correlated peers, thereby avoiding unrealistic and exaggerated user demands.

The total daily cost of the simulation, calculated by summing the total charging cost of all EV owners who plug in their cars during that day, is shown in Table 4.2.

As expected, the total daily energy cost follows the same pattern as peak load demand, given that the ACA algorithm manages to reduce it with direct user inputs and even further with predicted inputs for all days. This reduction is primarily due to the decrease in demand charges, which are proportional to peak demand and can account for 30 to 70 percent of the

	Mon., 10/12	**Tue.,11/12**	**Wed., 12/12**	**Thu., 13/12**	**Fri., 14/12**
Uncontrolled	8.33	6.96	7.58	11.31	7.81
ACA	**6.58** (7.49)	**6.09** (6.40)	**6.20** (6.95)	**9.22** (10.58)	**6.89** (7.30)

Table 4.2: Total daily simulation energy cost ($) for uncontrolled charging and charging under the ACA algorithm, with predicted and direct user inputs. Costs for direct user inputs are enclosed in brackets (omitted for uncontrolled charging as they are the same in both settings).

total charges on an electricity bill [104]. The ACA algorithm with predicted inputs manages to reduce the daily energy cost by 11.8% on Friday to 21.0% on Monday. It is important to note that although the absolute reduction in energy cost appears small, this is due to the limited number of EVs coordinated, which ranged from 11 to 14 in this simulation, as we only selected users with more than 20 charging sessions in the training dataset. Therefore, with a larger number of EVs the reduction in energy costs could be significant.

5 Time Series Forecasting

As outlined in Section 2.2, time series forecasting models are primarily categorized into two groups: statistical methods and deep learning approaches. On the one hand, statistical methods do not necessitate large volumes of training data, are relatively quick to train, and consume minimal computational resources. However, these methods often come with stringent assumptions limiting their effectiveness in real-world applications. On the other hand, deep learning models are highly expressive and can thus achieve superior accuracy. However, they are prone to overfitting, require substantial computational power, and need significant amounts of data to train effectively.

In response to the unique and often contradictory challenges posed by each method, our research adopts a hybrid approach, integrating both statistical and deep learning methods in our time series forecasting model. To determine the most effective combination of these methods for our hybrid model and benchmark our model against the SOTA models in both categories, a number of models were implemented. The models implemented include:

- Statistical Methods
  - Autoregressive Integrated Moving Average (ARIMA)
  - SAMoSSA
- Deep Learning Approaches
  - Long Short-Term Memory (LSTM)
  - Prophet
  - Inverted Transformer (iTransformer)
  - DeepAR
  - Temporal Fusion Transformer (TFT)

5.1 Theoretical Background of Time Series Models

5.1.1 Autoregressive Integrated Moving Average (ARIMA)

The ARMA(p, q) model, first intoduced by Box and Jenkins in 1970 is a combination of the autoregressive AR(p) and moving average MA(q) models [105]. The ARIMA model, an extension of the ARMA framework, is particularly useful in scenarios where the data exhibit non-stationarity in terms of the mean (though not necessarily in variance or autocovariance). In such cases, an initial differencing step – part of the "integrated" aspect of the model – is applied one or more times to remove the non-stationarity of the mean function, effectively eliminating trends.

To difference the data, the difference between consecutive observations is computed. Mathematically, this is shown as

$$y'_t = y_t - y_{t-1} \tag{47}$$

Differencing removes the changes in the level of a time series, eliminating trend and seasonality and consequently stabilizing the mean of the time series. Sometimes it may be necessary to difference the data a second time to obtain a stationary time series, which is referred to as second-order differencing:

$$y^*_t = y'_t - y'_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2} \tag{48}$$

In order to find the optimal differencing order, d, the Augmented Dickey–Fuller (ADF) test is used [106]. The differenced data are then used for the estimation of an ARMA model.

The orders (p, q) of an ARMA process are typically determined using the partial autocorrelation function (PACF) and the autocorrelation function (ACF), respectively. The optimal p and q values are identified as the lags at which the PACF and ACF, respectively, first fall below a critical value, which is set based on the specified confidence interval. For instance, with a 95% confidence interval, this critical value is set at $\frac{1.96}{\sqrt{N}}$, where N is the sample size. This approach helps ensure that the model includes only significant terms, enhancing its

predictive power and interpretability.

After finding the optimal p, q values, the ARMA model can be fitted by least squares regression to estimate the parameters $\{\phi_0, \phi_1, \ldots, \phi_p, \alpha_1, \alpha_2, \ldots, \alpha_q, \sigma^2\}$, from observations $\{z_t\}_{t=1}^{T}$, which minimize the error term. Using these parameters, future values can be predicted based on the autoregressive terms (ϕ) and moving average terms (α) using past observations and error terms as shown in Equation 49

$$\hat{Z}_{T+h|\epsilon_{T+h-q},\ldots,\epsilon_{T+h-1}} = \hat{\phi}_0 + \sum_{i=1}^{p} \hat{\phi}_i Z_{T+h-i} + \sum_{i=1}^{q} \hat{\alpha}_i \epsilon_{T+h-i} \tag{49}$$

5.1.2 SAMoSSA

The SAMoSSA model [107] proposes a two-stage algorithm, where multivariate Singular Spectrum Analysis (mSSA) [108] is first applied to estimate the non-stationary components, despite the presence of a correlated stationary AR component, which is subsequently learned from the residual time series. The proposed algorithm provides two main functionalities and can be applied for both univariate and mutlivariate time series. The first is decomposing the observations $y_n(t)$ into estimates of the non-stationary and stationary components for $t \leq T$. The second is forecasting $y_n(t)$ for $t > T$, which involves learning a forecasting model for both $f_n(t)$ (learned from the non-stationary part) and $x_n(t)$ (learned from the stationary part).

In the univariate case, the observations $y(t)$, where $t \in [T]$, are transformed into the $L \times T/L$ page matrix [109], $\mathbf{Z}_y$, such that $1 \leq L \leq \sqrt{T}$ and T/L is an integer. The SVD of $\mathbf{Z}_y$ is computed as $\mathbf{Z}_y = \sum_{l=1}^{L} s_l \boldsymbol{u}_l \boldsymbol{v}_l^\top$ where $s_1 \geq s_2 \geq \cdots \geq s_L \geq 0$ denote the ordered singular values, and $\boldsymbol{u}_l \in \mathbb{R}^L$ and $\boldsymbol{v}_l \in \mathbb{R}^{T/L}$ denote the left and right singular vectors, respectively, for $l \in [L]$. The matrix $\hat{\mathbf{Z}}_f$ is obtained by retaining the top $\hat{k}$ singular components of $\mathbf{Z}_y$, achieved by applying Hard Singular Value Thresholding (HSVT) with threshold $\hat{k}$ such that $\hat{\mathbf{Z}}_f = \sum_{l=1}^{\hat{k}} s_l \boldsymbol{u}_l \boldsymbol{v}_l^\top$.

To forecast $y(t)$ for $t > T$, forecasts are produced for both $\hat{f}(t)$ and $\hat{x}(t)$ using linear

models. Specifically, a linear model $\hat{\beta}$ is learned for $\hat{f}(t)$, defined as:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^{L-1}} \sum_{m=1}^{T/L} \left(y(Lm) - \beta^\top \hat{F}_m \right)^2, \tag{50}$$

where $\hat{F}_m = [\hat{f}(L(m-1)+1), \ldots, \hat{f}(L \times m - 1)]$ for $m \in [T/L]$. The forecast for $\hat{f}(t)$ is then produced using $\hat{\beta}$ and the $L-1$ lagged observations, formulated as $\hat{f}(t) = \hat{\beta}^\top Y(t-1)$, where $Y(t-1) = [y(t-1), \ldots, y(t-L)]$.

For the stationary component $\hat{x}(t)$, the parameters for an autoregressive process AR(p) are first estimated. The p-dimensional vectors $\hat{X}(t)$ are defined as $\hat{X}(t) := [\hat{x}(t), \ldots, \hat{x}(t-p+1)]$. From these, the ordinary least squares (OLS) estimate $\hat{a}$ is calculated by:

$$\hat{a} = \arg\min_{\alpha \in \mathbb{R}^p} \sum_{t=p}^{T-1} \left(\hat{x}(t+1) - \alpha^\top \hat{X}(t) \right)^2. \tag{51}$$

Subsequently, $\hat{x}(t)$ is forecasted using $\hat{a}$ and the p lagged entries of $\tilde{x}(\cdot)$, where $\tilde{x}(t') = y(t') - \hat{f}(t')$ such that $\hat{x}(t) = \hat{a}^\top \tilde{X}(t-1)$ where $\tilde{X}(t-1) = [\tilde{x}(t-1), \ldots, \tilde{x}(t-p_1)]$. Finally, the complete forecast for $y(t)$ is produced by combining the two component forecasts resulting in $\hat{y}(t) = \hat{f}(t) + \hat{x}(t)$.

For the multivariate case, the same procedure is applied with the key difference that $\mathbf{Z}_y$ is now a stacked page matrix, formed by the column-wise concatenation of the N page matrices induced by the N individual time series. A graphical depiction of the algorithm is illustated in Figure 5.1.

5.1.3 Long Short-Term Memory (LSTM)

LSTMs were introduced to handle the vanishing gradient problem [110] of the traditional Reccurent Neural Networks (RNNs) [111]. The architecture of an LSTM is similar to an RNN, but the recurrent cells are replaced with LSTM cells [112] which are constructed to retain information for extended periods. Similar to the cells in standard RNNs, the LSTM cell recurs the previous output, but also, keeps track of an internal cell state vector, $\mathbf{c}_t \in \mathbb{R}^h$, serving as long-term memory, where h is the number of hidden LSTM units. The main idea

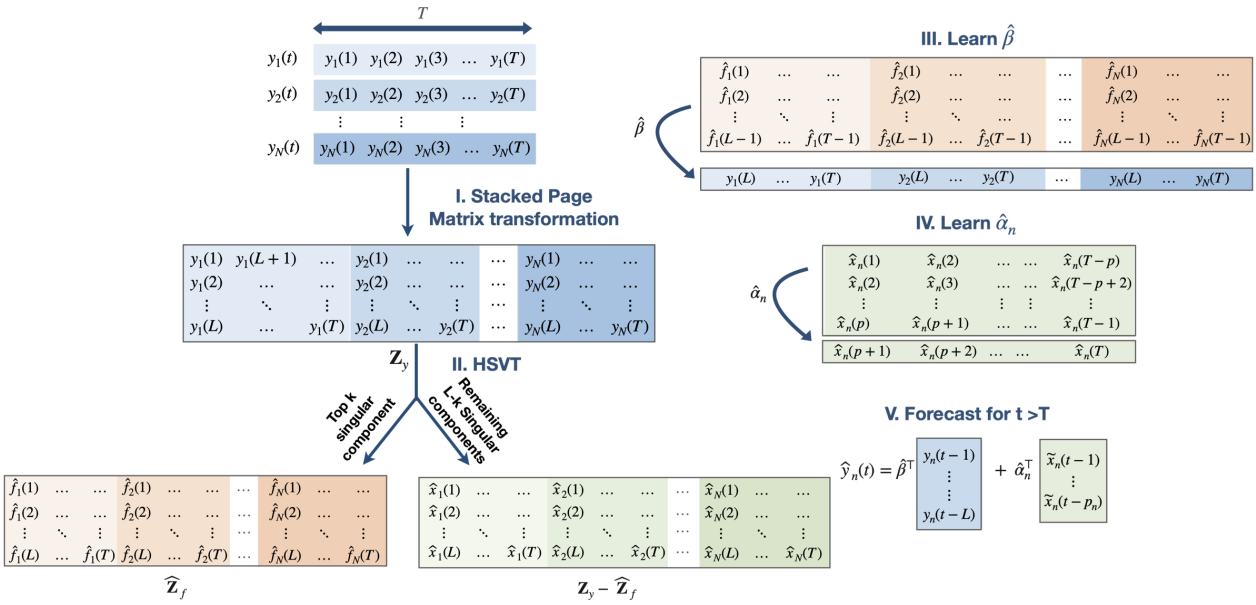


Figure 5.1: A visual depiction of the SAMoSSA algorithm. The five steps of the algorithm are:
(i) transform the time series into its stacked page matrix representation (ii) decompose the time series into non-stationary and stationary components (ii) estimate $\hat{\beta}$ (iv) estimate $\hat{\alpha}_n$ for all $n \in [N]$; (v) produce the forecast $\hat{y}_n(t)$ for $t > T$.

of the LSTM cell is to regulate the updates of the long-term memory (cell state), such that information and gradients (for training) can flow unchanged between iterations. In particular, to carefully regulate the cell-state, the LSTM employs three internal gates, namely the **forget** gate, the **input** gate, and the **output** gate as illustrated in Figure 5.2.

- **Forget** Gate: Decides what information and how much of it to erase from the cell state by using a sigmoid activation function, which ensures that the gate's output vector yields continuous values ranging from 0 (forget) to 1 (keep). The information in the cell state $\mathbf{c}_{t-1}$ is then altered through element-wise multiplication with the forget gate's output, $\mathbf{f}_t$. Mathematically, $\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$ and $\mathbf{c}_f = \mathbf{c}_{t-1} * \mathbf{f}_t$.

- **Input** Gate: Uses two functional units. The first functional unit uses a *tanh* activation function, which outputs values between -1 and 1 to determine the *change* in the cell state, denoted as $\tilde{\mathbf{c}}_t$. The second unit uses a sigmoid activation function responsible for the *magnitude* of the change, denotes as $\mathbf{m}_t$. After element-wise multiplication of the outputs from these two units, the input gate adds the result to the cell state, thus

completing the update of the cell state, $\mathbf{c}_t$. Mathematically, $\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$, $\mathbf{m}_t = \sigma(\mathbf{W}_m \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_m)$ and $\mathbf{c}_t = \mathbf{c}_f + \tilde{\mathbf{c}}_t * \mathbf{m}_t$

- **Output** Gate: Fetches relevant information from the current input and the previous output, and combines it with the newly adjusted cell state, $\mathbf{c}_t$, to predict the next output, $\mathbf{h}_t$. This output recurs to serve as input for the next iteration. If the model comprises multiple layers, this output is also utilized as input for the next layer; otherwise, it constitutes the final prediction. Mathematically, $\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$ and $\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{c}_t)$

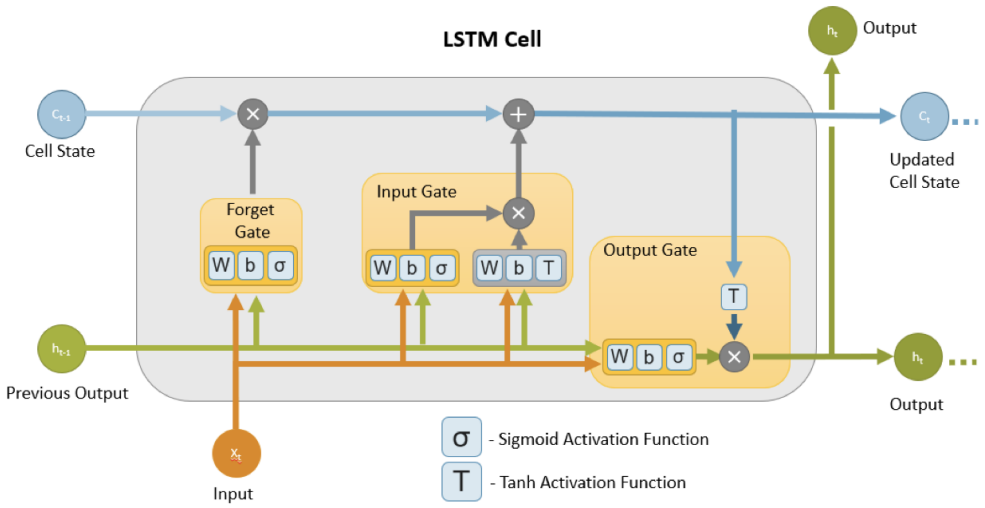


Figure 5.2: The architecture of an LSTM cell. The 'cross-sign' in the gates is a dot product, and the 'plus-sign' an addition. The current input and the previous output are concatenated before being fed into each gate.

It is important to note that the weight matrices $\mathbf{W} \in \mathbb{R}^{d \times h}$ and bias vectors $\mathbf{b} \in \mathbb{R}^h$, where d and h refer to the number of input features and the number of hidden units, respectively, constitute the trainable parameters of the LSTM model. The pseudocode detailing one iteration through an LSTM cell is provided in Algorithm 5.1.

5.1.4 Prophet

Prophet is an open-source tool provided by Facebook Inc. As explained by the authors in [113], the idea leading to Prophet was to develop a flexible forecasting tool which is easy to both use and tune. The underlying model features a decomposable time series with

Algorithm 5.1 One iteration through an LSTM cell

procedure ITERATION($\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t$)
  $\mathbf{c}_t \leftarrow$ Element-wise multiply $\mathbf{c}_{t-1}$ with output from forget gate $\mathbf{f}_t$
  $\mathbf{c}_t$ += The output from the input gate ($\tilde{\mathbf{c}}_t * \mathbf{m}_t$)
  $\mathbf{h}_t \leftarrow$ Combination of the cell state $\mathbf{c}_t$, input $\mathbf{x}_t$ and previous output $\mathbf{h}_{t-1}$
  return $(\mathbf{c}_t, \mathbf{h}_t)$
end procedure

three main model components: trend (or growth) $g(t)$, seasonality $s(t)$, and holidays $h(t)$ (if present). In our case, there are no obvious holidays to consider, as we aim to forecast the arrival times of EV owners which tend to be unaffected by public holidays. The time series is therefore decomposed as:

$$y(t) = g(t) + s(t) + \epsilon_t, \tag{52}$$

where ϵ_t encodes variations that are not taken into account by the model, and which are assumed to be normally distributed [113]. The Prophet model can be seen as a Generalized Additive Model (GAM) [114]. In this framework, forecasting is phrased as a curve-fitting task, with time as the only regressor, resulting in the model being suited for univariate time series only.

The trend function adopted for the problem under study is a piecewise linear function written as:

$$g(t) = k + \sum_{i:t>s_i} \delta_i(t - s_i) + m + \sum_{j:t>s_j} \gamma_j \tag{53}$$

where k is a scalar coefficient, s_i are the trend changepoints – i.e., S times $s_1, s_2, ..., s_S$ at which the angular coefficient of the trend is allowed to change – δ_i are the rate adjustments, and $\gamma_j = -s_j\delta_j$ are parameters used to make the function continuous. The algorithm starts with a number $S = 25$ of potential changepoints, placed in the first 80% of the time series in order to avoid responding to fluctuations in the last part of the series. Then, the actual changepoints are selected by putting a sparse prior of the kind $\delta_j \sim \text{Laplace}(0, \tau)$, with τ (a tunable hyperparameter) regulating the magnitude's rate adjustments. A larger τ means the model has more power to fit trend changes.

As for seasonality, Prophet accounts for it using Fourier series, namely:

$$s(t) = \sum_{n=1}^{N} a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \tag{54}$$

Given that the JPL station is a workplace parking facility with distinct charging variations between weekdays and weekends, weekly seasonality is most appropriate, and therefore, we set $P = 7$ days. Truncating the series at N applies a low-pass filter to the seasonality, so increasing N allows for fitting seasonal patterns that change more quickly, albeit with increased risk of overfitting. For weekly seasonality, the truncation parameter is set to $N = 3$ by the authors of [113] and we follow this specification. When performing the fit, a smoothing prior $\beta \sim N(0, \sigma^2)$ is imposed on the $2N$ components $\boldsymbol{\beta} = [a_1, ..., a_N, b_1, ..., b_N]^T$, with σ, a second hyperparameter, essentially acting as an L2-regularization parameter.

Prophet fits its GAM using the L-BFGS quasi-Newton optimization method of [115] in a Bayesian setting, finding a maximum a posteriori estimate.

5.1.5 Inverted Transformer (iTransformer)

The iTransformer [116] adopts the encoder-only architecture of the vanilla Transformer [117], including the embedding, projection, and Transformer blocks, focusing on representation learning and adaptive correlating of multivariate series. Each time series is firstly tokenized to describe the properties of the variate. It is then processed by self-attention for mutual interactions, and subsequently, each series is individually processed by feed-forward networks for generating series representations. Notably, the task of generating the predicted series is essentially handed over to linear layers, proven competent by previous work [118].

Based on the above considerations, in the iTransformer model, the process of predicting future series of each specific variate $\hat{\mathbf{Y}}_{:,n}$ based on the lookback series $\mathbf{X}_{:,n}$ is simply formulated as follows:

$$\begin{aligned} \mathbf{h}_n^0 &= \text{Embedding}(\mathbf{x}_{:,n}), \\ \mathbf{H}^{l+1} &= \text{TrmBlock}(\mathbf{H}^l), \quad l = 0, \ldots, L-1, \\ \hat{\mathbf{Y}}_{:,n} &= \text{Projection}(\mathbf{h}_n^L), \end{aligned} \tag{55}$$

where $\mathbf{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_N\} \in \mathbb{R}^{N \times D}$ contains N embedded tokens, each of dimension D, and the superscript denotes the layer index. The obtained variate tokens interact with each other by self-attention and are independently processed by the shared feed-forward network in each TrmBlock. Since each token is previously normalized on its feature dimension, the entries can somewhat reveal the variate-wise correlation, and the whole score map $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ exhibits the multivariate correlations between paired variate tokens. Consequently, highly correlated variates will be more weighted for the next representation interaction with values $\boldsymbol{V}$. Based on this intuition, the proposed mechanism is believed to be more natural and interpretable for multivariate series forecasting.

A visual depiction of the iTranformer model is illustrated in Figure 5.3, where it is observed that L blocks composed of the layer normalization, feed-forward network, and self-attention modules are stacked together to increase the expressive power and forecasting accuracy of the model.

The simple pseudo-code of the iTransformer model, clearly depicting the way the model is able to predict future values, is listed in Appendix B.

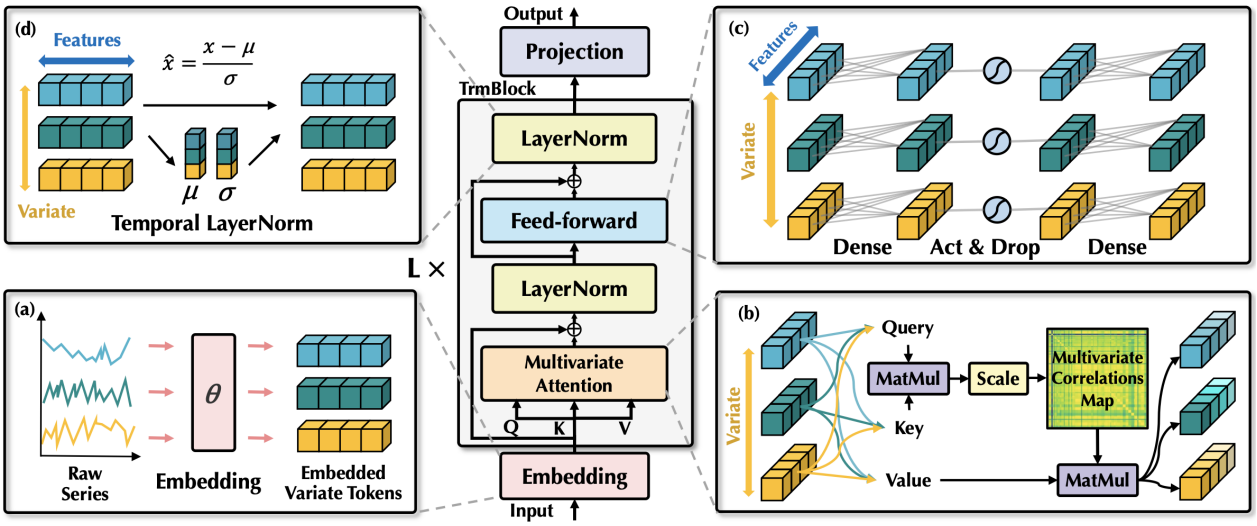


Figure 5.3: Overall structure of iTransformer, which shares the same modular arrangement with the encoder of Transformer. (a) Raw series of different variates are independently embedded as tokens. (b) Self-attention is applied to embedded variate tokens with enhanced interpretability revealing multivariate correlations. (c) Series representations of each token are extracted by the shared feed-forward network. (d) Layer normalization is adopted to reduce the discrepancies among variates.

5.1.6 DeepAR

DeepAR [119] is based on training an auto regressive recurrent network model [120, 121] on a large number of related time series, demonstrating how by applying deep learning techniques to forecasting, one can overcome many of the challenges faced by widely-used classical approaches to the problem.

Denoting the value of time series i at time t by $z_{i,t}$, the goal is to model the conditional distribution $P(\boldsymbol{z}_{i,t_0:T}|\boldsymbol{z}_{i,1:t_0-1}, \boldsymbol{x}_{i,1:T})$ of the future values of each time series $\boldsymbol{z}_{i,t_0:T} := [z_{i,t_0}, z_{i,t_0+1}, ..., z_{i,T}]$, given its past values $\boldsymbol{z}_{i,1:t_0-1} := [z_{i,1}, ..., z_{i,t_0-2}, z_{i,t_0-1}]$ and covariates $\boldsymbol{x}_{i,1:T}$. Here, t_0 denotes the time point from which we assume $\boldsymbol{z}_{i,t}$ to be unknown at prediction time, and $\boldsymbol{x}_{i,1:T}$ are covariates that are assumed to be known for all time points.

The model distribution $Q_\Theta(\boldsymbol{z}_{i,t_0:T}|\boldsymbol{z}_{i,1:t_0-1}, \boldsymbol{x}_{i,1:T})$ is assumed to consist of a product of likelihood factors:

$$Q_\Theta(\boldsymbol{z}_{i,t_0:T}|\boldsymbol{z}_{i,1:t_0-1}, \boldsymbol{x}_{i,1:T}) = \prod_{t=t_0}^{T} Q_\Theta(z_{i,t}|\boldsymbol{z}_{i,1:t-1}, x_{i,1:T}) = \prod_{t=t_0}^{T} l(z_{i,t}|\theta(\boldsymbol{h}_{i,t}, \Theta)) \tag{56}$$

parametrized by the output $\boldsymbol{h}_{i,t}$ of an autoregressive recurrent network:

$$\boldsymbol{h}_{i,t} = h(\boldsymbol{h}_{i,t-1}, z_{i,t-1}, \boldsymbol{x}_{i,t}, \Theta), \tag{57}$$

where h is a function implemented by a multi-layer recurrent neural network with LSTM cells and Θ denotes the model parameters. The likelihood $l(z_{i,t}|\theta(\boldsymbol{h}_{i,t}))$, directly predicts all parameters θ (e.g. the mean μ and variance σ^2) of the probability distribution for the next time point. In our wotk, the Gaussian likelihood

$$l_G(z|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) \tag{58}$$

is used, parameterized by $\theta = (\mu, \sigma)$. Here, $\mu(\boldsymbol{h}_{i,t}) = \boldsymbol{w}_\mu^\top \boldsymbol{h}_{i,t} + b_\mu$ and $\sigma(\boldsymbol{h}_{i,t}) = \log\left(1 + \exp(\boldsymbol{w}_\sigma^\top \boldsymbol{h}_{i,t} + b_\sigma)\right)$.

DeepAR, uses a sequence-to-sequence architecture, comprising of an encoder and a de-

coder that share a uniform architecture and weights. The encoder's state, $\boldsymbol{h}_{i,0}$, and the initial series value, $z_{i,0}$, are initialized to zero. On the other hand, the decoder's initial state, $\boldsymbol{h}_{i,t_0-1}$, is effectively set by the encoder's final output, which is computed based on all observed inputs up to $t_0 - 1$. Ancestral sampling is then applied to generate joint samples from the model distribution $Q_\Theta(\boldsymbol{z}_{i,t_0:T}|\boldsymbol{z}_{i,1:t_0-1}, \boldsymbol{x}_{i,1:T})$. Specifically, after obtaining $\boldsymbol{h}_{i,t_0-1}$, predictions for $z_{i,t}$ are sequentially sampled for $t = t_0$ to T using the likelihood function $l(\cdot|\theta(\tilde{\boldsymbol{h}}_{i,t}, \Theta))$, where $\tilde{\boldsymbol{h}}_{i,t} = h(\boldsymbol{h}_{i,t-1}, \tilde{z}_{i,t-1}, \boldsymbol{x}_{i,t}, \Theta)$ is initialized with $\tilde{\boldsymbol{h}}_{i,t_0-1} = \boldsymbol{h}_{i,t_0-1}$ and $\tilde{z}_{i,t_0-1} = z_{i,t_0-1}$.

A visual summary of the model in both the training and prediction stage is illustrated in Figure 5.4.

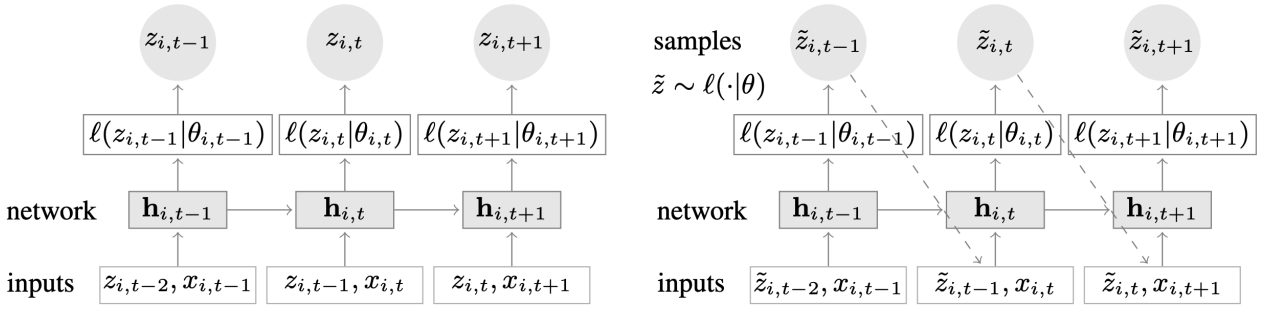


Figure 5.4: Summary of the model. Training (left): At each time step t, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, as well as the previous network output $\boldsymbol{h}_{i,t-1}$. The network output $\boldsymbol{h}_{i,t} = h(\boldsymbol{h}_{i,t-1}, z_{i,t-1}, \boldsymbol{x}_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(\boldsymbol{h}_{i,t}, \Theta)$ of the likelihood $\ell(z|\theta)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\tilde{z}_{i,t} \sim \ell(\cdot|\theta_{i,t})$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$ generating one sample trace. Repeating this prediction process yields many traces representing the joint predicted distribution.

5.1.7 Temporal Fusion Transformer (TFT)

The TFT, introduced by Lim *et al.* [122] in 2021, is an attention-based DNN architecture consisting of multiple aligned recurrent networks (LSTM encoder-decoder) and multi-head attention blocks. Unlike the black-box nature of LSTMs and Transformers, TFTs can be interpreted with the help of the variable selection networks, which identify relevant input features, and attention scores given by the model for past time steps.

The major constituents of the TFT are:

- **Gating mechanisms** to skip over any unused components of the architecture. This gives the model the flexibility to apply non-linear processing only when needed, recognizing that there may be instances that simple models can be beneficial, for example when datasets are small or noisy. Component gating layers based on Gated Linear Units (GLUs) [123] are used, to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset.

- **Variable selection networks** to automatically select the relevant input features with significant information about the target variable at each time step. Through these networks, TFTs can provide intrepretable insights into the underlying dynamics of a time series.

- **Static covariate encoders** to integrate static features into the network, through encoding of context vectors to condition temporal dynamics.

- **Temporal processing** to learn both long- and short-term temporal relationships from both observed and known time-varying inputs. A sequence-to-sequence layer is employed for local processing, whereas long-term dependencies are captured using an interpretable multi-head attention block, a modified version of the traditional multi-head attention used in transformer-based architectures [117, 124]. This modification involves sharing values across each head and employing additive aggregation to combine the outputs of all heads.

- **Prediction intervals** via quantile forecasts to determine the range of likely target values at each prediction horizon on top of point forecasts. This is achieved by the simultaneous predictions of various percentiles (e.g.$10^{th}, 50^{th}$ and 90^{th}) at each time step.

Figure 5.5 shows the high level architecture of the TFT.

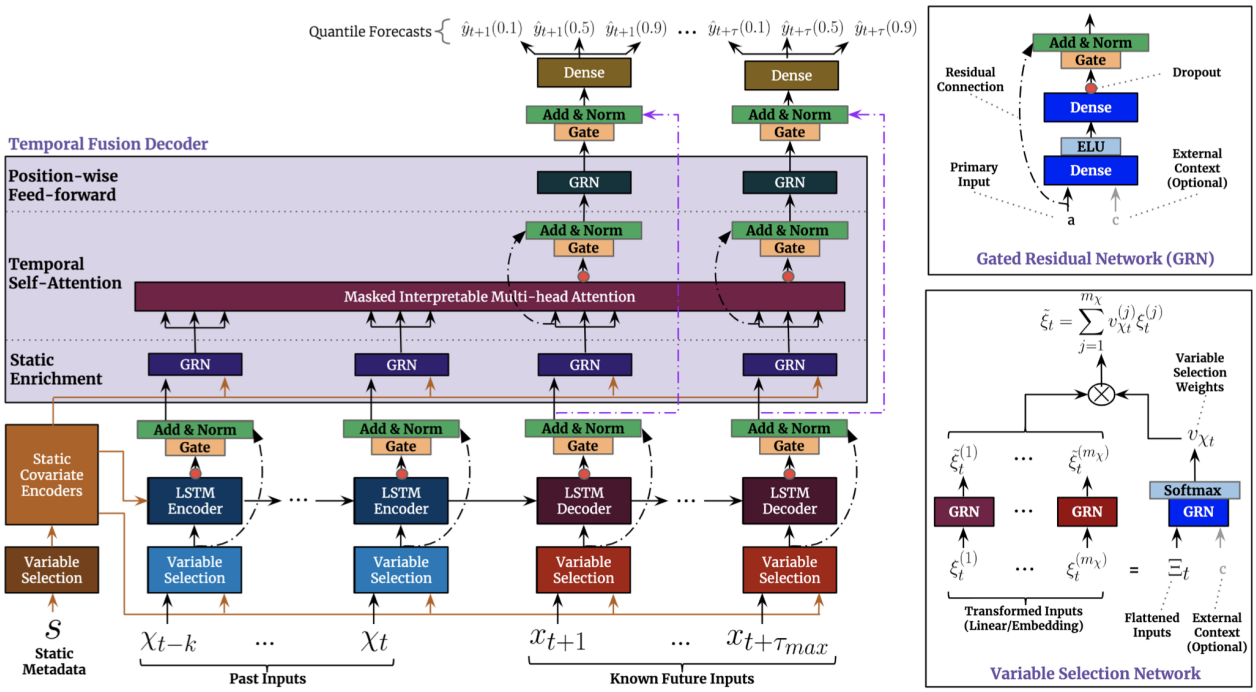


Figure 5.5: TFT architecture. TFT inputs static metadata, time-varying past inputs and time-varying a priori known future inputs. Variable Selection is used for judicious selection of the most salient features based on the input. Gated Residual Network blocks enable efficient information flow with skip connections and gating layers. Time-dependent processing is based on LSTMs for local processing, and multi-head attention for integrating information from any time step.

5.2 Noise Filtering Techniques

As we aim to predict the arrival times of EV vehicle owners, which often suffer from large data variability due to people's diverse schedules and unpredictable events, a further goal of our model is to make the predictions as robust as possible. To achieve this, we will first eliminate the noise present in the data. This process aims to extract the real signal from the noisy data, thereby increasing the signal-to-noise (SNR) ratio of the input signal to our

hybrid model. Two different noise filtering techniques will be implemented, namely Singular Spectrum Analysis (SSA) and Robust Principal Component Analsysis (RPCA). This choice was motivated by the extensive study of these methods in the literature, their non-parametric and highly explainable nature, and their varying strengths and limitations, allowing us to determine which is better suited for our specific task.

5.2.1 Singular Spectrum Analysis (SSA)

Singular Spectrum Analysis (SSA) [125] is a nonparametric spectral estimation method. The basic aim of SSA is to decompose the time series into the sum of interpretable components such as trend, periodic components and noise with no a-priori assumptions about the parametric form of these components. An example of SSA applied to a time series F is shown in Figure 5.6.

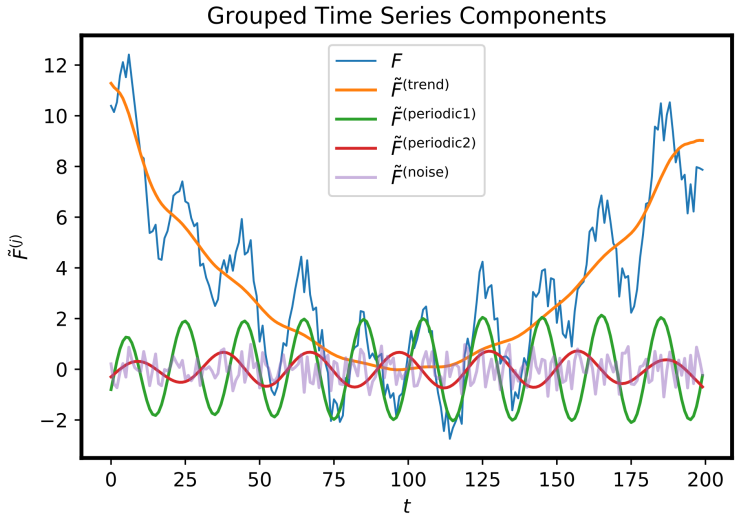


Figure 5.6: Singular spectrum analysis applied to a time series F, with reconstructed components grouped into trend, periodic components, and noise.

The areas where SSA can be applied are very broad and include climatology, marine science, geophysics, image processing and medicine among them. Hence, different modifications of SSA have been proposed and different methodologies of SSA are used in practical applications such as trend extraction, periodicity detection, seasonal adjustment, smoothing and noise reduction [125, 126]. For noise filtering applications, a discrete-time signal, $y(i)$, $i \in 1, \ldots, N$, is broken up into k time-shifted segments, and these segments are arranged as

rows in a Hankel matrix $\mathbf{Y} \in \mathbb{R}^{k \times j}$, $k < j$, $Y_{pq} = y(p+q-1)$.

$$\mathbf{Y} = \begin{bmatrix} y(1) & y(2) & \cdots & y(j-1) & y(j) \\ y(2) & y(3) & \cdots & y(j) & y(j+1) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ y(k) & y(k+1) & \cdots & y(k+j-2) & y(k+j-1) \end{bmatrix}$$

The matrix $\mathbf{Y}$ is a Hankel matrix because values along the antidiagonals are all equal to one another. It is known as the *trajectory* matrix of the time series.

The Singular Value Decomposition (SVD) of $\mathbf{Y}$ is given by $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, and a reduced-rank version of $\mathbf{Y}$, denoted as $\mathbf{Y}_n$, can be reconstructed from the first n dyads of the SVD:

$$\mathbf{Y}_n = \mathbf{U}_n \mathbf{\Sigma}_n \mathbf{V}_n^T$$

where $\mathbf{U}_n \in \mathbb{R}^{k \times n}$, $\mathbf{\Sigma}_n \in \mathbb{R}^{n \times n}$, and $\mathbf{V}_n \in \mathbb{R}^{j \times n}$ are the first n singular vectors of $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{\Sigma}_n$ contains the largest n singular values. This reduced-rank matrix $\mathbf{Y}_n$ is the rank-n matrix that is closest to $\mathbf{Y}$ in the sense of minimizing the Frobenius norm of their difference, $\|\mathbf{Y} - \mathbf{Y}_n\|_F^2$.

In general, $\mathbf{Y}_n$ will not retain the same Hankel structure as $\mathbf{Y}$. Thus, SSA is used to obtain a matrix with Hankel structure, by replacing elements along each anti-diagonal with the average of the anti-diagonal. The resulting matrix $\overline{\mathbf{Y}}_n$ will not have rank-n and will not be the closest matrix to $\mathbf{Y}$ in the sense of Frobenius norm, but it will have a Hankel structure.

The low-rank (filtered signal) can be recovered from the first row and last column of $\overline{\mathbf{Y}}_n$. It is important to note that selecting the appropriate value of n in the SVD is crucial for obtaining a filtered signal. Signal-to-noise separation can be effectively achieved by examining the slope break in a 'scree diagram', which plots the eigenvalues λ_n of $\mathbf{Y}$ against n. The point at which the slope significantly changes is typically chosen as the value for n.

SSA directly supports time series data and does not assume any distribution about the noise component of the data. However, it is highly sensitive to outliers and to the choice of the window length. Additionally, the value of n should be carefully selected given that noise

variance should be smaller than the selected n relevant eigenvalues for the decomposition to work well [127, 128].

5.2.2 Robust Principal Component Analsysis (RPCA)

As with SSA, in our implementation of RPCA, the time series is first arranged into a Hankel matrix, $\mathbf{Y}$, which serves as the trajectory matrix to accommodate the time series nature of the data. In this matrix the time series is the first representation, whereas the lagged matrix is the second representation that reflects the geometry of the time series [129]. More specifically, the lagged matrix is able to represent shapes and patterns of the time series such as the magnitude of trends and the frequency of periodicities [129]. If the time series data is contaminated with noise, matrix $\mathbf{Y}$ tends to become a full-rank matrix. The basic idea of RPCA [130] is to convert the denoising problem as the optimization of the low-rank approximation. Specifically, RPCA decomposes the original matrix $\mathbf{Y}$ such that $\mathbf{Y} = \mathbf{L} + \mathbf{S}$. Here, $\mathbf{L}$ is a low-rank matrix that aims to approximate the clean data in the original data matrix $\mathbf{Y}$, and $\mathbf{S}$ is a sparse matrix that consists of element-wise outliers that should not be captured by the low-rank matrix $\mathbf{L}$. In other words, if the low rank matrix $\mathbf{L}$ tries to capture the outliers in $\mathbf{S}$ then $\mathbf{L}$ is skewed to the outliers and thus cannot capture appropriately the clean data.

RPCA achieves the decomposition by solving the optimization problem shown in Equation 59.

$$\arg\min_{\mathbf{L},\mathbf{S}} \operatorname{rank}(\mathbf{L}) + \lambda \|\mathbf{S}\|_0 \quad \text{s.t. } \mathbf{Y} = \mathbf{L} + \mathbf{S} \tag{59}$$

Here, $\operatorname{rank}(\mathbf{L})$ is the rank of matrix $\mathbf{L}$; $\|\mathbf{S}\|_0$ is the ℓ_0 norm of matrix $\mathbf{S}$, which counts the number of non-zero elements in $\mathbf{S}$; and λ is a trade-off parameter that controls the relative importance of $\|\mathbf{S}\|_0$. In addition, the optimization is constrained by $\mathbf{Y} = \mathbf{L} + \mathbf{S}$. Minimizing the loss function makes it possible to identify a low rank matrix $\mathbf{L}$ that approximates the original matrix $\mathbf{Y}$ and a sparse matrix $\mathbf{S}$ that includes outliers.

The objective function in Equation 59 is a non-convex optimization problem. Thus, it is

often converted to

$$\arg\min_{\mathbf{L},\mathbf{S}} \|\mathbf{L}\|_* + \beta \|\mathbf{S}\|_1 \quad \text{s.t. } \mathbf{Y} = \mathbf{L} + \mathbf{S} \tag{60}$$

where $\|\mathbf{L}\|_*$ is the nuclear norm of matrix $\mathbf{L}$ and $\|\mathbf{S}\|_1$ is the ℓ_1 norm of matrix $\mathbf{S}$ [131].

RPCA assumes that noise is sparse and recovers the clean data through optimizing the low-rank approximation. Moreover, due to the its 'robust' nature, RPCA is able to effectively identify and remove outliers, unlike SSA, which is very sensitive to outliers due to its reliance on SVD. However, RPCA does not inherently support time series data and is limited to only linear transformations [132].

5.3 Methodology

In order to obtain the benefits of both statistical methods and deep learning models, a hybrid model comprising both types of models will be developed. The goal of our hybrid model is twofold: firstly, minimizing the number of trainable parameters through the use of non-parametric statistical methods, thereby reducing the dependency on large training datasets; secondly, capturing the complex temporal dynamics and dependencies in data through the use of deep learning methods. This approach is essential in our application of predicting the arrival times of EV users, given that the training dataset utilized spans only two months, with the most training data for an individual user being 41 data points (and the least being 20). The choice of this training dataset is not arbitrary but is based on findings in [14], which demonstrated that a 60-day period is optimal for predicting stay duration and energy consumption, the two key charging parameters of interest, after testing various periods ranging from 30 to 120 days. This prompted us to use this dataset in predicting EV user behavior, detailed in Section 3, and consequently, the same dataset must be used for arrival time prediction.

5.3.1 Architecture of Our Hybrid Model

Our hybrid model employs a series architecture divided into three distinct stages, each serving a specific purpose, which enhances the model's interpretability, intuitiveness, and

explainability. In the first stage, the noisy input signal is denoised using either SSA or RPCA, as detailed in Section 5.2. This initial processing step aims to increase both the robustness and accuracy of the model. Subsequently, the denoised signal is separated into two components: the non-stationary part and the stationary part.

For the non-stationary part, we employ a similar statistical method to the one used in SAMoSSA, as detailed in Section 5.1.2. This method offers theoretical guarantees for the error in non-stationary prediction and features zero trainable parameters, thus reducing the complexity of our model in response to the small training dataset. Once the prediction for the non-stationary component is obtained using the linear model, it is subtracted from the denoised signal to isolate the stationary, residual component of the time series. For this stationary component, an LSTM network is utilized. This is because, LSTMs are adept at capturing non-linearities present in the residual [133] that were not addressed by the decomposition process and can adapt to changes in the pattern of the residuals over time, enhancing their robustness to changes in the underlying processes of the time series data. Moreover, they are capable of capturing long-term dependencies that are often present in the residual far more effectively than simpler linear or RNN models, while still maintaining a relatively small number of parameters and significantly smaller than transformer networks.

As such, our hybrid model achieves the twofold objective set, whilst also enhancing robustness through the noise filtering techniques and the high adaptability of LSTM networks. Furthermore, the combination of low-rank decomposition and the LSTM network diversifies the risk of model failure, as different components handle various types of behavior in the data. This approach also improves forecasting accuracy, by effectively combining the strengths of linear and non-linear modeling.

5.3.2 Algorithm of Our Hybrid Model

During the training phase of our algorithm, the following sequence of steps is taken:

1. Denoise the time series signal using either SSA or PCA

2. Transform the denoised time series of each EV user into its page matrix representation

and concatenate all matrices into a stacked page matrix (depicted as the blue matrix in step I of Figure 5.1)

3. Apply HSVT to decompose the time series into its non-stationary and stationary components (illustrated as orange and green matrices, respectively, in step II of Figure 5.1)

4. Obtain the stationary page matrix of each EV user from the stationary stacked page matrix, which contains T/L columns per user (shown as the green matrix in step II of Figure 5.1). Specifically, the first T/L columns represent User 1, the next T/L columns represent User 2, and so on. Subsequently, convert this data back into the time series format and train a separate LSTM network for each user using their individual time series data

The training stage is crucial as it allows us to train the LSTM networks, which are the only components of our hybrid model with trainable parameters. This process is tailored to capture the unique patterns in the arrival times for each EV user.

In the inference stage, the first three steps are repeated as both SSA/PCA and low-rank decomposition are non-parametric statistical techniques. Following this, the parameters of the linear AR model, $\hat{\beta}$, are estimated using the non-stationary part of the stacked page matrix, in a similar way to the SAMoSSA model, as described in Section 5.1.2 and graphically depicted in stage III of Figure 5.1. Finally, to produce the next-step prediction for each user, the non-stationary prediction from the linear model and the stationary prediction from the trained LSTM model are summed. To generate these predictions, the previous L steps are utilized in a sliding window fashion. Specifically, for the first prediction, the last L points of the training series are used. For the next prediction, the last $L-1$ points of the training series, combined with the first point of the testing set, are used. This pattern continues, progressively incorporating each subsequent point from the testing set into the sliding window of size L.

5.3.3 Model Training

All models were implemented in Python due to its extensive availability of libraries and packages that facilitate the implementation of time series models. For the ARIMA model, the `statsmodels` module was utilized [134], while for Prophet, a Python API is available [135]. On the other hand, the iTransformer and DeepAR models can be accessed through `GluonTS`, a Python package for probabilistic time series modeling that focuses on deep learning based models [136], and the Temporal Fusion Transformer (TFT) is available in the `pytorch-forecasting` package [137]. For the SAMoSSA model, a custom library is used [138]. Finally, our hybrid model is implemented from scratch, utilizing the SVD implementation from NumPy and the LSTM layer from the Keras deep learning library.

For hyperparameter tuning, different strategies were followed depending on the model. For the ARIMA model, the optimal value for d was obtained through the ADF test, whilst the ACF and PACF were used to determine the p and q values, respectively, using a 95% confidence interval (resulting in a threshold of $\frac{1.96}{\sqrt{N}}$). For the Prophet and iTransformer models, the default hyperparameter settings provided in the respective models were utilized. Conversely, for the DeepAR and TFT models, Optuna [139] was employed due to the large number of hyperparameters that need to be tuned. Optuna automates the search for optimal hyperparameters, allowing for efficient exploration of large hyperparameter spaces and enabling the pruning of unpromising trials while parallelizing hyperparameter searches. This approach is significantly more efficient and yields more accurate results compared to the traditional combination of grid search and cross-validation used to find optimal hyperparameter values. For the SAMoSSA model, as only the value of L needs to be decided, this was adjusted manually, with the optimal value determined using a validation set.

Finally, for our hybrid model, the hyperparameters that needed to be determined were those present in the LSTM network. These included the number of layers, the width of each layer, the activation function, the number of epochs, and any necessary regularization techniques. A validation dataset was used to determine the optimal architecture and training settings for the LSTM network. The most effective configuration consisted of a two-layer

LSTM, with the first layer having 64 units and the second 32 units, both utilizing a softmax activation function. Additionally, a dropout rate of 0.2 was added after both layers to regularize the model and prevent overfitting. The training process utilised the AdamW optimizer [140] with a weight decay of 0.01 and early stopping set at a patience of 20. The AdamW optimizer was selected as it effectively decouples the weight decay from the optimization steps, leading to more effective training. Moreover, the combination of weight decay and early stopping served as key regularisation techniques to prevent overfitting, a crucial aspect given the limited size of our training dataset.

5.4 Experimental Results

To assess the performance of our hybrid model against the SOTA models currently available in the literature, we calculated the SMAPE error for the arrival time of EV users in the test set. The SMAPE metric was selected for the same reasons as detailed in Section 3.3. The one-step ahead time series forecasting error for the arrival times of EV users over one month of the testing dataset, for all models described in Section 5.1, is detailed in Table 5.1.

Model	SMAPE on Arrival Time Prediction (%)
SAMoSSA	6.93
Prophet	6.52
iTransformer	6.38
DeepAR	5.73
TFT	4.82
ARIMA model	5.11
Hybrid model (ours)	**3.73**

Table 5.1: SMAPE on arrival time prediction for the different time series forecasting models implemented.

As observed from Table 5.1, our hybrid model significantly reduces the arrival time prediction error. Compared to the SAMoSSA model, which employs a similar statistical method to predict the non-stationary part, our model reduces the error by 46.2%. This underscores the importance of using noise filtering techniques to obtain a cleaner signal from noisy data and highlights the effectiveness of deep learning approaches, such as LSTM, over linear models for the stationary part. This is because, these approaches can capture non-linearities and long-term dependencies not addressed by the decomposition employed for the non-stationary

part.

Furthermore, our hybrid model reduces the error relative to the SOTA deep learning models, in particular DeepAR and TFT, by 34.9% and 22.6% respectively, despite its significantly smaller size. This demonstrates our model's capability to achieve highly accurate results with smaller training datasets, due to its combination of deep learning and statistical methods. These methods not only reduce the number of trainable parameters but also diversify the risk of model failure, unlike deep learning models which require substantial amounts of data to train effectively. Additionally, the smaller size of our model offers significant advantages, requiring less training time to achieve good results and displaying greater robustness to hyperparameter tuning. This contrasts with larger deep learning models, which often require extensive tuning of numerous hyperparameters to achieve optimal performance. This is also evident in the results of the ARIMA model, which, despite its simplicity, achieves a lower error than two of the most accurate deep learning models for time series forecasting, namely DeepAR and iTransformer.

Although our primary focus was on forecasting the arrival times of EV users, our hybrid model is adaptable to any time series forecasting task. To evaluate its adaptability and effectiveness on different data, we tested the model on the UCI repository's electricity dataset, which consists of hourly electricity loads from 370 households [141]. This dataset is a popular choice for time series prediction testing and has been extensively used in the literature. To reduce the computational demands and assess model performance with limited training data – a key goal of our time series forecasting approach – we randomly selected 50 of the 370 households. The training dataset was limited to 48 data points, with the objective to predict the next 24 data points using a sliding window approach (one-step ahead forecasting). The results are summarized in Table 5.2.

Our hybrid model also achieves the best performance in terms of SMAPE error on the electricity dataset. Specifically, it reduces the SMAPE error by 38.1% relative to SAMoSSA and by 30.2% compared to the best deep learning model, which in this dataset is the iTransformer. These results underscore our hybrid model's capability to achieve high accuracy

Model	SMAPE on Electricity Load Prediction (%)
SAMoSSA	12.08
Prophet	18.51
iTransformer	10.71
DeepAR	11.67
TFT	16.89
ARIMA model	14.05
Hybrid model (ours)	**7.48**

Table 5.2: SMAPE on hourly electricity load prediction for the different time series forecasting models implemented.

across various time series forecasting tasks, independent of data frequency, particularly with small and noisy training datasets – areas where deep learning models often struggle. This is an important outcome, considering that many time series forecasting tasks involve limited data with substantial variability. Such conditions typically compromise the performance of deep learning models, despite their higher complexity and greater expressive capabilities compared to traditional statistical methods. However, these challenges do not impact our model, due to its particular aforementioned characteristics.

6 Conclusion

6.1 Summary of Project Achievements

In this work, we have presented a new framework for predicting the EV user behavior by estimating two of the most important EV charging parameters with regards to scheduling, namely session duration and energy consumption. Unlike previous work, our method leverages cross-user correlations and historical data from multiple users to improve predictions. Moreover, the use of an ensemble learning algorithm achieves better results compared to any single-model performance. The results obtained in terms of predictions show significant improvements over past approaches, underscoring the significance of exploiting data correlations to mitigate the impact of unforeseen events and data variability within individual user datasets. Additionally, our ensemble approach enables us to tailor our model to accommodate the diverse behaviors of different users, moving away from a less effective 'one-size-fits-all' model. Overall, compared to the best model in the literature, we have achieved reductions in SMAPE of **24.2**% for stay duration and **24.5**% for energy consumption predictions, respectively, using the same dataset comprising real charging data from the JPL station.

Utilizing the improved predictions, we have demonstrated the effectiveness of our online Adaptive Charging Algorithm which employs Model Predictive Control, in achieving our twofold objective function: minimizing peak load demand and reducing the charging cost to EV owners, thereby maximizing consumer welfare. Moreover, we have shown that utilizing our predicted values as inputs, as opposed to direct user inputs, significantly enhances the performance of the ACA algorithm, as they effectively eliminate the unrealistic and exaggerated demands of users that exist due to range anxiety. Overall, our optimized charging algorithm with predicted inputs achieves a substantial reduction in peak load demand, with a daily average decrease of **76.3**% across one week of testing, resulting in a corresponding decrease in total charging costs to EV owners of **16.4**%.

Furthermore, by incorporating three-phase infrastructure constraints in the optimization

setting of our ACA algorithm, we ensure that our simulations closely align with real-world conditions. To the best of our knowledge, our study is the first to incorporate predicted inputs, which have been proven to be more reliable than direct user inputs, in an optimized charging algorithm that accounts for the behavior and complexities of three-phase systems.

Recognizing the critical need for accurate arrival time predictions at large-scale, high-density charging facilities, such as a workplace parking, for day-ahead demand scheduling, we have extended our work by developing time series forecasting models. Our hybrid model employs a combination of statistical and deep learning methods to leverage the advantages offered by each, managing to achieve SOTA results. Particularly, our hybrid model reduces the forecasting error for EV arrival times by 46.2% compared to SAMoSSA and by 22.6% compared to TFT, which are often considered the most accurate statistical and deep learning models, respectively. Additionally, our model excels not only in daily frequency forecasting tasks, such as predicting the arrival times of EV users, but also in any time series forecasting task regardless of data frequency. This is demonstrated by the results achieved in the hourly electricity load prediction task, one of the most extensively tested datasets in the literature, underscoring the model's adaptability and effectiveness due to its unique architecture and characteristics.

6.2 Future Work

Overall, the project has successfully met its objectives by accurately estimating EV user behavior and optimizing EV charging, while also extending its application to time series forecasting tasks. Nonetheless, there are multiple ideas that could further enhance and expand all parts of our work.

For predicting the stay duration and energy consumption of EV users, incorporating additional factors such as weather conditions, traffic patterns, and local events data can be considered. Moreover, leveraging social media platforms, like Google Maps, can also be explored as a means to gather information about local events and driver behaviors. This is because, social media has been shown in past studies to be a good tool for estimating

human behaviour [142] and is also a significant predictor of travel times for truck drivers [143]. Furthermore, considering that the two metrics implemented – Pearson Correlation Coefficient and Cosine Similarity – measure linear relationships, there is an opportunity to explore methods that detect more complex and non-linear relationships. Drawing from recommender systems and collaborative filtering, the use of non-parametric methods such as Adjusted Mutual Information (AMI), Kendall's Tau, and K-Nearest Neighbors (KNN) with kernel functions could be investigated to address this need.

Additionally, to further align the simulations of our optimized charging algorithm with real-world scenarios beyond incorporating three-phase infrastructure constraints, integrating a realistic model for battery charging behavior could be considered. Currently, an ideal model is used, where EVs are assumed to follow the given pilot signal exactly. However, in practice, the charging rate of an EV is often strictly lower than the pilot signal and tends to decay as the battery approaches 100% state of charge [17, 29]. This discrepancy can significantly increase the total charging time required and results in the under-utilization of infrastructure capacity. Moreover, although the simulations were conducted during the busiest week in our testing dataset, with a maximum of 14 vehicles being charged per day, exploring how our optimized charging algorithm performs with a larger number of EVs, such as 52 when the JPL station is full, represents another potential path for our future work. This is because, it is crucial to assess how reductions in peak load and charging costs scale with an increasing number of EVs.

Finally, although time series forecasting is not the primary focus of our work, and our hybrid model has achieved SOTA results in predicting the arrival time of EV users in single-step forecasting, extending this model to multi-step time series prediction is a promising future direction. One approach is recursion, where the model predicts one time step ahead and then uses that forecast as input for the next step, continuing this process until the desired forecast horizon is reached. However, due to the accumulation of prediction errors with each additional step, ensuring the system's stability is crucial, as an unstable system will lead to divergent predictions after a certain number of steps. To address this, one

potential solution is to orthogonally project the potentially unstable solution, obtained from the initially unconstrained optimization problem aimed at minimizing the mean square error of one-step ahead forecasts, into a stable subspace. Although this may lead to a solution that is not optimal in the least squares sense for one-step ahead predictions, it ensures system stability and prevents divergence. Furthermore, this projection can also act as a noise reduction technique since it involves projecting the solution into a low-rank subspace, potentially improving the performance of one-step ahead predictions by acting as additional noise filtering. Thus, this is another aspect we can explore further.

Bibliography

1. Commission, E. 2050 long-term strategy. https://climate.ec.europa.eu/eu-action/climate-strategies-targets/2050-long-term-strategy_en (2020).

2. House, T. W. President Biden to Catalyze Global Climate Action through the Major Economies Forum on Energy and Climate. https://www.whitehouse.gov/briefing-room/statements-releases/2023/04/20/fact-sheet-president-biden-to-catalyze-global-climate-action-through-the-major-economies-forum-on-energy-and-climate (2023).

3. Valogianni, K., Ketter, W. & Collins, J. Smart charging of electric vehicles using reinforcement learning. *AAAI Workshop - Technical Report,* 41–48 (Jan. 2013).

4. Petroff, A. These countries want to ditch gas and diesel cars (June 2017).

5. Of California, S. Climate Change Investment Plan (2018).

6. The EV transistion explained. https://spectrum.ieee.org/the-ev-transition-explained-2658463709 (Nov. 2022).

7. Lopes, J. A. P., Soares, F. J. & Almeida, P. M. R. Integration of Electric Vehicles in the Electric Power System. *Proceedings of the IEEE* **99,** 168–183 (2011).

8. Chung, Y.-W., Khaki, B., Li, T., Chu, C. & Gadh, R. Ensemble machine learning-based algorithm for electric vehicle user behavior prediction. *Applied Energy* **254.** https://ideas.repec.org/a/eee/appene/v254y2019ics0306261919314199.html (2019).

9. Mu, Y., Wu, J., Jenkins, N. & Wang, C. A Spatial–Temporal model for grid impact analysis of plug-in electric vehicles. *Applied Energy* **114,** 456–465 (Feb. 2014).

10. Salah, F., Ilg, J. P., Flath, C. M., Basse, H. & van Dinther, C. Impact of electric vehicles on distribution substations: A Swiss case study. *Applied Energy* **137,** 88–96. https://www.sciencedirect.com/science/article/pii/S0306261914010393 (2015).

11. Foley, A., Tyther, B., Calnan, P. & Ó Gallachóir, B. Impacts of Electric Vehicle charging under electricity market operations. *Applied Energy* **101.** Sustainable Development of Energy, Water and Environment Systems, 93–102. https://www.sciencedirect.com/science/article/pii/S0306261912004977 (2013).

12. Johnson, T. Americans spend an average of 17,600 minutes driving each year. https://newsroom.aaa.com/tag/american-driving-survey/ (2018).

13. Gan, L., Topcu, U. & Low, S. *Optimal decentralized protocol for electric vehicle charging* in *2011 50th IEEE Conference on Decision and Control and European Control Conference* (2011), 5798–5804.

14. Lee, Z. J., Li, T. & Low, S. H. *ACN-Data: Analysis and Applications of an Open EV Charging Dataset* in *Proceedings of the Tenth ACM International Conference on*

Future Energy Systems (Association for Computing Machinery, Phoenix, AZ, USA, 2019), 139–149. ISBN: 9781450366717. https://doi.org/10.1145/3307772.3328313.

15. Khaki, B., Chu, C. & Gadh, R. Hierarchical distributed framework for EV charging scheduling using exchange problem. *Applied Energy* **241,** 461–471. ISSN: 0306-2619. https://www.sciencedirect.com/science/article/pii/S0306261919304027 (2019).

16. Bian, H., Guo, Z., Zhou, C. & Peng, S. Multi-time scale electric vehicle charging load forecasting considering constant current charging and parallel computing. *Energy Reports* **8.** 2022 The 5th International Conference on Electrical Engineering and Green Energy, 722–732. https://www.sciencedirect.com/science/article/pii/S2352484722014779 (2022).

17. Lee, Z. J. *et al. Large-Scale Adaptive Electric Vehicle Charging* in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (2018), 863–864.

18. Tableau. Time Series Forecasting: Definition, Applications, and Examples. https://www.tableau.com/learn/articles/time-series-forecasting.

19. Clement, K., Haesen, E. & Driesen, J. *Coordinated charging of multiple plug-in hybrid electric vehicles in residential distribution grids* in *2009 IEEE/PES Power Systems Conference and Exposition* (2009), 1–7.

20. Roe, C., Farantatos, E., Meisel, J., Meliopoulos, A. & Overbye, T. *Power System Level Impacts of PHEVs* in *2009 42nd Hawaii International Conference on System Sciences* (2009), 1–10.

21. Ma, Z., Callaway, D. S. & Hiskens, I. A. Decentralized Charging Control of Large Populations of Plug-in Electric Vehicles. *IEEE Transactions on Control Systems Technology* **21,** 67–78 (2013).

22. Chen, Z. *et al. Distributed Charging Control of Electric Vehicles Considering Cooperative Factor in Photovoltaic Charging Station* in *2021 3rd Asia Energy and Electrical Engineering Symposium (AEEES)* (2021), 839–845.

23. Alsabbagh, A. & Ma, C. *Distributed Charging Management of Electric Vehicles with Charging Anxiety for Charging Cost Reduction* in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)* (2019), 2489–2494.

24. Shao, S., Zhang, T., Pipattanasomporn, M. & Rahman, S. *Impact of TOU rates on distribution load shapes in a smart grid with PHEV penetration* in *IEEE PES TD 2010* (2010), 1–6.

25. Liu, H. *et al. Multi-objective Optimization of Electric Vehicle Charging under TOU Price with Tidal Energy Access* in *2023 5th Asia Energy and Electrical Engineering Symposium (AEEES)* (2023), 1230–1234.

26. Putri, S. M., Ashari, M., Endroyono & Suryoatmojo, H. *EV Charging Scheduling with Genetic Algorithm as Intermittent PV Mitigation in Centralized Residential Charging*

Stations in *2023 International Seminar on Intelligent Technology and Its Applications (ISITIA)* (2023), 286–291.

27. Dahmane, Y., Ghanes, M., Chenouard, R. & Alvarado-Ruiz, M. *Decentralized Control of Electric Vehicle Smart Charging for Cost Minimization Considering Temperature and Battery Health* in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)* (2019), 1–6.

28. Liu, M., Phanivong, P. K. & Callaway, D. S. *Customer-and Network-Aware Decentralized EV Charging Control* in *2018 Power Systems Computation Conference (PSCC)* (2018), 1–7.

29. Wang, Q., Liu, X., Du, J. & Kong, F. Smart Charging for Electric Vehicles: A Survey From the Algorithmic Perspective. *IEEE Communications Surveys Tutorials* **18,** 1500–1517 (2016).

30. Gan, L., Topcu, U. & Low, S. *Optimal decentralized protocol for electric vehicle charging* in *2011 50th IEEE Conference on Decision and Control and European Control Conference* (2011), 5798–5804.

31. Ma, Z., Callaway, D. S. & Hiskens, I. A. Decentralized Charging Control of Large Populations of Plug-in Electric Vehicles. *IEEE Transactions on Control Systems Technology* **21,** 67–78 (2013).

32. Ahn, C., Li, C.-T. & Peng, H. *Decentralized charging algorithm for electrified vehicles connected to smart grid* in *Proceedings of the 2011 American Control Conference* (2011), 3924–3929.

33. Lee, G., Lee, T., Low, Z., Low, S. H. & Ortega, C. *Adaptive charging network for electric vehicles* in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (2016), 891–895.

34. Wang, B. *et al.* Predictive Scheduling Framework for Electric Vehicles With Uncertainties of User Behaviors. *IEEE Internet of Things Journal* **4,** 52–63 (2017).

35. He, X. *A Survey on Time Series Forecasting* in *3D Imaging—Multidimensional Signal Processing and Deep Learning* (eds Patnaik, S., Kountchev, R., Tai, Y. & Kountcheva, R.) (Springer Nature Singapore, Singapore, 2023), 13–23.

36. Edwards, R., Magee, J. & Bassetti, W. Technical Analysis of Stock Trends. *CRC Press* (2018).

37. Qu, L., Li, W., Li, W., Ma, D. & Wang, Y. Daily long-term traffic flow forecasting based on a deep neural network. *Expert Syst. Appl.* **121,** 304–312. https://api.semanticscholar.org/CorpusID:59528742 (2019).

38. Ward, S. N. Area-based tests of long-term seismic hazard predictions. *Bulletin of the Seismological Society of America* **85,** 1285–1298 (Oct. 1995).

39. Topol, E. J. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine* **25,** 44–56. https://api.semanticscholar.org/CorpusID:57574615 (2019).

40. Andersen, T., Bollerslev, T., Christoffersen, P. & Diebold, F. *Volatility Forecasting* NBER Working Papers 11188 (National Bureau of Economic Research, Inc, 2005). https://EconPapers.repec.org/RePEc:nbr:nberwo:11188.

41. Yule, G. U. On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers. *Philosophical Transactions of the Royal Society A* **226,** 267–298. https://api.semanticscholar.org/CorpusID:122033085.

42. Walker, G. T. On Periodicity in Series of Related Terms. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* **131,** 518–532. https://api.semanticscholar.org/CorpusID:120223042 (1931).

43. Rojas, I. *et al.* Soft-computing techniques and ARMA model for time series prediction. *Neurocomputing* **71.** Neural Networks: Algorithms and Applications 50 Years of Artificial Intelligence: a Neuronal Approach, 519–537. ISSN: 0925-2312. https://www.sciencedirect.com/science/article/pii/S0925231207002858 (2008).

44. Chen, Z., Ma, M., Li, T., Wang, H. & Li, C. Long sequence time-series forecasting with deep learning: A survey. *Information Fusion* **97,** 101819. ISSN: 1566-2535. https://www.sciencedirect.com/science/article/pii/S1566253523001355 (2023).

45. Cortes, C. & Vapnik, V. N. Support-Vector Networks. *Machine Learning* **20,** 273–297. https://api.semanticscholar.org/CorpusID:52874011 (1995).

46. Freund, Y. Boosting a weak learning algorithm by majority. *Inf. Comput.* **121,** 256–285. https://api.semanticscholar.org/CorpusID:19728033 (1990).

47. Huang, Q. & Wei, S. Improved quantile convolutional neural network with two-stage training for daily-ahead probabilistic forecasting of photovoltaic power. *Energy Conversion and Management* **220,** 113085. https://api.semanticscholar.org/CorpusID:224916583 (2020).

48. Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A. & Jenssen, R. *Recurrent Neural Networks for Short-Term Load Forecasting* in *SpringerBriefs in Computer Science* (2017). https://api.semanticscholar.org/CorpusID:19141787.

49. Zeroual, A., Harrou, F., Dairi, A. & Sun, Y. Deep learning methods for forecasting COVID-19 time-Series data: A Comparative study. *Chaos, Solitons, and Fractals* **140,** 110121–110121. https://api.semanticscholar.org/CorpusID:220526685 (2020).

50. Zhang, P. & Ci, B. Deep belief network for gold price forecasting. *Resources Policy* **69,** 101806. https://api.semanticscholar.org/CorpusID:225010875 (2020).

51. Wu, N., Green, B., Ben, X. & O'Banion, S. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. *ArXiv* **abs/2001.08317.** https://api.semanticscholar.org/CorpusID:210861210 (2020).

52. Liu, Y. *et al. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting* 2023. arXiv: 2310.06625 [cs.LG].

53. Chu, J., Cao, J. & Chen, Y. *An Ensemble Deep Learning Model Based on Transformers for Long Sequence Time-Series Forecasting* in *Neural Computing for Advanced Applications* (Springer Nature Singapore, Singapore, 2022), 273–286. ISBN: 978-981-19-6135-9.

54. Ryo, M. & Rillig, M. C. Statistically reinforced machine learning for nonlinear patterns and variable interactions. *Ecosphere* **8,** e01976. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.1002/ecs2.1976. https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1002/ecs2.1976 (2017).

55. Adadi, A. A survey on data-efficient algorithms in big data era. *Journal of Big Data* **8** (Jan. 2021).

56. Herm, L.-V., Heinrich, K., Wanner, J. & Janiesch, C. Stop ordering machine learning algorithms by their explainability! A user-centered investigation of performance and explainability. *International Journal of Information Management* **69,** 102538. ISSN: 0268-4012. https://www.sciencedirect.com/science/article/pii/S026840122200072X (2023).

57. Cooper, A. F., Lu, Y. & Sa, C. D. *Hyperparameter Optimization Is Deceiving Us, and How to Stop It* in *Neural Information Processing Systems* (2021). https://api.semanticscholar.org/CorpusID:231839557.

58. Molnar, C. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable* 2nd ed. https://christophm.github.io/interpretable-ml-book (2022).

59. Awad, M. & Khanna, R. in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers* 67–80 (Apress, Berkeley, CA, 2015). https://doi.org/10.1007/978-1-4302-5990-9_4.

60. Kokash, N. & Makhnist, L. *Using Decision Trees for Interpretable Supervised Clustering* July 2023.

61. Uddin, S., Haque, I., Lu, H., Moni, M. A. & Gide, E. Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction. en. *Sci. Rep.* **12,** 6256 (Apr. 2022).

62. Botev, Z. I., Grotowski, J. F. & Kroese, D. P. Kernel density estimation via diffusion. *The Annals of Statistics* **38,** 2916–2957. https://doi.org/10.1214/10-AOS799 (2010).

63. Mood, A., Graybill, F. & Boes, D. *Introduction to the Theory of Statistics* ISBN: 9780070428645. https://books.google.com/books?id=Viu2AAAAIAAJ (McGraw-Hill, 1973).

64. *Gram matrix - Wikipedia — en.wikipedia.org* https://en.wikipedia.org/wiki/Gram_matrix. [Accessed 03-03-2024].

65. *Moment matrix - Wikipedia — en.wikipedia.org* https://en.wikipedia.org/wiki/Moment_matrix. [Accessed 03-03-2024].

66. Curry, H. B. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics* **2,** 258–261. https://api.semanticscholar.org/CorpusID:125304075 (1944).

67. Altman, N. S. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician* **46,** 175–185 (1992).

68. Quinlan, J. R. Induction of Decision Trees. *Machine Learning* **1,** 81–106. https://api.semanticscholar.org/CorpusID:189902138 (1986).

69. Sieling, D. Minimization of decision trees is hard to approximate. *Journal of Computer and System Sciences* **74.** Computational Complexity 2003, 394–403. ISSN: 0022-0000. https://www.sciencedirect.com/science/article/pii/S0022000007000815 (2008).

70. Strobl, C., Malley, J. & Tutz, G. An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests. en. *Psychol. Methods* **14,** 323–348 (Dec. 2009).

71. Breiman, L. Random forests. *Machine learning* **45,** 5–32 (2001).

72. Smola, A. & Schölkopf, B. A tutorial on support vector regression. *Statistics and Computing* **14,** 199–222. https://api.semanticscholar.org/CorpusID:15475 (2004).

73. Vapnik, V. *The nature of statistical learning theory* (Springer science & business media, 2013).

74. Bussotti, P. On the genesis of the Lagrange multipliers. *J. Optim. Theory Appl.* **117,** 453–459 (June 2003).

75. Wright, S. J. *Continuous Optimization (Nonlinear and Linear Programming)* in (2012). https://api.semanticscholar.org/CorpusID:12421760.

76. Wang, J., Chen, Q. & Chen, Y. *RBF Kernel Based Support Vector Machine with Universal Approximation and Its Application* in *Advances in Neural Networks – ISNN 2004* (eds Yin, F.-L., Wang, J. & Guo, C.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004), 512–517.

77. Bishop, C. M. *Pattern Recognition and Machine Learning* (Springer, New York, 2006).

78. Silverman, B. W. *Density Estimation for Statistics and Data Analysis* (Routledge, New York, 1988).

79. Botev, Z., Grotowski, J. & Kroese, D. Kernel Density Estimation via Diffusion. *The Annals of Statistics* **38** (Nov. 2010).

80. Lee, Z. J., Sharma, S., Johansson, D. & Low, S. H. *ACN-Sim: An Open-Source Simulator for Data-Driven Electric Vehicle Charging Research* 2021. arXiv: 2012.02809 [eess.SY].

81. Shahriar, S., Al-Ali, A. R., Osman, A. H., Dhou, S. & Nijim, M. Prediction of EV Charging Behavior Using Machine Learning. *IEEE Access* **9,** 111576–111586 (2021).

82. Lee, Z. J. *et al.* Adaptive Charging Networks: A Framework for Smart Electric Vehicle Charging. *IEEE Transactions on Smart Grid* **12,** 4339–4350 (2021).

83. Guo, L., Erliksson, K. F. & Low, S. H. *Optimal online adaptive electric vehicle charging* in *2017 IEEE Power Energy Society General Meeting* (2017), 1–5.

84. Nakahira, Y., Chen, N., Chen, L. & Low, S. H. Smoothed Least-laxity-first Algorithm for EV Charging. *Proceedings of the Eighth International Conference on Future Energy Systems*. https://api.semanticscholar.org/CorpusID:1014915 (2017).

85. Buitinck, L. *et al. API design for machine learning software: experiences from the scikit-learn project* 2013. arXiv: 1309.0238 [cs.LG].

86. Pearson, K. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London Series I* **58,** 240–242 (Jan. 1895).

87. Sidorov, G., Gelbukh, A., Gómez-Adorno, H. & Pinto, D. Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model. *Computación y Sistemas* **18.** https://api.semanticscholar.org/CorpusID:14713935 (2014).

88. Akoğlu, H. User's guide to correlation coefficients. *Turkish Journal of Emergency Medicine* **18,** 91–93. https://api.semanticscholar.org/CorpusID:52171443 (2018).

89. Majidpour, M., Qiu, C., Chu, P., Gadh, R. & Pota, H. R. Fast Prediction for Sparse Time Series: Demand Forecast of EV Charging Stations for Cell Phone Applications. *IEEE Transactions on Industrial Informatics* **11,** 242–250 (2015).

90. Majidpour, M., Qiu, C., Gadh, R., Chu, P. & Pota, H. R. *A novel forecasting algorithm for electric vehicle charging stations* in *2014 International Conference on Connected Vehicles and Expo (ICCVE)* (2014), 1035–1040.

91. Lilhore, A., Prasad, K. K. & Agarwal, V. *Machine Learning-based Electric Vehicle User Behavior Prediction* in *2023 IEEE IAS Global Conference on Renewable Energy and Hydrogen Technologies (GlobConHT)* (2023), 1–6.

92. Zimmerman, R. D., Murillo-Sánchez, C. E. & Thomas, R. J. MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education. *IEEE Transactions on Power Systems* **26,** 12–19 (2011).

93. Thurner, L. *et al.* Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. *IEEE Transactions on Power Systems* **33,** 6510–6521 (2018).

94. Coffrin, C., Bent, R. W., Sundar, K., Ng, Y. & Lubin, M. PowerModels.J1: An Open-Source Framework for Exploring Power Flow Formulations. *2018 Power Systems Computation Conference (PSCC)*, 1–8. https://api.semanticscholar.org/CorpusID:3830967 (2017).

95. Montenegro, D., Dugan, R., Henry, R., McDermott, T. & Sunderm, W. *OpenDSS - EPRI Distribution System Simulator* https://sourceforge.net/projects/electricdss/.

96. Chassin, D. P., Fuller, J. C. & Djilali, N. *GridLAB-D: An agent-based simulation framework for smart grids* 2014. arXiv: 1405.3136 [nlin.AO].

97. Stankovic, J. A., Spuri, M., Ramamritham, K. & Buttazzo, G. C. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms* (Springer Science & Business Media, 2012).

98. Lee, Z. J. & Sharma, S. *adacharge* Nov. 2020. https://github.com/caltech-netlab/adacharge.

99. Diamond, S. & Boyd, S. P. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of machine learning research : JMLR* **17.** https://api.semanticscholar.org/CorpusID:6298008 (2016).

100. Agrawal, A., Verschueren, R., Diamond, S. & Boyd, S. P. A Rewriting System for Convex Optimization Problems. *ArXiv* **abs/1709.04494.** https://api.semanticscholar.org/CorpusID:67856259 (2017).

101. Ravi, S. S. & Aziz, M. Utilization of Electric Vehicles for Vehicle-to-Grid Services: Progress and Perspectives. *Energies* **15.** ISSN: 1996-1073. https://www.mdpi.com/1996-1073/15/2/589 (2022).

102. PowerFlex. *Download the App* 2024. https://www.powerflex.com/download-the-app.

103. PowerFlex. *PowerFlex Case Study: NASA Jet Propulsion Laboratory* 2023.

104. Clean Energy Group. *An Introduction to Demand Charges* tech. rep. Accessed: 2024-05-16 (National Renewable Energy Laboratory, 2017). https://www.cleanegroup.org/wp-content/uploads/Demand-Charge-Fact-Sheet.pdf.

105. Box, G. E. P. & Jenkins, G. *Time Series Analysis, Forecasting and Control* ISBN: 0816211043 (Holden-Day, Inc., USA, 1990).

106. Jalil, A. & Rao, N. H. in *Environmental Kuznets Curve (EKC)* (eds Özcan, B. & Öztürk, I.) 85–99 (Academic Press, 2019). ISBN: 978-0-12-816797-7. https://www.sciencedirect.com/science/article/pii/B9780128167977000084.

107. Alomar, A., Dahleh, M., Mann, S. & Shah, D. *SAMoSSA: Multivariate Singular Spectrum Analysis with Stochastic Autoregressive Noise* 2023. arXiv: 2305.16491 [cs.LG].

108. Agarwal, A., Alomar, A. & Shah, D. *On Multivariate Singular Spectrum Analysis and its Variants* in (2020). https://api.semanticscholar.org/CorpusID:267811873.

109. Damen, A. A. H., den Hof, V. & Hajdasiński, A. K. *The page matrix : an excellent tool for noise filtering of Markov parameters, order testing and realization* in (1982). https://api.semanticscholar.org/CorpusID:122758913.

110. Basodi, S., Ji, C., Zhang, H. & Pan, Y. *Gradient Amplification: An efficient way to train deep neural networks* 2020. arXiv: 2006.10560 [cs.LG].

111. Sherstinsky, A. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena* **404,** 132306. ISSN: 0167-2789. http://dx.doi.org/10.1016/j.physd.2019.132306 (Mar. 2020).

112. Hochreiter, S. & Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **9,** 1735–1780. ISSN: 0899-7667. https://doi.org/10.1162/neco.1997.9.8.1735 (Nov. 1997).

113. Taylor, S. J. & Letham, B. Forecasting at Scale. *The American Statistician* **72,** 37–45 (2018).

114. Hastie, T. & Tibshirani, R. Generalized Additive Models. *Statistical Science* **1,** 297–310. https://doi.org/10.1214/ss/1177013604 (1986).

115. Byrd, R. H., Lu, P., Nocedal, J. & Zhu, C. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing* **16,** 1190–1208. https://doi.org/10.1137/0916069 (1995).

116. Liu, Y. *et al. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting* 2024. arXiv: 2310.06625 [cs.LG].

117. Vaswani, A. *et al. Attention is All you Need* in *Advances in Neural Information Processing Systems* (eds Guyon, I. *et al.*) **30** (Curran Associates, Inc., 2017).

118. Das, A. *et al. Long-term Forecasting with TiDE: Time-series Dense Encoder* 2024. arXiv: 2304.08424 [stat.ML].

119. Salinas, D., Flunkert, V. & Gasthaus, J. *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks* 2019. arXiv: 1704.04110 [cs.AI].

120. Graves, A. *Generating Sequences With Recurrent Neural Networks* 2014. arXiv: 1308.0850 [cs.NE].

121. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to Sequence Learning with Neural Networks* 2014. arXiv: 1409.3215 [cs.CL].

122. Koya, S. R. & Roy, T. *Temporal Fusion Transformers for Streamflow Prediction: Value of Combining Attention with Recurrence* 2023. arXiv: 2305.12335 [cs.LG].

123. Dauphin, Y. N., Fan, A., Auli, M. & Grangier, D. *Language Modeling with Gated Convolutional Networks* 2017. arXiv: 1612.08083 [cs.CL].

124. Li, S. *et al. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting* 2020. arXiv: 1907.00235 [cs.LG].

125. Golyandina, N., Nekrutkin, V. & Zhigljavsky, A. A. *Analysis of Time Series Structure: SSA and Related Techniques* 1st. https://doi.org/10.1201/9780367801687 (Chapman and Hall/CRC, 2001).

126. Markovsky, I. Structured low-rank approximation and its applications. *Automatica* **44,** 891–909. ISSN: 0005-1098. https://www.sciencedirect.com/science/article/pii/S0005109807003950 (2008).

127. Hansen, P. & Jensen, S. Subspace-Based Noise Reduction for Speech Signals via Diagonal and Triangular Matrix Decompositions: Survey and Analysis. *EURASIP Journal on Advances in Signal Processing* **2007,** 092953. https://doi.org/10.1155/2007/92953 (2007).

128. Hermus, K., Wambacq, P. & Van hamme, H. A Review of Signal Subspace Speech Enhancement and Its Application to Noise Robust Speech Recognition. *EURASIP J. Adv. Sig. Proc.* **2007** (Jan. 2007).

129. Packard, N. H., Crutchfield, J. P., Farmer, J. D. & Shaw, R. S. Geometry from a Time Series. *Phys. Rev. Lett.* **45,** 712–716. https://link.aps.org/doi/10.1103/PhysRevLett.45.712 (9 Sept. 1980).

130. Candes, E. J., Li, X., Ma, Y. & Wright, J. *Robust Principal Component Analysis?* 2009. arXiv: 0912.3599 [cs.IT].

131. Hu, Z., Zhao, H. & Peng, J. Low-rank reconstruction-based autoencoder for robust fault detection. *Control Engineering Practice* **123,** 105156. ISSN: 0967-0661. https://www.sciencedirect.com/science/article/pii/S0967066122000612 (2022).

132. Kieu, T. *et al. Robust and Explainable Autoencoders for Unsupervised Time Series Outlier Detection—Extended Version* 2022. arXiv: 2204.03341 [cs.LG].

133. Heindel, L., Hantschke, P. & Kästner, M. A data-driven approach for approximating non-linear dynamic systems using LSTM networks. *Procedia Structural Integrity.* https://api.semanticscholar.org/CorpusID:247533768 (2022).

134. Statsmodels. *Statsmodels ARIMA Model Documentation* https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html.

135. Prophet. *Prophet Quick Start Guide: Python API* https://facebook.github.io/prophet/docs/quick_start.html#python-api.

136. Alexandrov, A. *et al.* GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research* **21,** 1–6. http://jmlr.org/papers/v21/19-820.html (2020).

137. Beitner, J. *PyTorch Forecasting Documentation* https://pytorch-forecasting.readthedocs.io/en/stable/#pytorch-forecasting-documentation.

138. Alomar, A. *Implementation of the algorithm described in: SAMoSSA: Multivariate Singular Spectrum Analysis with Stochastic Autoregressive Noise* https://github.com/AbdullahO/SAMoSSA.

139. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. *Optuna: A Next-Generation Hyperparameter Optimization Framework* in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 2623–2631.

140. Loshchilov, I. & Hutter, F. Fixing Weight Decay Regularization in Adam. *ArXiv* **abs/1711.05101.** https://api.semanticscholar.org/CorpusID:3312944 (2017).

141. Trindade, A. *ElectricityLoadDiagrams20112014* UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C58C86. 2015.

142. Abbasi, M. A., Chai, S.-K., Liu, H. & Sagoo, K. P. *Real-World Behavior Analysis through a Social Media Lens* in *Social Computing, Behavioral Modeling, and Prediction* (2012). https://api.semanticscholar.org/CorpusID:18828033.

143. Yuniar, D., Djakfar, L., Wicaksono, A. & Efendi, A. Truck Driver Behavior and Travel Time Effectiveness Using Smart GPS. *Civil Engineering Journal* **6,** 724–732 (Apr. 2020).

Appendices

A Analytical Solution to OLS Algorithm

Given a dataset with n observations and k features, we define the design matrix $\mathbf{X} \in \mathbb{R}^{n\times(k+1)}$ (including a column of ones for the intercept), the response vector $\mathbf{y} \in \mathbb{R}^n$, and the coefficient vector $\boldsymbol{\beta} \in \mathbb{R}^{(k+1)}$. The model predicts $\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$, and our goal is to find $\boldsymbol{\beta}$ that minimizes the sum of squared residuals, defined as:

$$S(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Expanding $S(\boldsymbol{\beta})$ yields:

$$S(\boldsymbol{\beta}) = \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

To find the minimum of $S(\boldsymbol{\beta})$, we take the derivative with respect to $\boldsymbol{\beta}$ and set it to zero:

$$\nabla_{\boldsymbol{\beta}} S(\boldsymbol{\beta}) = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

Setting the derivative equal to zero gives:

$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = 0 \Longrightarrow \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

Assuming $\mathbf{X}^T\mathbf{X}$ is invertible (which is true iff $\boldsymbol{X}$ is full rank), we multiply both sides by $(\mathbf{X}^T\mathbf{X})^{-1}$ to obtain:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This equation gives the least squares estimator for $\boldsymbol{\beta}$, which minimizes the sum of squared residuals between the observed values and the values predicted by the model.

B iTransofmer Pseudo-Code

Algorithm B.1 iTransformer - Overall Architecture

Require: Input lookback time series $\mathbf{X} \in \mathbb{R}^{T \times N}$, input Length T, predicted length S; variates number N; token dimension D; iTransformer block number L.

$\mathbf{X} = \mathbf{X}^{\top}$ ▷ $\mathbf{X} \in \mathbb{R}^{N \times T}$

Multi-layer Perceptron works on the last dimension to embed series into variate tokens.

$\mathbf{H}^0 = \mathrm{MLP}(\mathbf{X})$ ▷ $\mathbf{H}^0 \in \mathbb{R}^{N \times D}$

for l in $\{1, \ldots, L\}$ **do** ▷ Run through iTransformer blocks.

    Apply self-attention layer on variate tokens.

    $\mathbf{H}^{l-1} = \mathrm{LayerNorm}(\mathbf{H}^{l-1} + \text{Self-Attn}(\mathbf{H}^{l-1}))$ ▷ $\mathbf{H}^{l-1} \in \mathbb{R}^{N \times D}$

    Apply feed-forward network for series representations, broadcasting to each token.

    $\mathbf{H}^{l} = \mathrm{LayerNorm}(\mathbf{H}^{l-1} + \text{Feed-Forward}(\mathbf{H}^{l-1}))$ ▷ $\mathbf{H}^{l} \in \mathbb{R}^{N \times D}$

    LayerNorm is adopted on series representations to reduce variates discrepancies.

end for

$\hat{\mathbf{Y}} = \mathrm{MLP}(\mathbf{H}^{L})$ ▷ Project tokens back to predicted series,$\hat{\mathbf{Y}} \in \mathbb{R}^{S \times N}$

$\hat{\mathbf{Y}} = \hat{\mathbf{Y}}^{\top}$ ▷ $\hat{\mathbf{Y}} \in \mathbb{R}^{S \times N}$

return $\hat{\mathbf{Y}}$ ▷ Return the prediction result, $\hat{\mathbf{Y}}$

C Source Code

All project code is publicly available and can be accessed at:

https://github.com/giacovides/FYP.