Machine Learning for the Analysis of Power System Loads: Cyber-Attack Detection and Generation of Synthetic Datasets

by

Andrea Pinceti

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2021 by the
Graduate Supervisory Committee:

Lalitha Sankar, Chair
Oliver Kosut
Anamitra Pal
Yang Weng

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

As the field of machine learning increasingly provides real value to power system operations, the availability of rich measurement datasets has become crucial for the development of new applications and technologies. This dissertation focuses on the use of time-series load data for the design of novel data-driven algorithms. Loads are one of the main factors driving the behavior of a power system and they depend on external phenomena which are not captured by traditional simulation tools. Thus, accurate models that capture the fundamental characteristics of time-series load data are necessary.

In the first part of this dissertation, an example of successful application of machine learning algorithms that leverage load data is presented. Prior work has shown that power systems energy management systems are vulnerable to false data injection attacks against state estimation. Here, a data-driven approach for the detection and localization of such attacks is proposed. The detector uses historical data to learn the normal behavior of the loads in a system and subsequently identify if any of the real-time observed measurements are being manipulated by an attacker.

The second part of this work focuses on the design of generative models for time-series load data. Two separate techniques are used to learn load behaviors from real datasets and exploiting them to generate realistic synthetic data. The first approach is based on principal component analysis (PCA), which is used to extract common temporal patterns from real data. The second method leverages conditional generative adversarial networks (cGANs) and it overcomes the limitations of the PCA-based model while providing greater and more nuanced control on the generation of specific types of load profiles. Finally, these two classes of models are combined in a multi-resolution generative scheme which is capable of producing any amount of time-series load data at any sampling resolution, for lengths ranging from a few seconds to years.

DEDICATION

This dissertation is dedicated to my better half, Kirstie Swingle. You are a true inspiration and without your love and support I could not have done this.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisors Dr. Lalitha Sankar and Dr. Oliver Kosut for their guidance throughout my PhD career. With her constant support, wisdom, and encouragement, Dr. Sankar has truly been the best mentor anyone could ask for, both inside and outside the lab. Dr. Kosut was an invaluable source of knowledge and ideas that helped shape my work as a PhD student. I am deeply thankful for all they taught me.

I would also like to express my gratitude to my committee members, Dr. Anamitra Pal and Dr. Yang Weng who have provided valuable comments and suggestions.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

1.1 Background

For many decades, power system research has been based on the development of simplified physical models and the study of their interactions within a system. As a result, a large number of software tools now exists that allow to perform virtually any type of power system analysis. However, in recent years, the field of machine learning has matured to the point where it can provide real value to power system operations; for this reason, a large portion of research efforts now focuses on applying machine learning techniques to power system applications. This change in direction represents a major shift: moving away from physics-based device models to a data-centric analysis of system behaviors.

Within this new paradigm, the availability of large amounts of real data is crucial. Unfortunately, while power system models of all kinds are readily available, data is a much more scarce resource. The few researchers who have relationships with electric utilities can get access to real measurements through long processes involving non-disclosure agreements; in general though, the broader research community must rely on the very few and limited datasets that are publicly available.

The goal of the work presented in this dissertation is to develop mechanisms for the generation of synthetic time-series transmission-level load data. The focus on bus-level load data is motivated by the fact that loads are one of the main external drivers of power system behaviors; loads depend on phenomena outside of the power system itself (consumer behaviors, weather, etc.). Thus, realistic load profiles can be

used as an input to existing power system programs to accurately determine electric quantities such as voltages and currents via dynamic simulation. Moreover, while load data is used for virtually every power system study, the requirements in terms of sampling rate and time length vary based on the specific application. For this reason, a complete generative scheme is proposed which can produce realistic time-series data at any time resolution and for lengths ranging from a few seconds to years.

The effort aimed at addressing the lack of publicly available bus-level load data started during previous research on the cyber-security of electrical grids. Historical load data can be very effectively leveraged to design countermeasures against false data injection (FDI) attacks on state estimation (SE). Chapters 2 and 3 describe a scalable algorithm based on nearest neighbor for the detection and localization of FDI attacks. This work is presented here as an example of a successful data-driven application which relies on historical load data; it serves to highlight the importance of publishing large power system datasets and making them available to the research community for the development of machine learning applications.

In this dissertation, two distinct methods for the generation of synthetic time-series load data are proposed. Chapter 4 presents an algorithm which uses principal component analysis (PCA) to extract prototypical load patterns from a real dataset and use them to create new, realistic load profiles. The second method, described in Chapter 6, leverages a more advanced approach based on generative adversarial networks (GANs) which obviates some of the shortcomings of the PCA-based approach while also providing a more flexible model for the generation of load profiles. As explained in Chapter 7, these two methods are combined into a multi-resolution generative scheme which allows for the creation of large synthetic datasets at any sampling resolution. The training of this comprehensive generative model is based on a large database of real PMU data (about 70 terabytes) from a US electric utility;

Chapter 5 describes the design and practical implementation of the database structure and its applications.

1.2 Outline of Thesis

The following Dissertation is organized in six main chapters, as follows:

- Chapter 2 presents three distinct data-driven algorithms for the detection of cyberattacks.

  In particular, Sections 2.1 and 2.2 introduce the problem of FDI attacks on SE and related work; Section 2.3 presents the system and attack models used while Section 2.4 describes the three proposed detection algorithms. Finally, Section 2.5 presents experimental results to illustrate the detection performance of the attack detectors.

- Chapter 3 describes how the detection algorithm is improved to be effective on large scale systems.

  Sections 3.1 and 3.2 summarize our contribution and the related work, Sections 3.3 and 3.4 describe the attack model and the proposed detector, Section 3.5 shows the experimental results of the detection algorithm, and Section 3.6 presents a framework for the localization of FDI attacks.

- Chapter 4 describes the data-driven generative model based on PCA.

  Section 4.1 presents related work, Section 4.2 describes the dataset used, Section 4.3 and Section 4.4 detail the generative model, and Section 4.5 shows the results. Section 4.6 illustrates how the proposed approach can be adapted to be used on datasets of PMU load data.

- Chapter 5 describes the design of a large database of PMU data which represents the foundation for the advanced generative models proposed.

  Section 5.1 introduces the problem of PMU data storage and visualization; Section 5.2 describes the data-structure we developed and Section 5.3 presents some study cases to illustrate the effectiveness of the proposed solution.

- Chapter 6 describes the improved generative model based on GANs.

  Section 6.1 introduces the GAN-based approach, Section 6.2 provides a theoretical description of GANs, Sections 6.3 and 6.4 describe the dataset used, Section 6.5 illustrates the proposed GAN architecture, and Section 6.6 describes the several tests used to verify the quality of the generated data.

- Chapter 7 describes the complete multi-resolution generative scheme.

  Section 7.1 highlights the need for a comprehensive generative model; Section 7.2 presents an overview of the generative scheme proposed; Section 7.3 details the different models adopted; Section 7.4 describes how the generative models can be used together to create any type of load; Section 7.5 shows the open-source application developed and Section 7.6 and 7.7 present some use cases and validation of the generative scheme.

Chapter 2

ALGORITHMS FOR THE DETECTION OF CYBER-ATTACKS

2.1 Background

The electrical grid is a constantly evolving cyber-physical system and, as such, it is increasingly reliant on information and communication technology. A vast research effort undertaken in the past decade in the field of cybersecurity of power systems has identified some crucial vulnerabilities of the cyber layer which can be exploited to disrupt the physical system. In this context, [1] shows that state estimation (SE) and the traditional bad data detector (BDD) used in energy management systems (EMSs) can be easily spoofed and bypassed via false data injection (FDI) attacks. This finding represents the basis for the design of a wide class of attacks called load redistribution (LR) attacks. Load redistribution attacks can be performed by injecting intelligently designed false measurements that lead to a wrong estimate of the system state. From the operator's perspective the attack makes it appear as if the system loads have changed from their actual values, without changing the net load.

In [2] and [3], the concept of load redistribution attacks is formalized by developing a bi-level attacker-defender problem for targeted attacks. In this setting, the attacker can design false measurements which can cause physical consequences on the system; specifically, [3] attempts to find an attack, unobservable to the EMS, to cause a power overload on a target line. In a similar fashion, [4, 5] present examples of LR attacks on the electricity market: the authors show that it is possible to launch load redistribution attacks that create system congestion, thus manipulating locational marginal prices.

2.2 Related work

The proposed approach to the detection of load redistribution attacks exploits the fact that system operators have access to large amounts of historical load data. This data can be used to learn patterns in an offline manner and design machine learning algorithms that can check load consistency in real time scenarios. We use *nearest neighbor algorithm*, *support vector machines* and *replicator neural networks* to design three different detectors which are tested on realistic historical load data created for the IEEE 30-bus system. To design a detector that works for any possible load pattern that might occur, we exploit publicly available PJM zonal data [6] to create realistic load data for multiple years for the IEEE 30-bus system.

Other attempts have been made to design detection algorithms and countermeasures against cyber-attacks. In [7] for example, sparse optimization is leveraged to detect corrupted measurements while in [8] the detection is performed by using phasor meaurements. In these studies, SE is used as a tool for identifying attacked measurements without any considerations on the system state they represent. Our work differs in that by applying machine learning on historical data we are able to check for consistency of the observed load data and judge if it corresponds to a realistic system configuration. In [9], machine learning techniques are applied to SE for anomaly detection but the models used to test the detection algorithms are not representative of the behavior of a real system because only the base case operating point is considered. The training data used by the authors represents measurements of the same system configuration and differs only for the random noise that is added. This makes it possible to find very efficient detection schemes, which are not guaranteed to perform as well when the system loads are different from the base case.

2.3 System model

2.3.1 Load data: model and design

Throughout our analysis of the proposed detectors we use the IEEE 30-bus system and real load data published by PJM. On its website, PJM regularly provides the hourly loads of the 20 zones of the northeastern US grid. This historical data is available starting from the year 1993 and it is updated monthly with the most recent figures. We leverage the fact that the 30-bus system contains 20 load buses to create a test case with realistic load profiles, by mapping each PJM zonal load to a corresponding load in the IEEE system. The zonal loads and the loads in the IEEE 30-bus system are ranked by relative size compared to the total load size of their respective system, as shown in Table 2.1. The percentage distribution of the loads of the PJM system is very close to that of the loads in the 30-bus sytem, which justifies the mapping from each zone to the similarly sized load in our test system. The mapping has been performed by first determining a mapping ratio between the PJM and the 30 bus system and then multiplying every hourly load of the 20 zones by this factor. In this way, we created new load configurations corresponding to every hour of the year for our test system. Creating load configurations in which many lines are congested or at their rated limits maximizes the chances of finding successful attacks since a congested system often represents a worst-case scenario in terms of vulnerability to cyber-attacks and possible physical consequences. Thus, we defined the mapping ratio as the ratio between the net load of the 30-bus system base case and the maximum net load of the PJM system:

$$\begin{aligned} m_{\text{ratio}} &= \frac{\text{30 bus net load}}{\text{max PJM net load}} \times k = \\ &= \frac{189.2\,MW}{144644\,MW} \times 1.39 = 1.308 \times 10^{-3} \end{aligned} \tag{2.1}$$

where k is a constant that was chosen in order to obtain a system with moderate amount of congestion.

We used the PJM data to create a load dataset for the IEEE 30 bus system for five years, corresponding to the period from January 2012 to December 2016 which we assumed to be our *normal* data. In the detection experiments, the first four years are used as historical data to train the detectors and the last year is used as the test data. Thus, the training data contains 35064 distinct load profiles (one for each hour), while the test data comprises 8784 load profiles.

2.3.2 Attack model and design

For a DC-SE model, the measurements vector z can be written as

$$z = Hx + e \tag{2.2}$$

where x is the vector of bus voltage angles, H is the matrix describing the relationship between states and measurements, and e is the vector of random noise affecting the measurements. As described in [2], an attack is unobservable when a corrupted measurement $\bar{z}_i$ satisfies

$$\bar{z}_{\mathrm{i}} = z_{\mathrm{i}} + H_{\mathrm{i}}c \tag{2.3}$$

where z_i is the original measurement, c is the arbitrary error created on the corresponding state and H_i is the i^{th} row of the H matrix. In [10] the authors formulate a bi-level optimization problem to find the attack vector c that maximizes the flow on a specific target line by redistributing loads; [11] improves on this formulation making it more efficient and scalable to large scale systems. The first level of the optimization problem models the attacker's actions: the objective is to maximize the flow on the target line, subject to constraints on the number of states that can be modified and the magnitude of change in the loads. This last parameter, defined as load shift (LS),

Table 2.1: Relative size of PJM zones and 30-bus system loads

PJM zone		**30-bus system load**	
Zone name	**Size [%]**	**Bus location**	**size [%]**
AEP	14.5	**8**	15.9
CE	13.7	**7**	12.1
DOM	12.4	**2**	11.5
ATSI	8.25	**21**	9.25
PS	6.34	**12**	5.92
AP	5.59	**30**	5.60
PE	5.41	**19**	5.02
PL	4.51	**17**	4.76
BC	4.27	**24**	4.60
PEP	4.08	**15**	4.33
JC	3.85	**4**	4.02
DEOK	3.35	**14**	3.28
DPL	2.60	**10**	3.07
DAY	2.14	**16**	1.85
ME	1.89	**26**	1.85
PN	1.88	**18**	1.69
DUQ	1.81	**23**	1.69
AE	1.73	**3**	1.27
EKPC	1.46	**29**	1.27
RECO	0.26	**20**	1.16

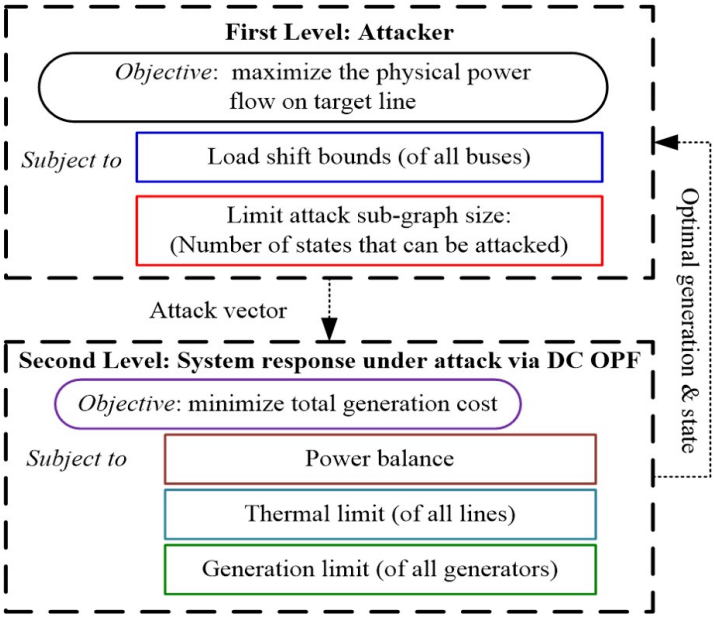

Figure 2.1: Bi-level attacker-defender optimization problem. Figure courtesy of Mr. Zhigang Chu, ASU.

is the maximum percentage by which a load can be changed by the attacker. This parameter is critical in computing an FDI attack: too small of a change will not cause any harmful consequences on the system, too large and the control center can easily the detect the load pattern as fraudulent. In our experiments we tested load shifts of 10% and 15%, as those values represent the best balance between undetectability and attack success. The second level of the optimization problem is a DCOPF that models the response of the system to the designed attack vector. Figure 2.1 shows a visualization of the bi-level optimization problem.

After computing the attack vector, the set of false measurements is passed to the state estimator which calculates the corresponding loads. It is this set of attacked loads that need to be flagged by the proposed detectors as malicious. We use this attack strategy to compute attacks on every hour of the 2016 synthetic data during which one or more lines were congested. A DCOPF is run on each of the hourly load profiles to measure the level of congestion in the system. Every time a solution

showed a line at or above 85% of its rating, an attack was computed. We define an attack as successful if it leads to a flow on the target line greater than 100% of its rating. The results of this process are summarized below:

DCOPF results

- 1197 hours out of 8784 total hours with at least one line above 85% rating;
- 450 hours out of 1197 hours with at least one line at 100%.

Successful attacks on congested hours

- LS = 10% : 437 successful attacks;
- LS = 15% : 479 successful attacks.

As shown by these results, depending on the specific load configuration at each hour, the search for an attack is not always successful. Moreover, the greater the allowable load shift, the higher the chances of computing a successful attack.

2.4 Detection algorithms design

We propose three detectors which analyze the measured loads of a power system during any hour of the day and determine if they are normal loads or if they have been maliciously modified through a cyber-attack. Each detector is based on a different machine learning technique, but the general approach in determining the soundness of a set of measurements is similar. The measured load configuration to be tested is given as input to the detector, which generates a scalar value based on a metric specific to the machine learning technique used. This value is compared against a predetermined threshold to label the loads as *normal* or *attacked*. The specific value of the threshold is chosen as a tradeoff between detection probability and false alarm rate.

2.4.1 Nearest neighbor algorithm

Nearest neighbor algorithms are based on the assumption that data labeled as normal lies in limited, dense regions of space while anomalies are located further from these neighborhoods [12, 13]. Define $\boldsymbol{l_i}$ as the $1 \times n_l$ vector of loads at time i, where n_l is the number of loads in the system. The normal data is represented by the n_h historical load vectors which have been measured in the past and we indicate as $\boldsymbol{h_j}$, for $j = [1 : n_h]$. The classification is done by measuring the Euclidean distance between the current load profile $\boldsymbol{l_i}$ and every vector $\boldsymbol{h_j}$ in the historical data set (assumed to be attack free). The closest distance d_i for a sample $\boldsymbol{l_i}$ is defined as

$$d_{\mathrm{i}} = \min_{j=[1:n_h]} ||\boldsymbol{l_i} - \boldsymbol{h_j}||_2. \tag{2.4}$$

To label $\boldsymbol{l_i}$ as normal or attacked, d_i is compared against the predetermined threshold.

2.4.2 Support vector machine

The second machine learning technique we tested is a support vector machine (SVM), in which data is used to define a boundary of the region of space in which all the normal points lie [13]. The training of an SVM results in identifying a hyperplane which includes all of the normal training data, and none of the abnormal training data. For non-linear classification, the training phase uses kernels to map the data to high-dimensional spaces making it possible to learn complex regions. Training of SVMs can be supervised or semi-supervised: in the former, both *normal* and *abnormal* data is used for learning, while in the latter only *normal* instances are considered.

In our tests, the data points used for training are the historical load vectors $\boldsymbol{h_j}$ from 2012 to 2015. All historical points are labeled as 1, indicating that they are normal loads, making this a semi-supervised machine learning algorithm. Preliminary testing showed that a linear hyperplane is not effective in classifying the load data, so

instead we use a Gaussian radial basis kernel function to create a complex, non-linear boundary [14].

The testing phase consists in feeding the normal load vectors for 2016 and the attacked load vectors to the SVM; for each load instance, the SVM computes a score between [-1,1]. The closer a score is to +1, the higher the confidence that the loads are normal; vice versa, scores close to -1 indicate that the loads have likely been maliciously modified by an attacker. These scores are the metric which is compared against a threshold to label the observed loads of the system as normal or attacked.

2.4.3 Replicator neural network

Replicator neural networks are a particular type of neural networks which are trained to compress and then reconstruct the data that is fed to them. For this reason, replicator neural networks have the same number of output neurons as the number of inputs. The neural network topology typically used includes three hidden layers with an equal number of hidden nodes each. The training phase aims at creating connections that are able to compress the input data (which usually has higher dimensionality than the number of nodes in each hidden layer) and then reconstruct it at the output nodes minimizing the error between input and output. The anomaly detection is performed by feeding to the trained network the data to be tested and measuring the discrepancy between input and output. For each load vector $\boldsymbol{l_i}$ at time i the error is computed as

$$\delta_{\mathrm{i}} = ||\mathbf{l_i} - \mathbf{o_i}||_2 \tag{2.5}$$

where $\boldsymbol{o_i}$ is the output load vector of the neural network. Comparing the replication error with a set threshold allows for the labeling of the data point as normal or attacked.

Intuitively, this neural network is trained to learn a model of the correlation

between the different loads in the system. Real loads will have a close correspondence to the model learnt and they will be reproduced with a small error, while a sample with loads that have been maliciously modified will yield bigger replication errors. In this work we used a neural network made of three hidden layers, each consisting of 15 nodes. The nodes in layers 1 and 3 have a sigmoid activation function, while the nodes in layer 2 have a linear activation function. We also tested a similar neural network, with all layers having sigmoid activation functions but it did not perform as well.

The data used for training is the load profiles from 2012 to 2015. In the training process, each sample is fed to the network and the weights of the internal connections are adjusted to minimize the difference between input and output. This process is perfomed through backpropagation of the error and it is solved using Levenberg-Marquardt optimization with mean squared error as performance function [15, 16].

2.5 Experiments

2.5.1 Experimental methodology

The testing methodology we follow consists of two steps and is the same for each detector. First, we compute the detector specific metric (minimum distance, score, or replication error) for each load configuration in the test data of 2016 and then we evaluate the performance in terms of *missed detection (M_D)* and *false alarm rate (F_A)*. Missed detection is defined as the ratio of the number of attacked cases flagged as normal to the total number of attacked cases, while false alarm rate is the ratio of the number of cases in which normal data is flagged as attacked to the total number of normal cases (8784 hours). These two metrics are evaluated for the dataset across a range of detection thresholds, thereby characterizing the tradeoff between missed

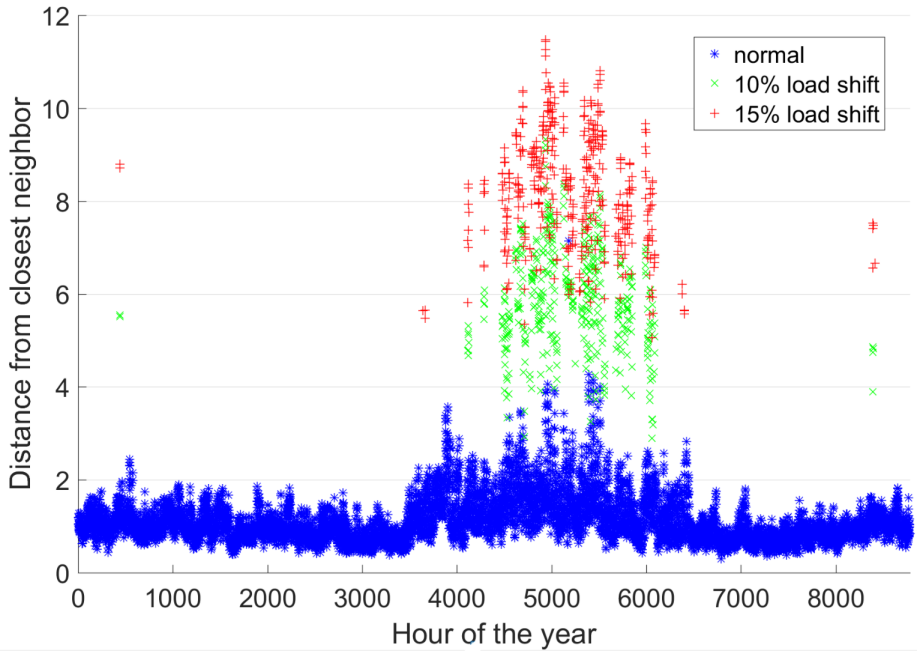

Figure 2.2: Distribution of nearest neighbor distance for normal and attacked cases.

detection and false alarm rate. These results are used to plot the receiver operating characteristic (ROC).

2.5.2 Experimental results

For the nearest neighbor algorithm, the minimum distance of every normal and attacked case from the closest sample in the training dataset is presented in Fig. 3.1. For each hour of the year (x-axis) the minimum distance is plotted on the y-axis: the blue asterisks correspond to normal loads, while the green cross signs and red plus signs represent the attacked cases, with 10% and 15% load shift respectively. Similarly, Fig. 2.3 shows the scores computed using the SVM and Fig. 2.4 shows the replication error of the neural network based detector. The performance of the detectors with 10% LS attacks is evaluated by plotting the ROC in Fig. 2.5. In the case of 15% LS, detecting the attacks becomes trivial, as shown in Table 4.1. For each detector, the values of missed detection and false alarm rate are shown for

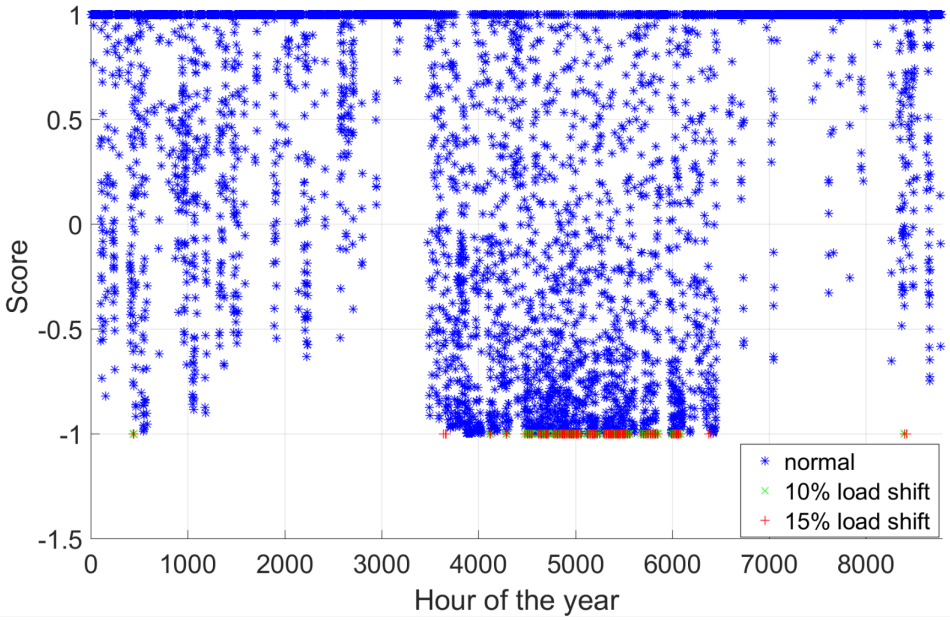

Figure 2.3: Distribution of SVM scores for normal and attacked cases. Note that for SVM a higher score (closer to 1) represents a belief that the data is normal, whereas a lower score (closer to -1) represents attacked data.

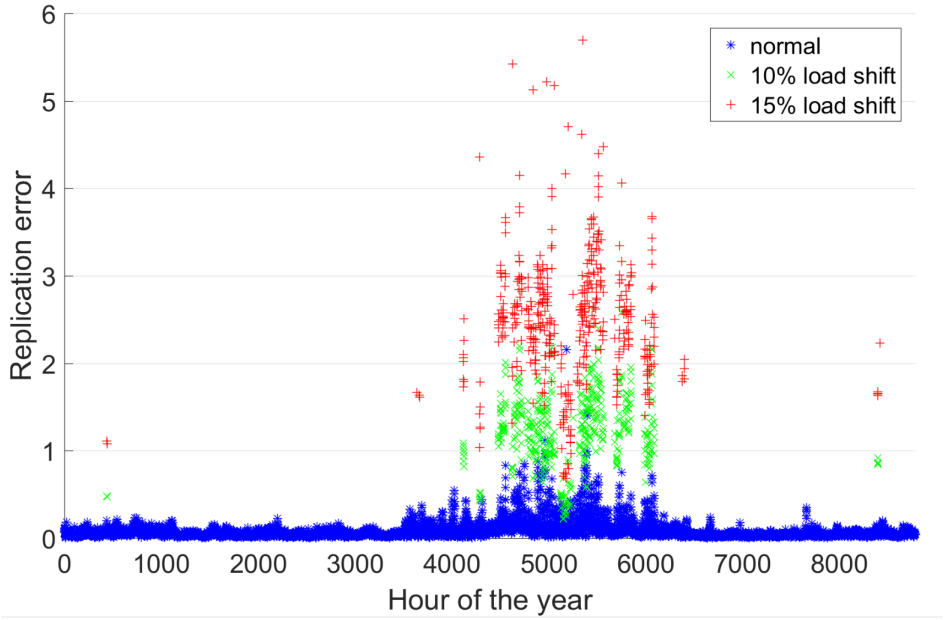

Figure 2.4: Distribution of replication error from the replicator neural network detector for normal and attacked cases.

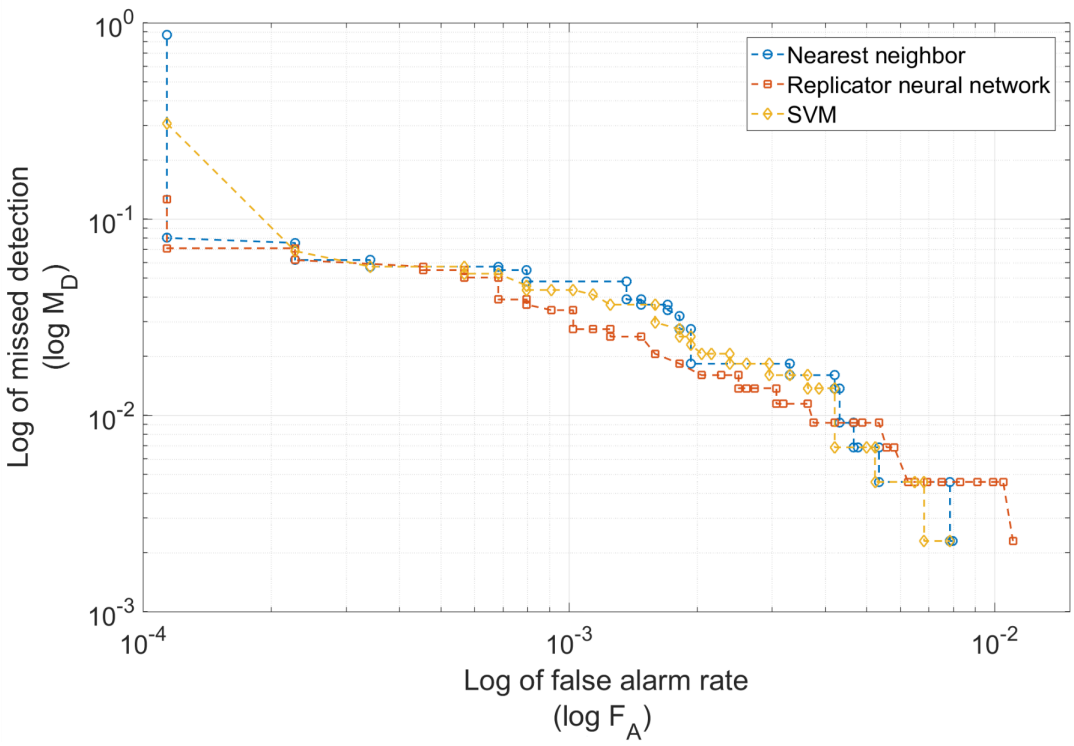

Figure 2.5: Receiver operating characteristic of the three detectors with 10% load shift attacks.

two different detection thresholds highlighting the near perfect performance of the detectors.

Overall the results of the three detectors are very similar. An important difference in the case of the neural network-based detectors is the computational complexity. The training phase for our neural network required almost 6 hours to complete using the Matlab machine learning toolbox, while the training of the SVM on the same data took less than 1 minute. The nearest neighbor algorithm does not necessitate a training phase, but unlike the other two methods all the computing is done in real time by searching for the minimum distance. This search requires the calculation of the distance from every entry in the historical data, but in our tests this operation only takes a fraction of a second. In terms of applications to real time detection, after performing the training of SVM and replicator neural network offline, all three

Table 2.2: Performance of the detectors with 15% LS attacks

Detector	M_D	F_A	*# of false alarms*
Nearest neighbor	0.2192	0	0
	0	1.138×10^{-4}	1
SVM	0.0021	1.138×10^{-4}	1
	0	2.277×10^{-4}	2
Replicator	0.0125	0	0
Neural Network	0	1.138×10^{-4}	1

detectors are able to analyze the loads much faster than the sampling rate of modern SCADA systems.

Chapter 3

ATTACK DETECTION ON LARGE SCALE SYSTEMS

3.1 Our contribution

We propose a new bad data detector that can identify LR attacks based on the analysis of load estimates, thus overcoming the limitations of SE and the traditional BDD. In [17], we developed three anomaly detectors, each based on a different machine learning technique: replicator neural network, support vector machine, and nearest neighbor. These detectors can effectively determine if the observed loads represent a normative system state or if they have been maliciously modified. The nearest neighbor-based detector works by finding in the historical data the closest load vector to the real time loads and, based on the measured Euclidean distance, a thresholding technique is used to decide if the loads are normative or anomalous. From our tests, nearest neighbor demonstrated the best performance out of the three proposed. In this chapter, we build on this preliminary work to design an improved detector and an attack localization scheme.The novel contributions of this work are as follows:

- The basic detector is modified so that *it scales to much larger power system models* while preserving the good detection performance shown in [17]. This is achieved by devising a grouping strategy to organize the system loads into clusters that can be analyzed independently.

- *Extensive testing and sensitivity analysis is performed to evaluate the performance of the detector* against intelligently designed LR attacks as well as random anomalous load changes. This allows for the characterization of the strengths

and limitations of the detector. Furthermore, *the proposed algorithm is integrated within a complete EMS platform* to showcase its detection performance and computational efficiency.

- Building on the improved detector, *a statistical approach is presented to localize the attacks and determine the likelihood of each load of being attacked.* The deviation in loads is captured via a Z-score and log-loss is used as a measure to find the likelihood function that minimizes the error. This represents a crucial step towards the development of decision tools that can help operators to securely manage power systems when targeted by cyber-attacks.

3.2 Related work

In addition to the related work presented in the previous chapter, further examples of FDI and LR attack detectors can be found in the literature. To highlight the novel contributions of the detector presented in this chapter, we list more related work and explain how our method overcomes their limitations. For example, in [18], multiple linear regression is used to study the voltage profiles in a system and determine if a LR attack is taking place. Unlike our work, the method there proposed is designed for distribution systems and, as we will explain later, the attacks tested are not realistic as they simulate changes in loads of up to 100%. Other work focuses on using deep neural networks to learn the temporal correlation which exists between the real-time measurements and previous samples [19], or verifying the statistical behavior of the estimated states over time [20]. The assumption on which these detectors are built is that when an attack is injected, the false measurements are not compatible with the dynamics observed from the previous measurements thus making it possible to flag them as attacked. Based on this, a slow ramping attack which only slightly changes

the system's state at each sampling time will likely not be detected. Moreover, these detectors are tested on limited attack scenarios and their performance is not verified against multiple classes of attacks. Finally, while many attack detectors have been proposed, to the best of the authors' knowledge, the idea of detecting FDI attacks by identifying patterns in the observed loads has not been explored before.

The rest of the chapter is organized as follows: in Section 3.3 we summarize the basic detection algorithm we have presented in [17] and show its performance limitations when used on large scale systems. The required improvements are described in Section 3.4 and the detection results on a wide range of load redistribution attacks are presented in Section 3.5. Section 3.6 illustrates the statistical analysis that leverages the improved nearest neighbor-based detector to determine the buses that have been attacked.

3.3 Basic detection algorithm

3.3.1 Small systems

The proposed attack detection mechanism works by analyzing the correlation structure within the currently observed load values and comparing it to attack-free historical load data. The measured load configuration to be tested is given as input to the detector which generates a scalar value. This value is then compared against a threshold τ to label the loads as *normative* or *attacked*. To evaluate the detection performance, two metrics are used: *detection probability*, which is the ratio between the number of cases correctly labelled as attacked and the total number of attacked cases tested, and *false alarm rate*, which is the number of normative cases that are labelled as attacked divided by the total number of normative cases tested. The specific value of the threshold is chosen as a tradeoff between detection probability

and false alarm rate. Our approach can be considered a semi-supervised learning problem since the detectors are trained only on normative data which is already widely available to operators. Because no attacked data is needed in the training phase, the detectors will not be biased towards specific types of attacks. Given the almost identical detection capability of the three detectors tested in the previous chapter [17], in the following work, the nearest neighbor detector is chosen for its computational and explanatory simplicity.

Nearest neighbor algorithms are based on the assumption that data labelled as normative lies in limited, dense regions of space while anomalies are located further from these neighborhoods [21, 22]. Let us define $\boldsymbol{p} \in \mathbb{R}^n$ as the vector of observed load values to be tested, where n is the number of loads in the system. The normative data is represented by the set $\boldsymbol{P}_N^{\text{hist}} \in \mathbb{R}^{n \times n_h}$ of historical load vectors $\boldsymbol{h}_i \in \mathbb{R}^n$ that have been observed in the past, where n_h is the total number of historical vectors. The classification is done by measuring the Euclidean distance between the current load profile $\boldsymbol{p}$ and every vector $\boldsymbol{h}_i$ in the historical dataset (assumed to be attack-free). The closest distance d for sample $\boldsymbol{p}$ is defined as

$$d = \min_{j=[1:n_h]} \|\boldsymbol{p} - \boldsymbol{h}_j\|_2. \tag{3.1}$$

To label $\boldsymbol{p}$ as normative or attacked, d is compared against a predetermined threshold τ.

In the previous chapter [17], we tested this approach on the IEEE 30 bus system. Publicly available zonal historical load data from the PJM system [23] was mapped to the loads of the 30 bus system to create hourly load profiles for 5 consecutive years. The proposed detector showed very high detection capability with low false alarm rates. Figure 3.1 is taken from [17] and it shows some of the results obtained on this small system. The blue points represent the minimum distance for the normative

load vectors (not attacked) while in green and red are the distances corresponding to attacked cases with 10% and 15% load shift, respectively. This illustrates how loads resulting from attacks lead to much higher nearest neighbor distances compared to normative load profiles; thus, suggesting that the minimum distance is an effective metric for attack detection.

3.3.2 Large systems

While the results obtained on the 30 bus system are promising, the detector needs to be tested on large scale systems to verify its performance in a more realistic setting and to guarantee its suitability for implementation in real system operations. To this end, the same analysis presented in [17] and summarized in the previous section is performed on the synthetic Texas system [24, 25]. This model, developed at Texas A&M, is a synthetic grid of the state of Texas. It has 2000 buses, 3206 branches, and 1125 loads and it includes bus-level hourly load data for the year 2016. Using the attack model described in the previous Chapter, around 280 attacks with load shift of 15% have been designed on the most congested cases. We randomly selected 90% of the 8784 normative load vectors to represent the historical data, and the remaining 10% for testing. The nearest neighbor algorithm is used to compute the minimum distance for the test and the attacked load vectors against the historical load data. Figure 3.2 shows the minimum distance for each normative load vector (blue points) and for the attacked cases (red points). It is easy to see that the detector does not perform well, and that the attacked cases are indistinguishable from the normative ones. This can be explained by the fact that when measuring the Euclidean distance between two high-dimensional vectors, the contribution of a limited subset of dimensions is small. That is, if only a few tens of loads are attacked, the total distance measured over hundreds of loads will deviate only slightly from

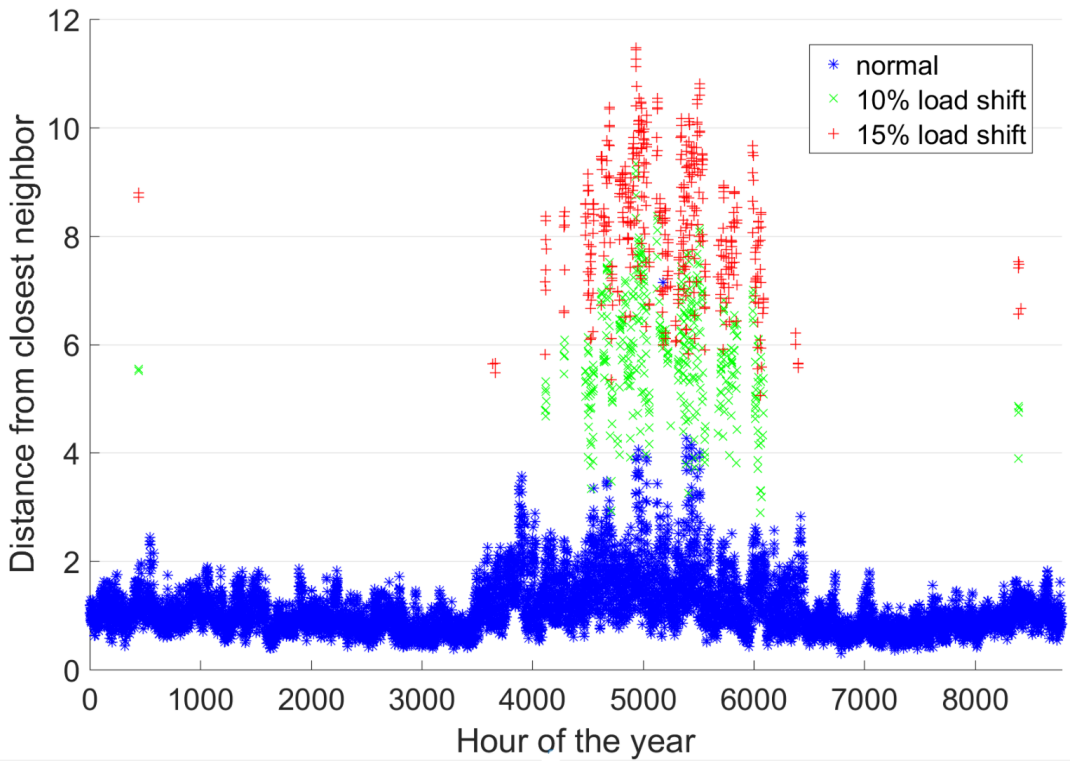

Figure 3.1: IEEE 30 bus system: distribution of nearest neighbor distance for normative and attacked cases.

the distance computed on the load vector where no loads are modified. In this case, each load vector has dimension 1125 and the attacks modify only about 100 to 200 loads; the effect of the attacked loads is not large enough to result in distance values significantly higher than those of the normative data.

3.4 Detection on large scale systems

The simple test presented in the previous section shows that the basic nearest neighbor detector introduced in [17] does not perform as well when applied to large scale systems (hundreds or thousands of buses). For this reason we need a new approach to improve the detection mechanism to be effective for any system, regardless of its size. The method we propose aims to leverage the capability of the nearest neighbor algorithm to identify anomalous loads even when only a small fraction of

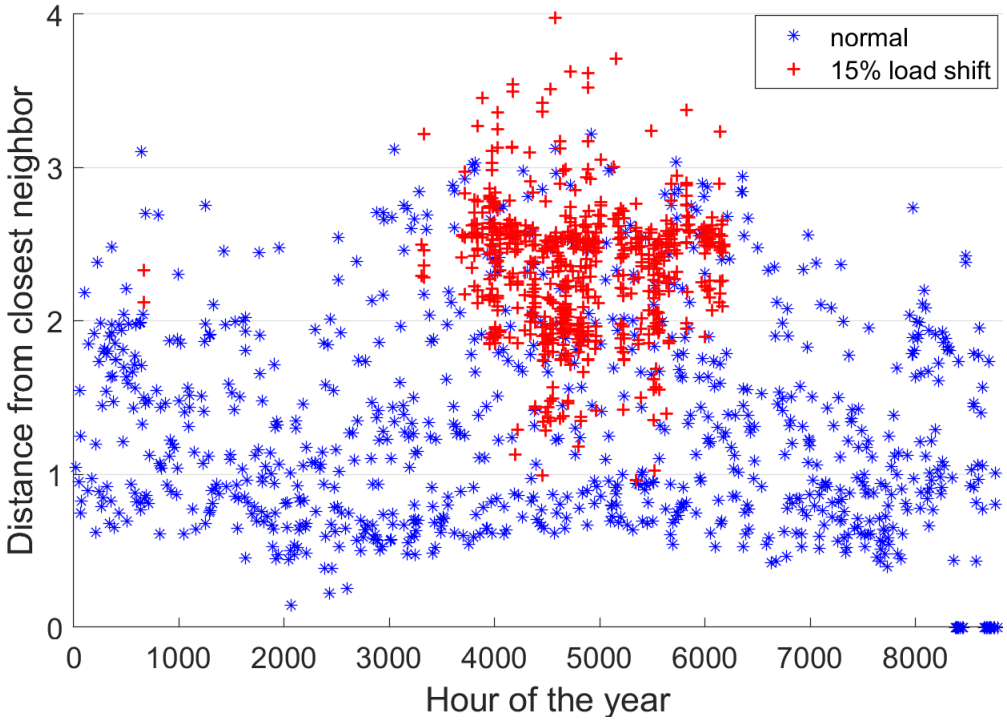

Figure 3.2: Synthetic Texas system: distribution of nearest neighbor distance for normative and attacked cases.

the total system loads are being attacked.

Previous work has shown that in a large transmission-level system, load redistribution attacks tend to target only some portions of the network. As a consequence, the loads which are modified represent a subset of the total system loads and they are restricted to a subgraph of limited size. Based on these observations, the detection algorithm is modified so that it analyzes multiple predefined subsets of the system loads. In this work we propose a grouping strategy that can be used to divide the loads into relatively small groups such that they can be analyzed independently and in parallel by the attack detector. It is important to notice that the strategy presented in the next section is just one example of grouping which empirically worked well for the systems tested. Different grouping techniques, perhaps leveraging specific knowledge and insights regarding the power system to be secured, can be easily implemented within the framework of the proposed detection algorithm.

3.4.1 Grouping strategy

The first step required to define the load groups is to sort the loads based on their MW rating, from largest to smallest. Starting from the largest load, the first group is created by including the load itself and all its neighboring loads within a certain radius r_g, where the radius is measured as the smallest number of branches connecting two loads. At this point, the next largest load is selected and if it is not contained in any of the previous groups, a new group is created. This process is repeated until n_g groups are defined. Note that it is possible for a bus to be contained in multiple groups, or no groups. The parameters r_g and n_g have a direct effect on the detection performance and their selection will be discussed in the next sections. As our results show, this grouping strategy proves to be very effective in the detection of LR attacks because it ensures that the largest loads in a system are monitored. Prior work on FDI attacks on SE shows that, to cause significant consequences, an attacker is required to target large loads in order to create large power flow changes [2, 3, 4, 5].

3.4.2 Detection algorithm

Dividing the n system loads into groups allows us to overcome the dimensionality issue observed in Section 3.3.2. The basic nearest neighbor detector can be used on large systems by running the algorithm individually on each load group. In this case, a threshold τ_j must be defined for each individual group g_j, for $j \in [1 : n_g]$. The vector $\boldsymbol{p} \in \mathbb{R}^n$ containing the estimated loads computed by SE is divided into n_g groups according to the procedure described in the previous subsection. Let us define $\boldsymbol{p}^j$ as the vector containing the real-time values of the loads in group g_j. For each group, the minimum distance between the load vector $\boldsymbol{p}^j$ and the corresponding loads

in the historical dataset is calculated as

$$d_j = \min_{r=[1:n_h]} \|\boldsymbol{p}^j - \boldsymbol{h}_r^j\|_2 \tag{3.2}$$

where $\boldsymbol{h}_r^j$ is the subset of loads belonging to group g_j from the r^{th} historical load vector. The minimum distance is then compared to the threshold τ_j to determine if the loads in group g_j are normative or anomalous. Specifically, if $d_j > \tau_j$, an alarm is raised, while if $d_j < \tau_j$ the loads are considered attack-free. This process is repeated for every group and if one or more alarms are raised, the load vector p is labelled as anomalous.

3.5 Testing the improved detector

3.5.1 Experimental procedure

The performance of the nearest neighbor-based detector in conjunction with the grouping strategy is tested in depth in the following sections. The detection capability is measured both on intelligently designed attacks as well as random load redistribution attacks; moreover, we study its sensitivity to different parameters, such as the load shift of the attack and the number of attacked buses.

The goal of the following experiments is to analyze the quality of the detector at understanding if a load vector is normative or attacked. The primary test system used is the synthetic Texas system described in Section 3.3.2; all numerical results discussed below are based on this system. Additional testing performed on the 2383 bus Polish test case [26], for which we generated historical load profiles based on real data from a major US ISO [27], showed comparable results and it is here omitted due to space constraints.

First, the 1125 system loads in the Texas system are divided into groups following the procedure from Section 3.4.1. For the tests described below, the parameters

chosen for the creation of the groups are: radius $r_g = 7$ and number of groups $n_g = 35$. These values ensure that more than 60% of the loads in the system are included in one or more groups and the ones that are outside of the groups have at least one monitored neighboring load. Moreover, these load groups are equally spread across the system; as a result, the system is effectively monitored in its entirety. Preliminary testing has shown that increasing the number of groups (and, thus, of the loads considered) did not improve detection performance.

In each experiment, two datasets are needed: the normative load data $\boldsymbol{P}_N \in \mathbb{R}^{1125\times 8784}$ and the anomalous load data $\boldsymbol{P}_A \in \mathbb{R}^{1125\times H}$, where H varies for the different type of attacks. The normative data represents one load vector for each hour of 2016 (2016 was a leap year). The set $\boldsymbol{P}_A$ contains attacked load vectors which are designed starting from the normative load vectors in $\boldsymbol{P}_N$; depending on the type of attack, some of the loads are modified either intelligently or randomly, as described below.

To compute detection probability and false alarm, the load vectors of dataset $\boldsymbol{P}_N$ are first divided into three subsets: historical, training, and testing. The historical dataset $\boldsymbol{P}_N^{\text{hist}}$ includes 70% of the total hours of 2016 and it represents the past loads known to the system operator and used in its nearest neighbor algorithms. The training dataset $\boldsymbol{P}_N^{\text{train}}$ represents another 20% of $\boldsymbol{P}_N$ and it is needed to determine the thresholds τ_j for each load group. The remaining 10% of normative load vectors is used as the testing dataset $\boldsymbol{P}_N^{\text{test}}$ to determine the false alarm rate. To determine the threshold τ_j for group g_j, the minimum distance $d_{i,j}$ between each load vector $\boldsymbol{p}_i^j$ for i in $\boldsymbol{P}_N^{\text{train}}$ and the historical dataset is computed using (3.2). The threshold τ_j is defined as a fixed fraction of the maximum closest distance, defined for each group as

$$d_{\max,j} = \max_{p_i \in \boldsymbol{P}_N^{\text{train}}} d_{i,j}. \tag{3.3}$$

For each load vector in $\boldsymbol{P}_N^{\text{test}}$, the minimum distance from $\boldsymbol{P}_N^{\text{hist}}$ is computed and

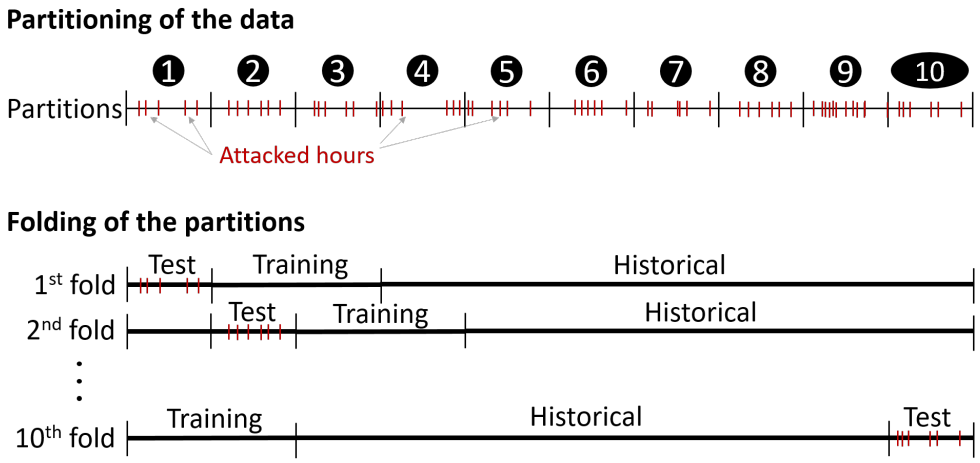

Figure 3.3: Description of the 10-folding technique and definition of the datasets

compared with the threshold: the false alarm rate is the ratio between the number of times a load vector is labelled as attacked (e.g. at least one load group has minimum distance greater than its corresponding threshold) and the total number of load vectors in $\boldsymbol{P}_N^{\text{test}}$. Similarly, the minimum distance is calculated for every attacked case and the detection probability is computed. As we will explain in more detail in the next sections, varying the threshold about the value $d_{\max,j}$ allows to span different detection probabilities and false alarm rates in order to determine the receiver operating characteristic (ROC). The proposed algorithm is extremely efficient and testing a load vector only takes a fraction of a second on a normal laptop; thus, even on large power systems, the detector can easily run in real-time.

Because the normative load dataset is limited to one year, in order to have a more complete assessment of the performance of the detector, a k-folding technique is used to test every hour of the year by rotating through multiple sets of historical, training, and testing datasets. The hours of 2016 are randomly divided into ten equally sized partitions as illustrated in Fig. 3.3; the partitions are fixed throughout the testing process. For the first fold, the load vectors corresponding to the hours in the first partition are assigned to $\boldsymbol{P}_N^{\text{test}}$, the second and third partitions to $\boldsymbol{P}_N^{\text{train}}$

and the remaining seven represent the historical data $\boldsymbol{P}_N^{\text{hist}}$. Given these partitions, the number of false alarms and the number of attacks detected are calculated on the normative and attacked load vectors in the testing partition. The subsequent folds are created by shifting the partitions assigned to the three datasets by one: for example, in the second fold $\boldsymbol{P}_N^{\text{test}}$ will coincide with the second partition, $\boldsymbol{P}_N^{\text{train}}$ with the third and fourth partitions, and $\boldsymbol{P}_N^{\text{hist}}$ with the remaining ones. The final detection probability is then calculated by adding up the correctly identified attacks across all folds and dividing by the total number of attacks; the false alarm rate is the total number of false alarms divided by 8784.

3.5.2 Detection of intelligently designed attacks

We use the bi-level problem in [3] to design attacks that simulate specific changes in loads to cause physical overflows on a target line, while being unobservable to the system operators (and SE). The testing procedure described in the previous sections is employed here to verify the ability of the proposed detector and grouping strategy in correctly identifying malicious loads resulting from these intelligent attacks.

The bi-level problem in [3] is structured so that any one branch can be selected as a target, and an attack will be designed to maximize the flow on it. Depending on the specific system conditions, a successful attack (that is, one causing the resulting power flow to go above the branch rating) may not exist; generally, the higher the pre-attack flow, the more likely the attack will lead to overflow. For this reason, the first step in designing the attacks is to run an AC optimal power flow (ACOPF) for every load vector in $\boldsymbol{P}_N$ to identify any congested branch. For the purpose of this study, a congested branch is any line or transformer that has a base case power flow loading of 90% of its rating or more. Attacks are designed on each hour of 2016 for which one or more branches are congested. These branches are individually selected

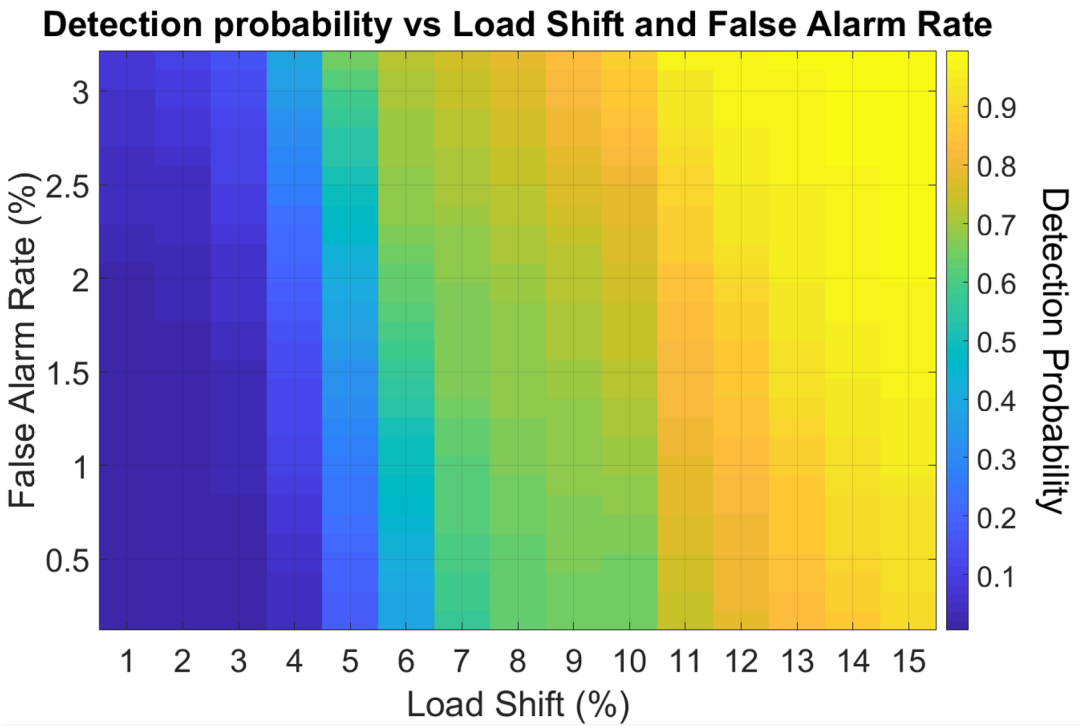

Figure 3.4: Intelligent attacks: detection probability as a function of load shift and false alarm rate.

as targets of the attacks; thus, an hour will have as many different attacks as the number of branches with base case flow above 90% in that hour. Moreover, for each target branch, attacks are designed with a load shift factor ranging from 1% to 15% in steps of 1%. This allows us to study how the detection performance varies in relation to the attack magnitude. As a result of this process, 8861 successful attacks are computed, across every hour, target line, and load shifts.

The resulting attacked load vectors have been tested following the k-folding procedure in Section 3.5.1, where the threshold for each group g_j was varied from 0.9 to 1.1 times $d_{\max,j}$. Figure 3.4 shows the detection probability (colored scale) as a function of the load shift (x-axis) and the false alarm rate (y-axis). It can be seen that the detector does not perform well on attacks with very low load shifts, while for load shift between 10 and 15% the detection probability goes from 80 to 100% with false alarm rates ranging from 0 to 3%. While the load shift factor is an important

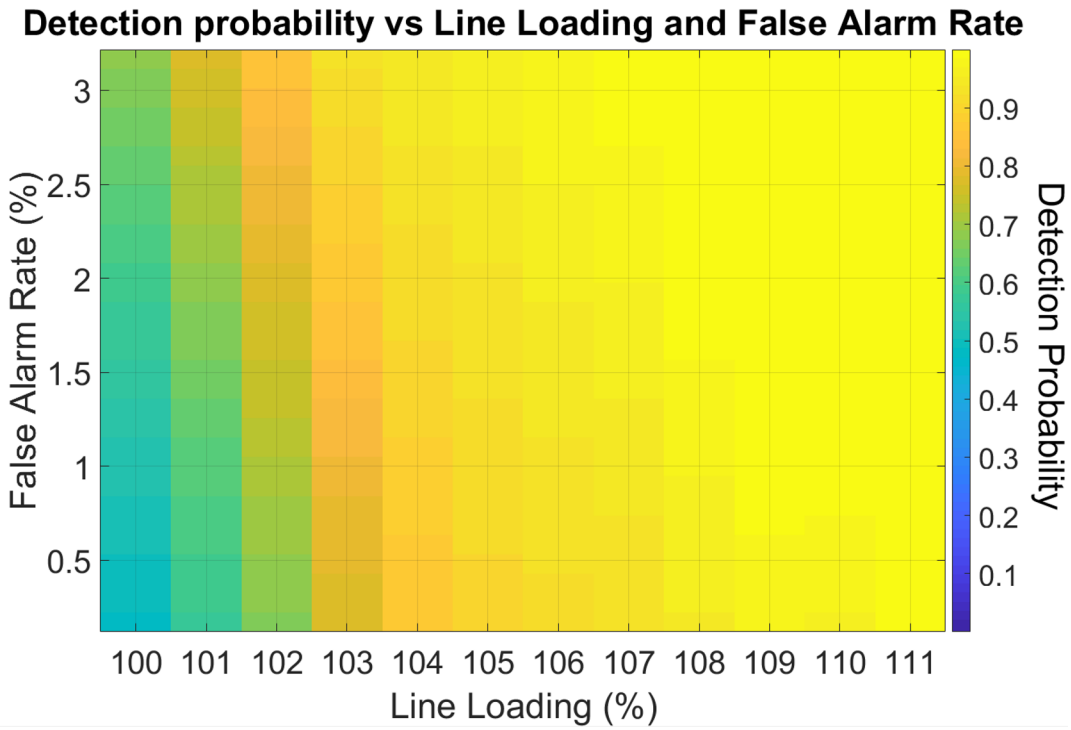

Figure 3.5: Intelligent attacks: detection probability as a function of line overload and false alarm rate.

metric in the design phase of the attacks, from an operator's perspective it is more meaningful to evaluate the physical consequences of the attacks. Figure 3.5 shows the detection probability as a function of the line overload resulting from the attacks. From this figure we can easily see that the detector has extremely high probability of detecting any attack that would cause important physical damage: considering the safety margins built into the operational tools, an overload of 2 or 3% is not likely to cause any system disruption.

3.5.3 Detection of random load redistribution attacks

The experiments in the previous section have shown that the proposed detector is very effective in identifying attacked load vectors designed to create significant overflows on specific target lines. In this section we study the sensitivity of this algorithm to anomalous loads which have not necessarily been intelligently designed.

To do so, a large number of false load vectors will be created starting from the historical data; the detection performance is then computed as the number of modified loads and the amount of load change are varied across a broad spectrum.

The false load vectors are created by randomly selecting a subset of the loads in each vector of $\boldsymbol{P}_N$ and modifying them by either increasing or decreasing their value by a given load shift factor. For this study, the same load shifts as in the previous section are used, while the footprint size of the attack as a percentage of the total number of system loads is varied between 10% and 100% in steps of 10% for every hour. The resulting anomalous load dataset $\boldsymbol{P}_A$ has dimensions $1125 \times H$, where $H = 8784 \times 15 \times 10 = 1,317,600$. Similarly to what done in the previous section, all these false load vectors are fed to the proposed detector and the detection probability computed. In this case, the detection probability is a function of three parameters: the false alarm rate, the load shift, and the footprint size. Figure 3.6 shows the detection probability (colored scale) as a function of the footprint size (x-axis) and the load shift (y-axis), when a high false alarm rate of 5.5% is allowed (top) and for a very low false alarm rate of 0.4% (bottom). Clearly, for a given load shift and footprint size, the detection probability is higher when the false alarm rate is higher. Overall, the detector performs well, having perfect detection capability for a wide range of different attacks. Compared to the detection performance on intelligently designed attacks, the detector is not as good at identifying the random attacks with small load shift and small percentages of attacked loads. This can be explained by the fact that the intelligent attacks are designed in such a way that the modified loads belong to a spatially concentrated subgraph, thus it is likely that some of the load groups will include a large number of attacked loads. In the random attacks, the loads are modified across the whole network and, hence, distributed across a higher number of groups; because of this, each group will experience a smaller deviation from the

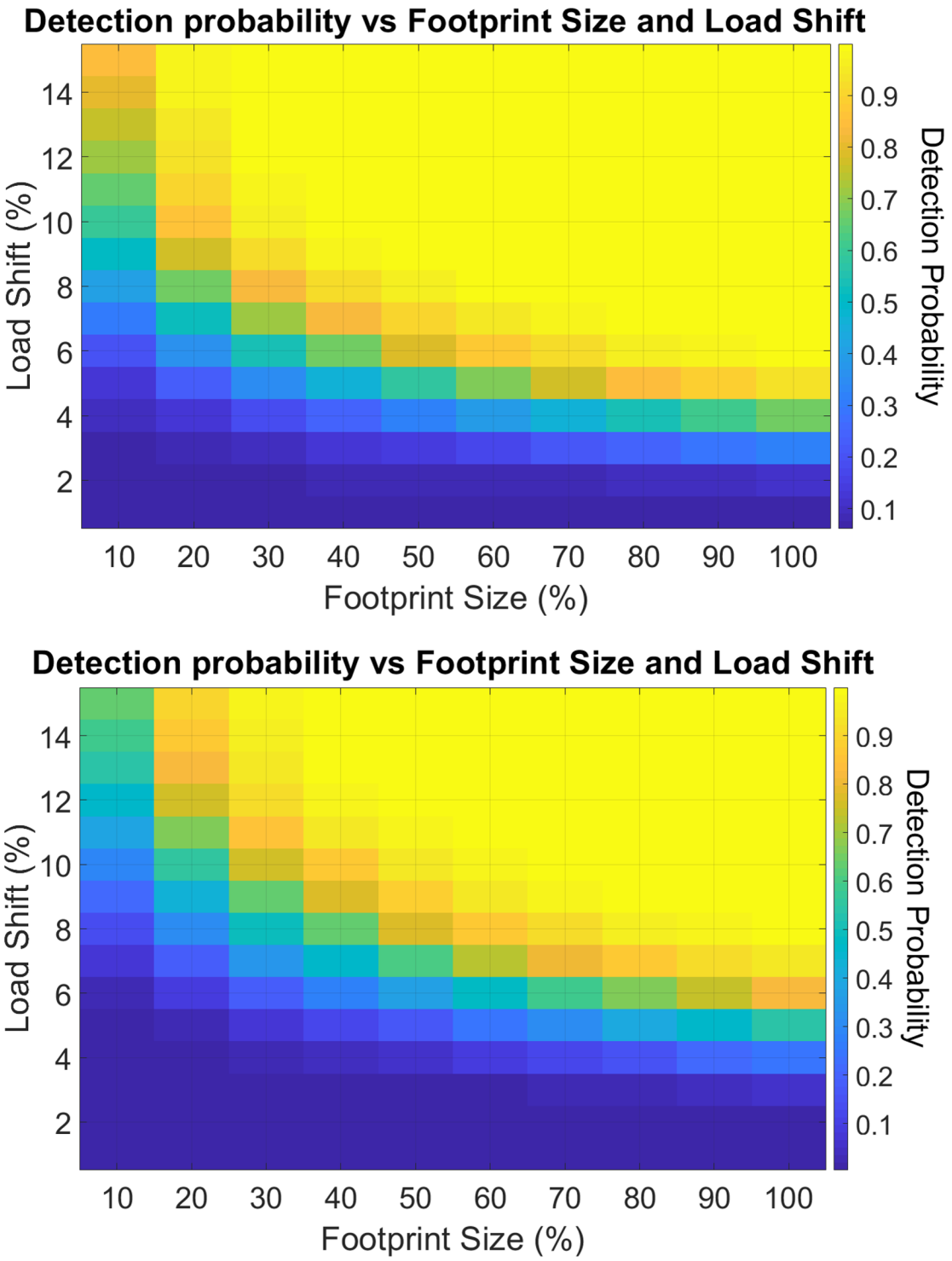

Figure 3.6: Random attacks: detection probability as a function of load shift and footprint size for false alarm rate of 5.5% (top) and 0.4% (bottom).

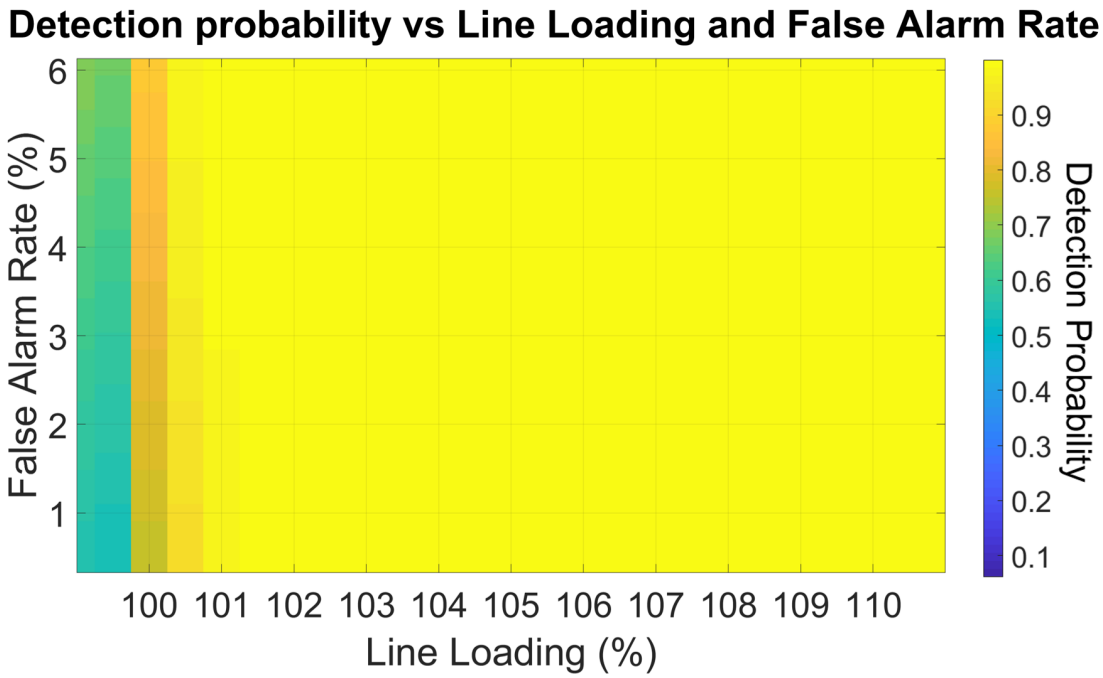

Figure 3.7: Random attacks: detection probability as a function of line overload and false alarm rate.

normative data, resulting in worse detection capability. On the other hand, because of this fact, the random attacks are less likely to cause line overloads. Figure 3.7 shows the detection probability versus line overload and false alarm rate. From these results it can be seen that any random attack that would result in line overloads is easily detected, demonstrating the high effectiveness of the proposed algorithm in detecting anomalous and dangerous load vectors.

3.5.4 Integration within EMS

The proposed detector has been fully implemented in a state-of-the-art EMS platform developed at Arizona State University [28]. This software was created as part of NSF Grant 1449080, by an ASU team lead by Dr. Lalitha Sankar, Dr. Kory Hedman, and Dr. Oliver Kosut and in collaboration with Dr. Robin Podmore from IncSys, Inc., leader in power system simulation tools and operator training [29], [30].

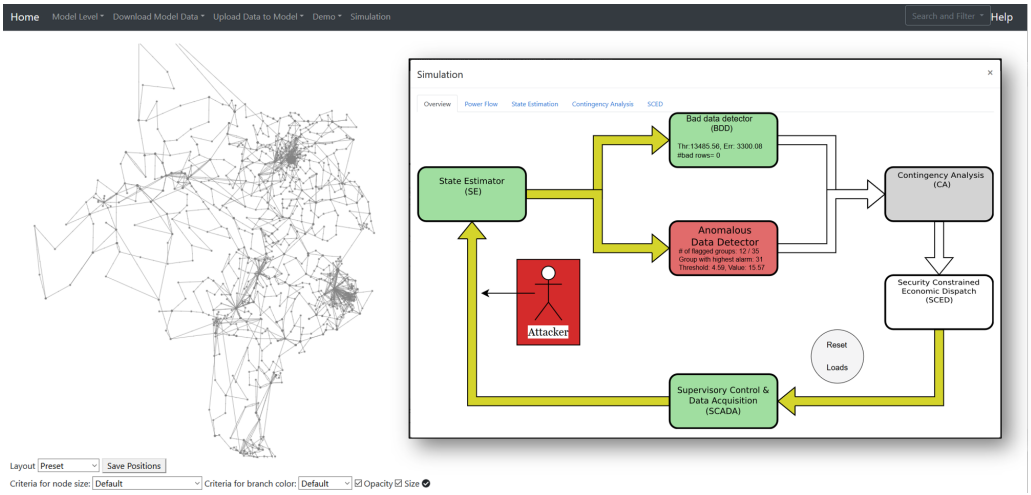

Figure 3.8: Implementation of the improved bad data detector within an EMS

Figure 3.8 shows the interface of the platform: on the left is the network graph of the Texas system, while, on the right, the simulation page with the main blocks of the EMS is displayed. In the example shown, the traditional residue-based BDD has easily been bypassed, while the proposed anomalous load detector identifies the attack and gives information on the extent of the attack based on the number of groups that raised a flag. Overall, this platform allows for the testing of the detector in a realistic power system operations environment while showcasing its effectiveness in terms of computational efficiency and integration within energy managements systems. Details on the design of the software platform and its building blocks can be found in [31], while the code for the attack detection algorithm is freely available on Github [32].

3.6 Attack localization

In the previous sections, we have introduced a load anomaly detector based on nearest neighbor and a grouping strategy which was shown to have excellent performance against both intelligently designed attacks and random load changes. This algorithm can be extended beyond simply determining whether a load vector con-

tains anomalous data or not; it can be leveraged to determine which buses have been modified or are deviating from their usual behavior. Localizing the subgraph affected by an attack or load anomaly represents a step forward in terms of system operations security. Knowing which loads are likely to have caused the detector to raise an alarm is an important step in the implementation of secure EMS functionalities. For example, the load values which are determined to be unreliable could be replaced by forecasted values or an uncertainty margin assigned to them so that the system could be operated in a secure state.

A similar approach for secure operations against cyber-attacks is studied in [33], where the authors present an optimal dispatch problem to find a secure and cost-effective dispatch solution considering variable bus loads and, thus, protecting the system from unexpected load changes. Also, in [34], a secure unit commitment (UC) problem is formulated such that in case of a cyber attack the system operator can switch from the normal UC solution to a secure one while following all network constraints. The issue with these approaches is that it would cause the system to be operated in a too conservative and, thus, less efficient state for most of the time. The advantage of being able to detect and localize an attack is that the system operator can make a better informed decision on when and how to secure the system, without impacting normal operations.

3.6.1 Likelihood determination

The grouping strategy provides an approximate way of localizing the attacks by identifying groups of loads that deviate from their normative behavior. In this section, we describe a statistical approach to further analyze the values of the individual loads to identify which ones are more likely to have triggered the detector. Because of the many attack subgraphs that are possible, determining exactly which are the attacked

loads would be extremely hard. For this reason, our goal is to assign to each load a probability q_l that represents the likelihood of that load being attacked. In this sense, the likelihood is a risk measure and it can be quantified using an empirical metric that relies on estimated likelihoods, namely average log-loss (also known as cross-entropy) [35]. Average log-loss is defined as

$$l = \frac{1}{n_L} \sum_{l=1}^{n_L} -[y_l \log_2(q_l) + (1 - y_l)\log_2(1 - q_l)] \tag{3.4}$$

where n_L represents the total number of samples (e.g. loads tested), q_l is the probability associated with each load, and y_l is 1 if the load was indeed attacked and 0 if it was not modified.

We define the values of the loads in group j at time i as $p_i^j = [p_{i,1}^j, p_{i,2}^j, \ldots, p_{i,k_j}^j]^T$, where k_j is the number of loads in group j. The minimum distance $d_{i,j}$ between the load vector p_i^j and the historical data is computed using (3.2); as explained in Section 3.4.2, if $d_{i,j}$ is greater than threshold τ_j, group j is said to raise a violation at time i. Moreover, define the loads in the nearest neighbor of p_i^j as $h_r^j = [h_{r,1}^j, h_{r,2}^j, \ldots, h_{r,k_j}^j]^T$, where h_r is the r^{th} historical load vector in $\boldsymbol{P}_N^{\text{hist}}$. For each load in group j, the normalized difference between load l at time i and its corresponding value in the nearest neighbor h_r^j is computed as

$$\delta_{i,l}^j = \left| \frac{p_{i,l}^j - h_{r,l}^j}{h_{r,l}^j} \right| \qquad l = 1, \ldots, k_j. \tag{3.5}$$

We cannot directly look at this normalized difference to know if a load is attacked because different loads could have different amounts of deviation. In order to account for this variability, we determine the normative behavior of each load by computing the first and second order statistics of its normalized difference:

$$\mu_{\delta_l^j} = \frac{1}{n_i} \sum_{i \in \boldsymbol{P}_N^{\text{train}}} \delta_{i,l}^j \qquad \forall l, \forall j \tag{3.6}$$

$$\sigma_{\delta_l^j} = \sqrt{\frac{1}{n_i} \sum_{i \in \boldsymbol{P}_N^{\text{train}}} (\delta_{i,l}^j - \mu_{\delta_l^j})^2} \qquad \forall l, \forall j. \tag{3.7}$$

Given a specific load vector $p_i \in P_N^{\text{test}}$ and its corresponding $\delta_{i,l}^j$ for all l and j, we determine how far each load deviates from the normative behavior using a Z-score which is defined as follows:

$$z_{i,l}^j = \frac{\delta_{i,l}^j - \mu_{\delta_l^j}}{\sigma_{\delta_l^j}}. \tag{3.8}$$

Intuitively, the Z-score indicates the number of standard deviations by which $\delta_{i,l}^j$ is above (or below) the mean for load l in group j observed in the attack-free data.

Based on this setup, there exists a joint distribution $Q(a, v, z)$ between whether a load was attacked ($a = 1$) or not ($a = 0$), if it belongs to a group that raised a violation ($v = 1$) or not ($v = 0$), and its Z-score z. While $Q(a, v, z)$ is not known, we can empirically estimate the conditional probability $Q(a|v, z)$ of a load being attacked given its Z-score and whether it raised a violation or not. In other words, our goal is to define a likelihood function $\mathcal{L}(v, z)$ that takes as inputs a load's Z-score and whether it raised a violation to determine the probability that the load is attacked.

First, we compute the Z-score (3.8) for all intelligently designed attacks in P_A that result in an overload of 3% or more. As discussed in Section 3.5.2, those are the attacks that can cause significant damage and they are almost always detected by the nearest neighbor algorithm. The histogram of the Z-score for the loads that belonged to groups that raised a violation is shown in Fig. 3.9. In particular, the solid black line represents the histogram of Z-scores for loads that were attacked and we indicate as $\phi_{a=1,v=1}(z)$, while the black dotted line represents the histogram for loads that were not attacked $\phi_{a=0,v=1}(z)$. From these two curves, we notice that overall if a load belongs to a group that raised a violation it is very likely that the load is indeed being attacked. Moreover, the higher the Z-score, the more likely it is that a load is attacked. Based on these observations we can now define a function that

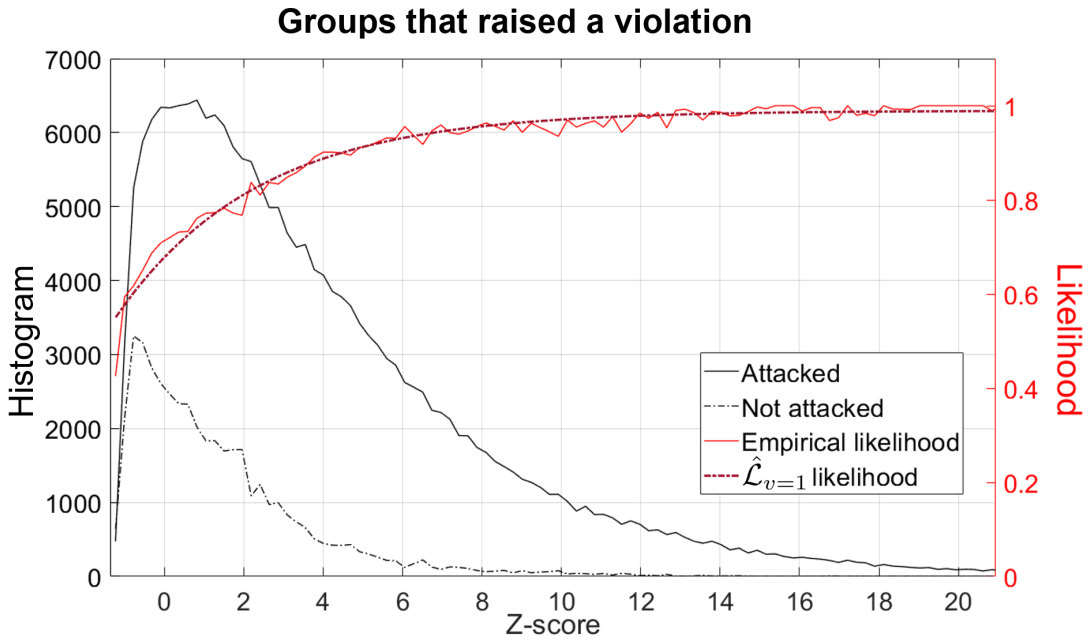

Figure 3.9: Distribution of Z-scores (in black) and likelihood function (in red) for loads in groups that raise a violation.

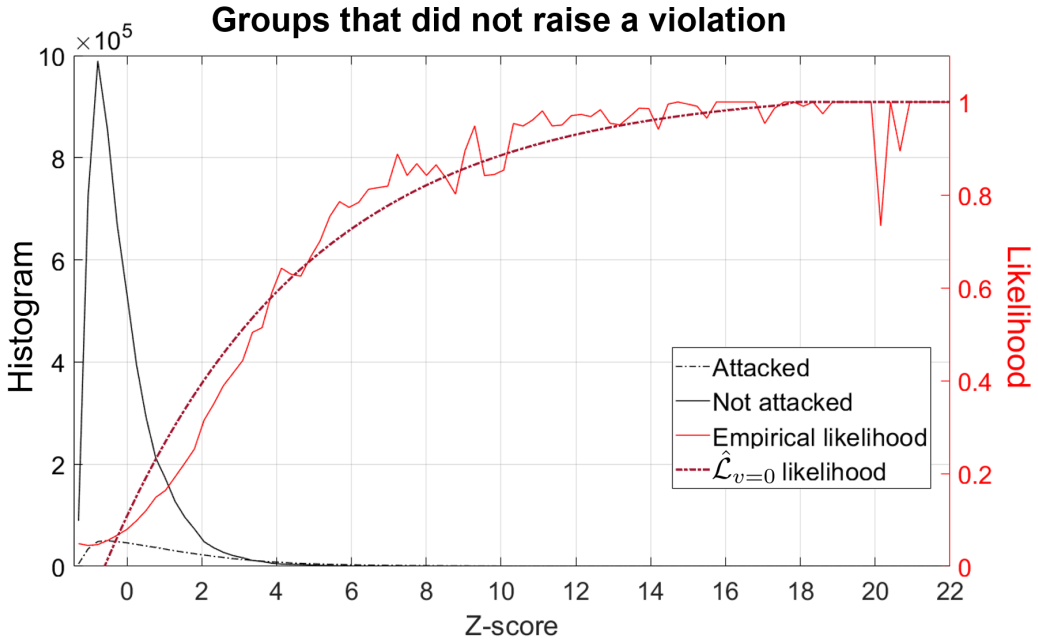

Figure 3.10: Distribution of Z-scores (in black) and likelihood function (in red) for loads in groups that do not raise a violation.

maps the Z-score of a load to the likelihood of the load being attacked. The estimated conditional likelihood for loads that are in groups that raise a violation is computed as

$$\bar{\mathcal{L}}_{a|v=1,Z=z} = \frac{\phi_{a=1,v=1}(z)}{\phi_{a=1,v=1}(z) + \phi_{a=0,v=1}(z)} \tag{3.9}$$

and it is shown by the solid red line in Fig. 3.9. For the set of data points we obtain using (3.9), we fitted a smooth curve of the form $a \cdot e^{-b \cdot x} + c$ to avoid overfitting as shown by the dotted red line. This curve is defined as the conditional likelihood $\hat{\mathcal{L}}_{a|v=1,Z=z}$ which can be used to assign to each load a probability of being attacked based on its Z-score. The same procedure is performed on the loads in groups that do not raise a violation and the corresponding likelihood $\hat{\mathcal{L}}_{a|v=0,Z=z}$ is estimated; the results are shown in Fig. 3.10. Comparing the two conditional likelihood functions we notice that, for low Z-score values, $\hat{\mathcal{L}}_{a|v=1,Z=z}$ reaches a minimum likelihood value of around 0.5 while $\hat{\mathcal{L}}_{a|v=0,Z=z}$ reaches zero.

3.6.2 Numerical results

The performance of this approach is tested on the intelligently designed attacks from Section 3.5.2, with $\tau_j = d_{\max,j}$. The conditional likelihood functions $\hat{\mathcal{L}}_{a|v=1,Z=z}$ and $\hat{\mathcal{L}}_{a|v=0,Z=z}$ are learned on 70% of the attacks and they are tested on the remaining 30%. The Z-score for every load is computed using (3.8) and the average log-loss as in (3.4).

As a way of comparison, we also tested two simpler approaches to assign likelihood values to each load. The first one does not rely on the Z-score and only considers if the load belongs to groups with violations or not: based on our data, on average, in a group that raised a violation 82% of the loads are attacked, while in groups that did not raise violations only 10% are actually attacked. Based on this prior knowledge, the first simple approach assigns a likelihood $q_l = 0.82$ if load l is in a

group with violations and $q_l = 0.10$ otherwise. The second approach is even simpler and it assigns a fixed likelihood to every load regardless of which group they belong to; from our results, the optimal value for this approach is $q_l = 0.15$. The results of the Z-score-based approach (indicated as $q(a|v, z)$), as well as the two simpler ones (indicated as $q(a|v)$ and $q(a)$) are summarized in Table 3.1. We can see that the more sophisticated the approach (i.e. the more information is used), the smaller the average log-loss.

Table 3.1: Performance comparison of the three likelihood approaches.

Approach	$q(a\|v, z)$	$q(a\|v)$	$q(a)$
Average log-loss	0.340	0.489	0.608

Chapter 4

SYNTHETIC LOAD GENERATION USING PRINCIPAL COMPONENT ANALYSIS

4.1 Related work

In this chapter, we develop an automatic, data-driven technique to generate bus-level historical load data for any given transmission-level grid model. The goal is to create a generative model which takes as only inputs the system topology and a set of base case loads and returns individual time-series load data for every bus in the system for an arbitrary period of time.

Currently, only a few of the grid models publicly available to researchers include historical load curves. For these models, the most common approach is to provide time-series data for the system demand at a net or zonal level and calculate the bus loads at each time step as a fixed fraction of the aggregate load. This method is simple to implement since many utilities and system operators publish historical net load information for their systems, often over multiple years [36]. One of the most widely used system models which was developed following this approach is the IEEE RTS-96 case described in [37]. The drawback is that because a fixed load ratio is used for every bus, the variability in behavior that exists between different types of loads at different times of the day or week is not captured; having each load follow the same profile over time is not realistic. An alternative method consists in creating load data as a combination of prototypical load models at a bus level. This technique provides an overall more realistic dataset by creating individual profiles for each load, but it requires detailed geographical and/or demographical information to determine

the load composition at each bus. For example, in [25], historical load data for the synthetic Texas model [24] is created starting from several typical load curves and combining them according to load types and actual population data of the state of Texas.

In our work, we introduce a technique based on principal component analysis (PCA) to model the temporal behavior of loads, and topology-based factors to model the spatial correlation between loads. The features learned from the real data are then used to generate realistic, individual temporal profiles for the loads of a new grid model. Examples of the application of PCA to the study of electrical loads can be found in load forecasting applications [38]. In [39, 40, 41], the authors describe the use of PCA for the processing of the data used in long and short-term forecasting models for a system's net load. Our approach differs in that we use PCA to extract temporal profiles from bus-level time-series data rather than identify the correlation between the variables governing the system net load.

4.2 Dataset description

The real data used in this chapter is proprietary and was provided by a large American independent system operator.

Our proposed technique requires two pieces of data: the bus-level historical load values and the topology of the system. The load data can be represented as a matrix $P \in \mathbb{R}^{n \times t}$, where n is the number of load buses in the system, and t is the number of time samples. The proprietary dataset contains more than 3500 loads, each sampled at hourly intervals for 167 consecutive hours (which is one hour short of a full week). The topology of the system can be represented as an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of all buses and $\mathcal{E}$ is the set of branches.

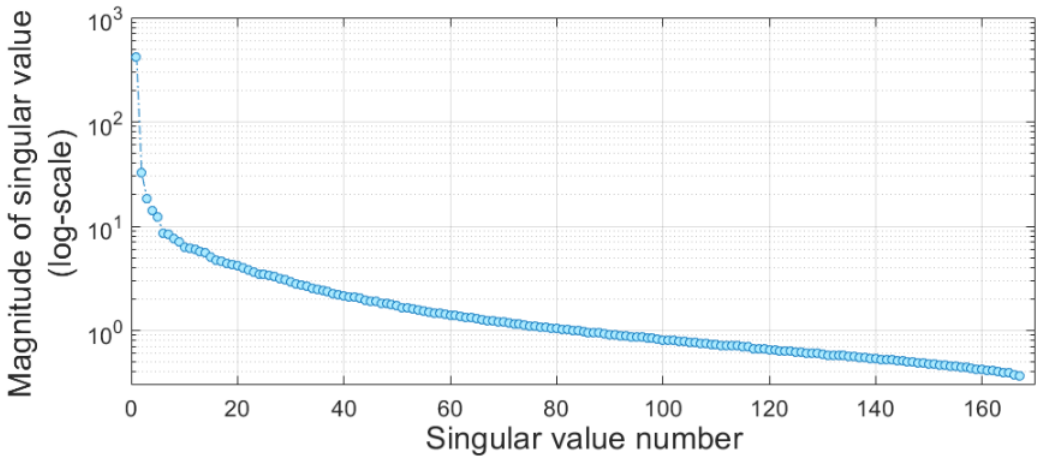

Figure 4.1: Magnitude of the singular values.

4.3 Temporal correlation

4.3.1 Principal component analysis

At a transmission level, the loads represent aggregates of residential, commercial, and industrial entities. Based on the assumption that the loads within each type behave similarly (especially residential and commercial), in a power system we can expect to observe common profiles among all loads. An effective way to identify and extract patterns from a dataset is by using PCA via singular value decomposition (SVD). The load matrix P can be factorized using SVD as $P = U\Sigma V^T$, where $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{t\times t}$ are unitary matrices and $\Sigma \in \mathbb{R}^{n\times t}$ is an upper diagonal matrix. This factorization is able to extract and rank the common basis which, via linear combination, can reconstruct each load profile. In particular, the rows of V^T, which are vectors of size $1 \times t$, correspond to archetypal *temporal profiles* and they constitute the principal components. Each diagonal element of Σ, called a *singular value*, represents a scale factor which multiplies each corresponding principal component. Because the singular values are sorted from largest to smallest, they give an indication on the relative importance of each temporal profile contained in the

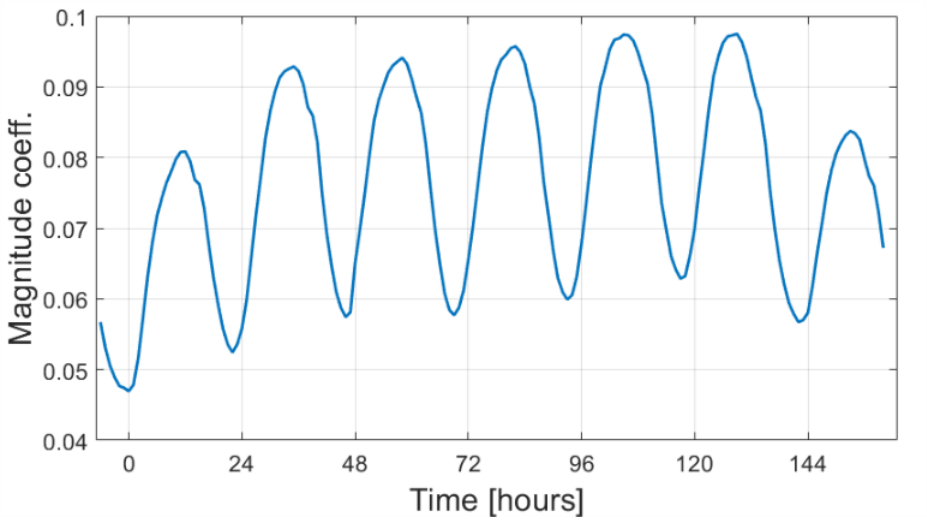

Figure 4.2: Temporal profile corresponding to the largest singular value. Each hour multiple of 24 indicates midnight of the corresponding day.

V matrix. Fig. 4.1 shows the singular values obtained from the factorization of P, and it is clear that the first value is much larger than the following ones. Thus, the temporal profile corresponding to the first principal component, shown in Fig. 4.2, is the most dominant in determining the behavior of the loads. This profile shows the simplest and most common behavior: the load increases during the day, reaches a peak around noon and decreases in the evening. Moreover, considering that this data starts on a Sunday, we can see how the weekend peaks are lower than those of weekdays. The load profile at each individual bus is obtained as a linear combination of the principal components (columns of V) scaled by the singular values and multiplied by the corresponding coefficients in each row of U. These coefficients determine the composition of any given bus in terms of the archetypal profiles constituted by the principal components.

4.3.2 Feature selection

As Fig. 4.1 shows, the first few rows of V^T have a significantly higher weight compared to the remaining ones, meaning that the original load matrix P can be

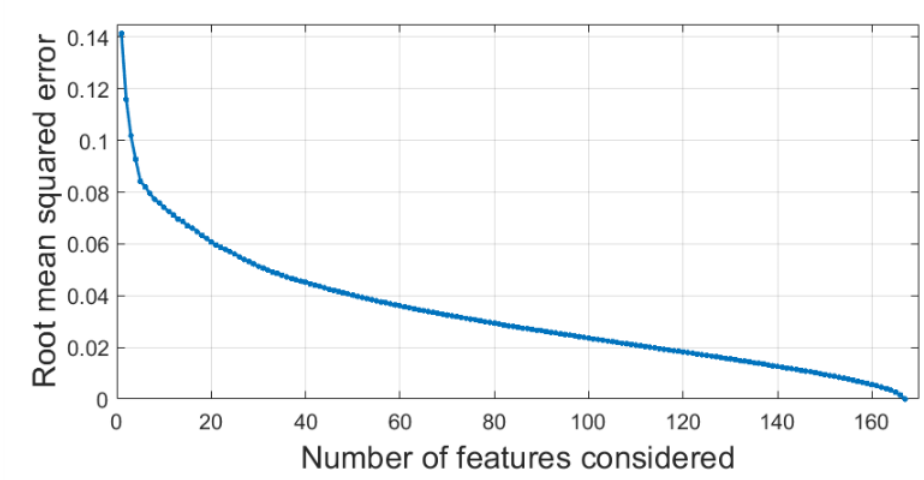

Figure 4.3: Root mean squared error as a function of the number of features used to approximate the real loads.

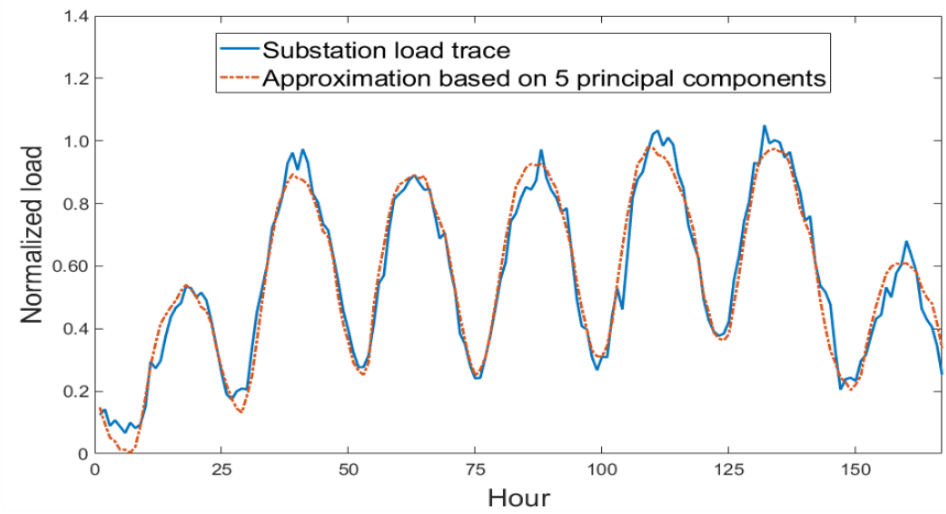

Figure 4.4: Load trace across one week for an example substation. Also shown is the approximation based on the first 5 largest features.

approximated with good accuracy using only a subset of the basis vectors, or principal components. To demonstrate this fact, an approximation $\hat{P}$ of the original load matrix can be computed as $\hat{P} = U(:, 1:f)\Sigma(1:f, 1:f)V^T(1:f, :)$, where $1 \leq f \leq t$ is the number of temporal profiles with the largest singular values. [1] The average root mean squared error (RMSE) as a function of the number of features used is shown in Fig. 4.3 and it allows for two observations. First, as one would expect, increasing the number of basis leads to a progressively better approximation which reaches an error of zero for $f = t$. Second, the error decreases sharply and almost linearly until $f = 5$, and then it slowly decays to zero. This means that the first five features can capture the main behavior of the load and they are sufficient to generate realistic synthetic load profiles. Fig. 4.4 is an example showing a real load and its approximation using a limited number of temporal profiles (notice that the load values have been normalized for anonymity reasons). It can be seen that the behavior of the load is captured very accurately while the small magnitude variability is smoothened out; this drawback is addressed by adding random noise, as explained in the next section.

4.3.3 Temporal generative model

Having identified some typical patterns, a new load profile can be created by generating a vector of coefficients and multiplying it by the set of base profiles contained in V. To compute these new coefficients we need to learn the distribution of the coefficients in the original data (e.g. the columns of U). The probability distribution functions (PDFs) are estimated using the Matlab Distribution Fitter App. Each column of U is analyzed independently and the best PDF for each is determined. Fig. 4.5 shows a histogram representation of the empirical coefficients for the first

[1] Notation: $U(:, 1:f)$, $\Sigma(1:f, 1:f)$, and $V^T(1:f, :)$ indicate the first f columns of U, the first f columns and rows of Σ, and first f rows of V^T respectively.

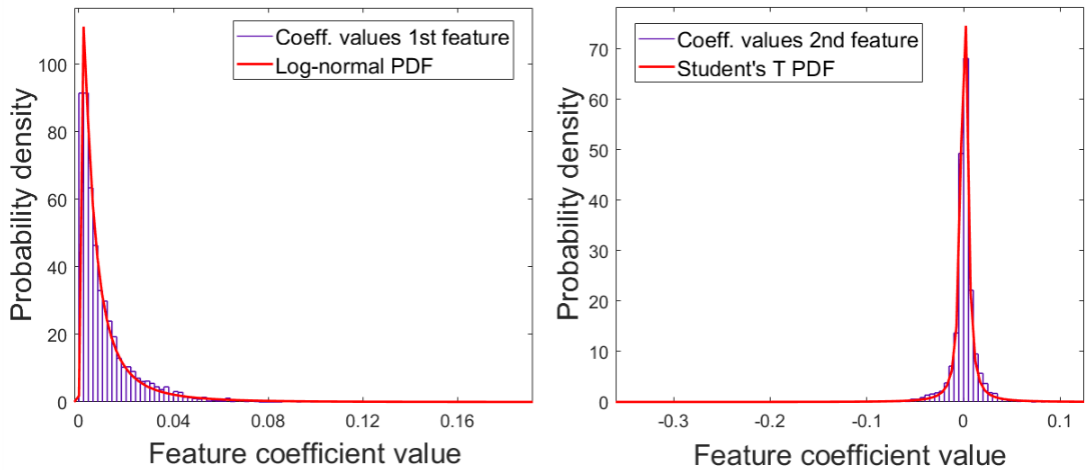

Figure 4.5: Empirical and estimated probability density functions for the first feature (left) and the second feature (right) of the SVD load model.

two features and the respective fitted PDFs as an example.

Since it was determined that the first five features will be used to generate the new data, this fitting procedure is performed for the first five columns of U. Table 4.1 shows the selected PDFs and their defining parameters. It is interesting to notice that the first coefficient is best approximated by a log-normal function, while all the successive ones follow a Student's t distribution.

At this point, to generate new profiles it is sufficient to create a new coefficient matrix U_{new}, where each entry is sampled from the appropriate distribution, and multiply it by $\Sigma \in \mathbb{R}^{f \times f}$ and $V^T \in \mathbb{R}^{f \times t}$, where f is the chosen number of basis to be used ($f = 5$ in our case). The remaining uncertainty which is not captured by using a limited number of features is approximated by adding random noise to the profiles resulting from the above generative process. The noise has been empirically estimated to be normally distributed, with zero mean and $\sigma = 0.02$. The resulting generative model for m new loads can be written as

$$P_{\text{new}} = U_{\text{new}} \Sigma V^T + W \tag{4.1}$$

where $P_{\text{new}} \in \mathbb{R}^{m \times t}$, $U_{\text{new}} \in \mathbb{R}^{m \times f}$, $\Sigma \in \mathbb{R}^{f \times f}$, $V^T \in \mathbb{R}^{f \times t}$, and $W \in \mathbb{R}^{m \times t}$ is the

Table 4.1: Probability distribution functions

Feature number	**PDF**	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	$\boldsymbol{\nu}$
1	log-normal	-5.13	1.15	-
2	Student's t	7.55×10^{-4}	3.7×10^{-3}	1.16
3	Student's t	1.29×10^{-4}	4.7×10^{-3}	1.26
4	Student's t	1.51×10^{-3}	3.5×10^{-3}	1.08
5	Student's t	1.01×10^{-3}	4.3×10^{-3}	1.18

matrix whose entries are sampled from $\mathcal{N}(0, \sigma^2)$.

The model is tested by generating 3000 synthetic load profiles using the first five basis of the real data, both with and without noise. Each resulting load matrix is then decomposed and approximated via SVD using an increasing number of features in the same way as done on the original data in Section 4.3.2. The average root mean squared error of the synthetic data with and without noise is shown in Fig. 4.6. We can see that in the absence of noise, as one would expect, the approximation error reaches zero when five features are used to reconstruct the data. When noise is included, the error follows a similar curve to that of the original data, shown in Fig. 4.3. Thus, we can confirm that the generative model is able to capture both the predominant, long-term load behaviors as well as the short-term, high variability and randomness of the data. An example of load profile generated using the model in (4.1) is shown in Fig. 4.7.

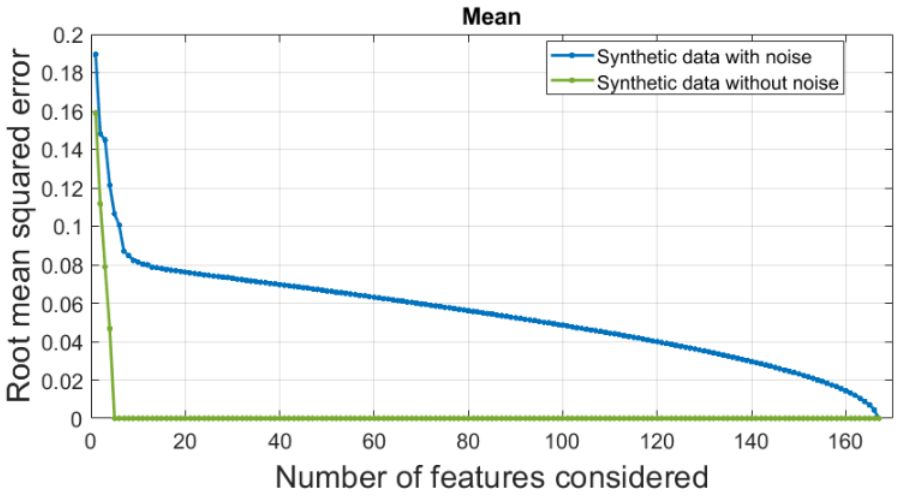

Figure 4.6: Root mean squared error as a function of the number of features used to approximate synthetic data generated with and without the addition of noise.

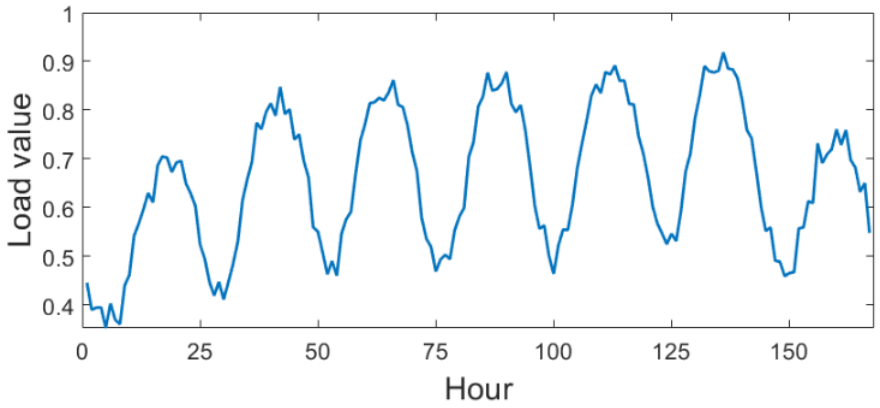

Figure 4.7: Example load trace across one week generated using the model described by (4.1).

4.4 Spatial correlation

The profiles resulting from (4.1) are generated independently of each other, which, in general, is not a valid assumption about the loads in a power system. Realistically, loads that are geographically close should show some degree of correlation in their temporal behaviors. This is true because of two factors: (i) nearby loads are likely to be of the same type (residential, commercial, industrial, etc..), and (ii) geography-dependent factors (such as weather conditions) will affect neighboring loads in similar

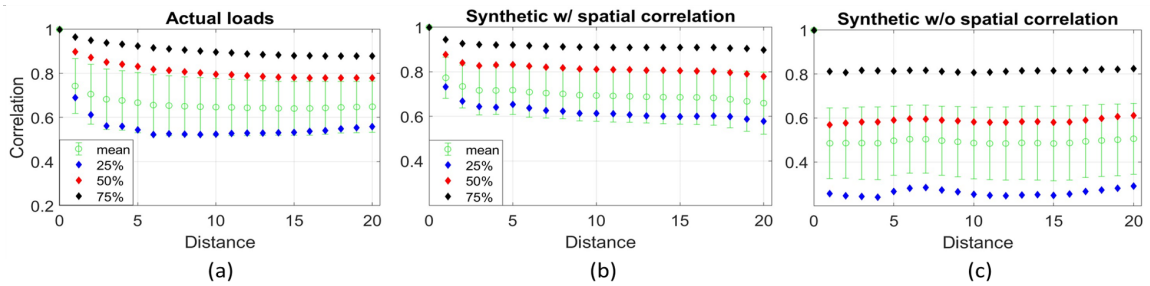

Figure 4.8: Statistics of the correlation coefficients between load profiles as a function of the distance between buses, for the original data (left) and synthetic data generated considering the spatial correlation (center) and without considering it (right).

ways. For this reason, it is important for any generative model to take into account the spatio-temporal correlation between loads that can be learned from a real dataset.

Let us define $b_i = \{b_{i,1}, b_{i,2}, ..., b_{i,t}\}$ as the load vector for bus i from time 1 to t; the correlation coefficient $r_{i,j}$ between the load vectors of buses i and j is:

$$r_{i,j} = \frac{\sum_{k=1}^{t}(b_{i,k} - \bar{b}_i)(b_{j,k} - \bar{b}_j)}{\sum_{k=1}^{t}(b_{i,k} - \bar{b}_i)^2 \sum_{k=1}^{t}(b_{j,k} - \bar{b}_j)^2} \tag{4.2}$$

where $\bar{b}$ indicates the sample mean. To understand the spatial characteristics of the original dataset, the correlation coefficient between buses is computed for every combination (i, j) with $1 \leq i \leq m$ and $1 \leq j \leq m$. Furthermore, each value $r_{i,j}$ is paired with the distance between the two corresponding buses, indicated as $\text{dist}_{i,j}$ and defined as the number of branches along the shortest path connecting buses i and j. The correlation coefficients are then collected as a function of their associated distance and the following metrics are computed: mean, standard deviation, and 25^{th}, 50^{th}, and 75^{th} percentiles. This process allows us to understand at a general level how the similarity between buses varies as a function of their relative distance. Fig. 4.8(a) shows the statistics computed for the real loads dataset. We can see that, on average, as the distance between two buses increases the correlation slowly decreases, confirming the previous hypothesis that some spatial correlation exists.

To capture this behavior, the individual coefficients of matrix U_{new} from the model

in (4.1) must be modified to take into account the values of the neighboring buses. To do so, the coefficients are first generated by randomly drawing from the estimated distributions as described in Section 4.3.3. Then, each row of U_{new} is modified by adding to the coefficients vector of each bus a linear combination of the randomly generated vectors of the neighboring buses. Simulations have shown that the best results are obtained when each bus is modified by taking into account its neighbors within a maximum distance of 3. Moreover, the scaling factor that multiplies the coefficients of the neighbors is defined as a function of the distance, such that the greater the distance between two buses, the smaller the scaling factor. Formally, the model is rewritten as

$$P_{\text{new}} = (DU_{\text{new}})\Sigma V^T + W \tag{4.3}$$

where $D \in \mathbb{R}^{m \times m}$, and each entry is computed as

$$d_{i,j} = \begin{cases} 1, & \text{if } i = j \\ e^{-2\text{dist}_{i,j}}, & \text{if } \text{dist}_{i,j} \leq 3 \text{ and } i \neq j \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

4.5 Testing of the generative model on the Polish test case

We test our proposed generative model by creating load profiles for the publicly available grid model for the country of Poland [42]. This test case, commonly referred to as the 2383 Polish case, contains 2383 buses and 1822 loads. For each load, random coefficients are sampled from the five distributions described in Table 4.1 and the matrix D is computed based on the topology of the Polish system. Equation (4.3) is used on this data along with the singular values and principal components from Sections 4.3.1 and 4.3.2 respectively.

4.5.1 Spatial correlation

The spatial characteristics of the generated data are shown in Fig. 4.8. Plot (b) shows the statistics of the correlation coefficients as a function of the distance between buses for the synthetic Polish load data generated using (4.3). Plot (c) shows the same metrics for synthetic data resulting from (4.1), which does not take into consideration the spatial correlation between loads. It is clear how this latter graph greatly differs from that of the real data, while the data generated considering the spatial characteristics of the loads closely matches the behaviors of real loads. These results prove that the weight matrix D with exponential coefficients in (4.4) closely approximates the correlation between load profiles observed in real world data.

4.5.2 Loads scaling

The last step in creating realistic historical load data consists in scaling the profiles obtained from (4.3) to the values of the base case loads. This can be done by scalar multiplication of P_{new} by the column vector $S \in \mathbb{R}^m$. The entries of this scaling vector can be determined in various ways, depending on the nature of the available base case loads. For example, s_i can be defined as $s_i = P_{\text{base},i}/f(P_{\text{new},i})$, where $f(P_{\text{new},i})$ is any function of the profile i, such as average, minimum etc. In the case of the Polish system, the loads provided in the model represent peak hour values so we have selected the scaling function to be $f(P_{\text{new},i}) = \max(P_{\text{new},i})$. In this way, the maximum load values of the synthetic historical data will coincide with the original base case loads.

4.5.3 *Power flow validation*

Having demonstrated the realistic nature of the generated loads in terms of spatio-temporal characteristics, it is important to verify that this data actually represents feasible cases from a power system perspective. To this end, ACOPF is run on every hour of the synthetic historical data to check for convergence. The results of this study show that all the 167 synthetic cases lead to converging solutions with bus voltages within the limits and moderate levels of congestion. This confirms that the generated loads can be supplied without exceeding transmission capacity, while leading to valid operating cases.

4.6 Application to PMU Time Scale Load Data

One of the advantages of the proposed PCA-based generative model is that it can be used on load data at any time scale. In fact, this method was successfully applied to the generation of PMU-speed load data which was used to create a testbed for the validation of FDI attacks against PMU-based system operations [43].

Specifically, the PMU measurements used to test the cyber attacks and the related bad data detectors (BDD) must reflect realistic operating conditions. In our tests, we achieved this by simulating the dynamics of the IEEE 118 bus system with time varying loads and primary generation control. The bus-level time-series load data for this test system is generated based on a real PMU dataset that was provided by a large utility company in the southwest of the US. I this work, we adapted the previously described approach to the learning and generation of load profiles at PMU data speeds.

The utility company provided us with one week worth of PMU data for a group of neighboring substations. From the voltage and current measurements of each bus

and line, we compute the loads of two substations, one at the 500kV level and one at 230kV level. Each time-series is 168 hours long, sampled at 30 samples/sec. From these two data streams we can learn the behavior of loads at different voltage levels and subsequently map them to the loads of the IEEE 118 bus system according to their voltage levels.

For our simulations, we were interested in generating load data at each bus for 10 minutes. For this reason, the time-series load data for one consecutive week was broken into segments of length of 10 minutes; this resulted in 1008 segments, each containing 18,000 samples. The segments were then stacked to form the load matrix $P \in \mathbb{R}^{1008\times 18000}$ which can be factorized as described in Section 4.3.1. In this case, the archetypal temporal profiles (rows of V^T) were vectors of size 1×18000. Figure 4.9 shows the approximation error of P as a function of the number of basis vectors used. It can be seen that the error decreases rapidly up to $f = 10$ and then it slowly reaches zero when all the basis vectors are used. For this reason, the first 10 temporal profiles were used in the generation of the synthetic load profiles.

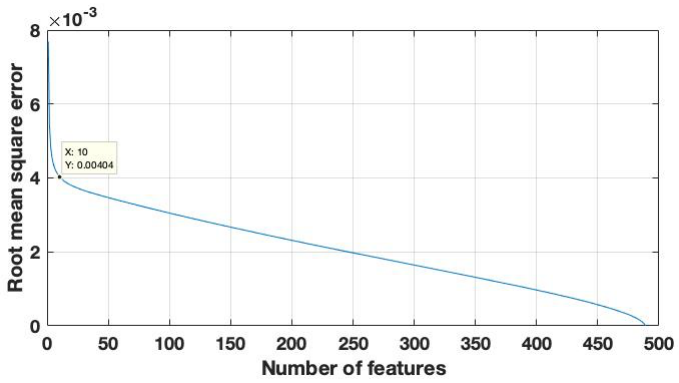

Figure 4.9: Root mean squared error between P and $\hat{P}$ as a function of the number of basis used.

After determining the number of features needed to satisfactorily capture the load behavior at PMU scale, we followed the same process as described in the previous sections to generate the required PMU load profiles.

Chapter 5

PMU DATA STORAGE AND VISUALIZATION

5.1 Background

The PCA-based approach described in the previous chapter was implemented and tested on datasets which had a limited time-length (1 week for the hourly data and 10 minutes for the PMU data) as well as a limited range of load behaviors and patterns. As part of our continued work on power system data analytics, we gained access to a very large dataset (over 70 TB) of real PMU data from a US electric utility. In the next chapters, we will show how this data is used for the learning of load behaviors over very long time horizons (up to multiple consecutive years) at resolutions as high as 30 samples per second. However, in order to store and process such a large amount of data, we first had to design a fast and efficient database solution. In this chapter, we describe the storage scheme we developed and its performance in terms of data retrieval speed. Moreover, in collaboration with Dr. Christopher Bryan and his students at ASU, we created a visualization platform (called PMUVis) which leverages the PMU database to offer data analysis tools for the study of power system events. Specifically, we developed a mechanism for the localization of the source of an oscillation based on the analysis of the frequency spectrum of different measurements across the power grid. The effectiveness of this tool is shown via two study cases in this chapter.

5.2 Data storage and management

The historical PMU dataset we obtained consists of ∼500 PMUs over a three year period, where each PMU records measurements for 18 attributes at a 30 Hz frequency. (At our institution, access to this PMU dataset is provided under non-disclosure agreement provisions, so we anonymize certain features such as PMU IDs and locations as well as the exact number of PMUs.) As this raw PMU data is very large and noisy (total raw dataset size is over 70 TB), we describe here the steps we took to aggregate, store, and retrieve it, as well as our use of signal processing for event analysis.

5.2.1 Storing and accessing raw PMU data

The raw PMU data is saved on our compute cluster's servers using the Parquet file format. Parquet is a columnar storage scheme based on an algorithm for record shredding and assembly [44], that supports efficient and flexible compression and encoding. Blocks in a Parquet file are stored in the form of row groups, each of which in turn contains column chunks. In a column chunk, column values are stored in contiguous memory locations. As Parquet compression is transparent and can be adapted in a column-specific manner, column-wise queries can fetch values with much higher performance compared to row-wise queries.

In our dataset, each Parquet file stores one attribute's worth of data for all PMUs for one day (using 96 fifteen-minutes groups), see Figure 5.1. Each column corresponds to one PMU, and each row is one timestamp. As the PMUs record measurements at 30 Hz (raw values stored as 64-bit floating point values, null values are recorded during data dropout), each Parquet file is ∼500 columns × 2.592M rows. Since each PMU records 18 attributes, each day contains 18 Parquet files.

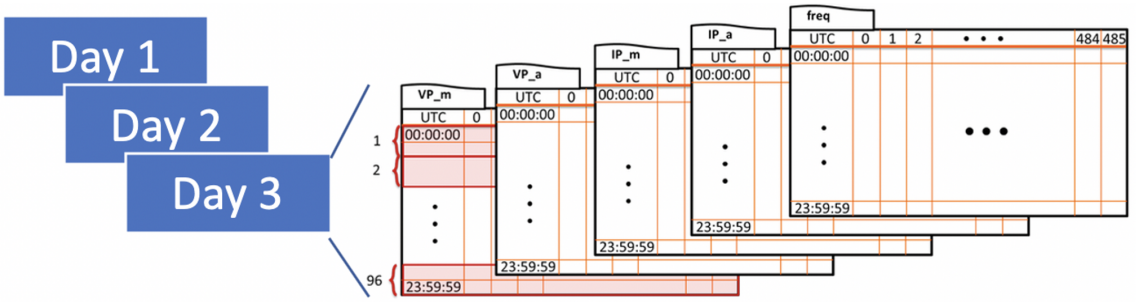

Figure 5.1: Storage format for PMU data. Raw data is saved into Parquet format (1 file equals 1 PMU attribute for 1 day), allowing fast column-wise reads for small time durations.

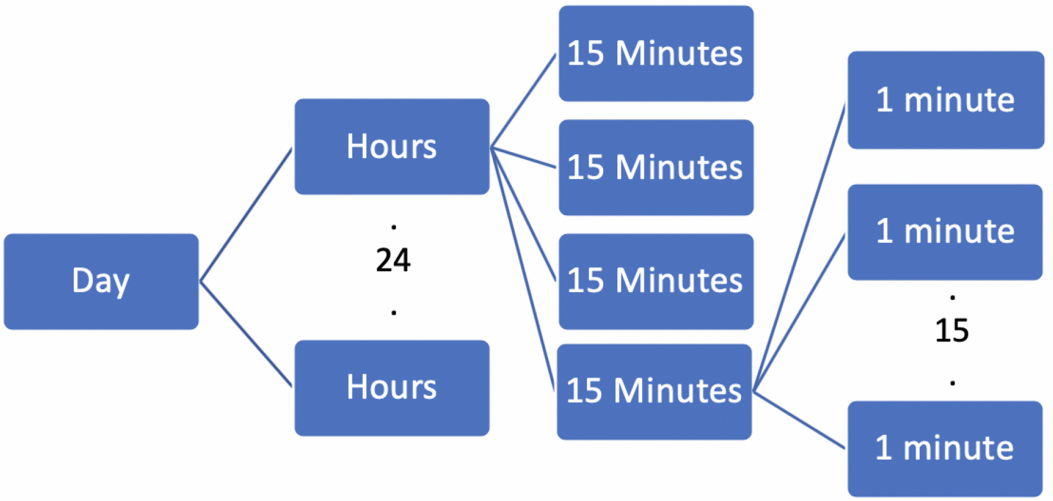

Figure 5.2: Storage of PMU data at different granularities. To enable long-duration data retrieval with fast access times, relevant statistics are computed at successive levels of granularity.

One of the advantages of using the Parquet file format for the storage of our time-series data is the fact that it offers several built-in compression/decompression algorithms. The performance of a compression scheme is evaluated as a trade off between size of the stored data and speed of writing/reading the data. To identify the best option for our dataset, we tested three different compression algorithms: *snappy* (which is the default option), *gzip*, and *brotli*. Table 5.1 shows the size in gigabytes of one parquet file containing all PMUs for one day based on the different

Table 5.1: Parquet file sizes. One parquet file contains one measurement type, for all PMUs for one day

	File size [GB]			
Measurement	**No compression**	**snappy (default)**	**gzip**	**brotli**
Voltage magnitude	2.26	2.01	1.87	1.84
Voltage angle	4.98	3.94	3.28	3.11
Current magnitude	4.47	3.41	2.87	2.74
Current angle	10.2	7.65	5.98	5.53

compression algorithms. This test is performed on four different measurement types (or attributes). These results show that brotli offers the highest compression ratio for every measurement type; however, the difference between brotli and gzip is not as big as the difference with snappy or the case without any compression. The second test performed aimed at assessing the speed (in seconds) of reading a parquet file based on the different compression algorithms. Table 5.2 shows the results when reading one entire column (that is, one PMU for one day) and 60 entire columns at a time (60 PMUs for one day). Here, we can see that in three out of four cases gzip is the fastest algorithm, with brotli being slightly slower. Since the difference in file sizes between brotli and gzip is very small, we chose to adopt the gzip algorithm for our database. Overall, storing one entire day of data for all PMUs and all 18 attributes results in about 50-70 GB of stored data.

Despite the relative speed and efficiency of our database structure, depending on the downstream application, processing such large amounts of data might result in prohibitively long reading times. For example, the interactive visualization of raw PMU data over long timespans becomes impossible when looking at several data streams. For this reason we also utilized a hierarchical aggregation scheme for computed PMU statistics as described in the next section.

Table 5.2: Reading speed of parquet files: reading 1 entire column / 60 entire columns

	Reading time of 1 column / 60 columns [s]			
Measurement	**Uncompressed**	**snappy**	**gzip**	**brotli**
Voltage magnitude	0.930/21.21	1.327/11.91	0.497/10.68	0.531/11.60
Voltage angle	0.617/16.14	1.650/20.69	0.982/21.29	0.690/15.74
Current magnitude	0.572/9.39	0.240/1.50	0.174/1.55	0.179/2.43
Current angle	0.807/34.92	1.410/22.78	0.895/26.41	1.01/22.52

5.2.2 Aggregating PMU statistics for fast querying

Our aggregation approach mimics the Parquet file structure for the raw data, only now storing aggregate computed statistics (Figure 5.2). For example, we store the average value of each attribute over different aggregation times (minutes, hours, etc.): these down-sampled time-series can be used for fast plotting of data over long periods of time, where sub-second resolution is not needed. The aggregation scheme we developed is modular and it can be used for the storage of any aggregate statistic in addition to the average, such as: minimum, maximum, standard deviation, signal-to-noise ratio, etc.

The decision to maintain data storage and access via Parquet files was chosen based on the following reasons: (i) Aggregated data files can be stored in the same directory locations as raw data, making administration straightforward. (ii) APIs previously developed to access Parquet files storing raw PMU data can be tweaked to additionally access Parquet files storing aggregated statistics. This maintains a consistent data access API for both raw and aggregate data. (iii) Even when conducting subsecond analysis, PMU measurements are almost always considered over time ranges as opposed to a single measurement/timestep snapshot. Storing aggregated

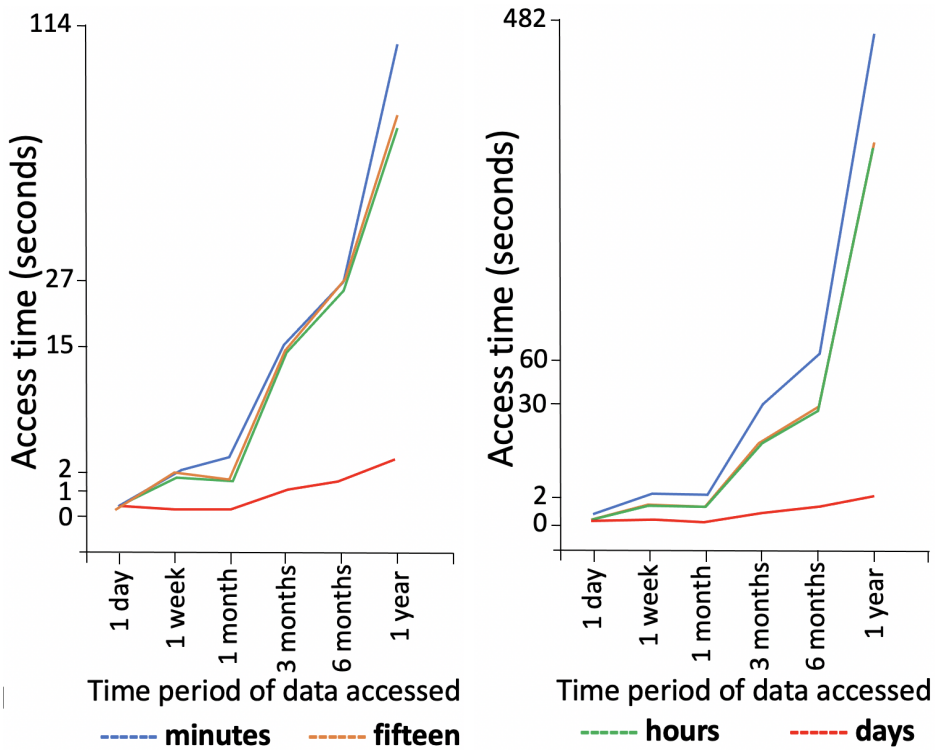

Figure 5.3: Access times for aggregated PMU data stored in Parquet format. The left plot shows access times of aggregated PMU data for two attributes (VPm and IPa) for one PMU (each line represents a different aggregation granularity). The right plot shows the same quantities when accessing two attributes for all PMUs. The x-axis indicates the time-length of the data accessed, while the y-axis shows the access time (in seconds). Figure courtesy of Anjana Arunkumar.

PMU statistics in column-wise format in Parquet files optimizes for the temporal queries. In contrast, SQL/NoSQL/time series databases would require implementing and maintaining indexes with additional overhead, thus slowing data retrieval.(iv) Besides raw data aggregation, we are able to compute aggregated statistics such as signal to noise ratio (SNR) over a sliding window and its standard deviation for all attributes, as well as positive and negative voltage sequences, simultaneously.

We consider four aggregation levels for PMU data– minute, hour, day, and week– by averaging over the granularity considered. Like before, we store each PMU's data to a single column, but each row now represents the respective time granularity considered. For example, by aggregating over minutes, each day's measurements are

reduced to 1440 rows; we store a week's worth of data using the minute granularity with 10,080 rows and a size of 30MB. Like the raw data, each attribute is stored in a separate Parquet file. Figure 5.3 shows how access time increases based on the time duration and the aggregation level—we note that minute and day granularities are sufficient to efficiently conduct multi-year data retrievals (the hour and fifteen minute granularities performing equivalently well, with a less than 3 second difference than minute access time at maximum duration, i.e., 1 year). In this way, PMUVis uses data aggregation to support interactive querying, in line with existing analytics platforms for sensor data (see [45, 46] and references therein).

Despite this approach being straightforward, it proved successful both for facilitating fast data retrieval as well as the retrieval of many PMUs for visualization.

5.2.3 Analyzing events using fast-Fourier transform

The PMU database structure we created represents a crucial step in enabling new PMU-based applications for the monitoring and control of power systems. As an example, in this section we describe a tool we developed to identify the source of an oscillation. In the next section, two study cases are presented to demonstrate how PMUVis can be used for real-time monitoring.

One of the most promising applications of PMU technology is to detect and monitor power system oscillations. Generally, oscillations are triggered by system faults or generator misoperation and they can be temporary (the oscillation is damped and dies off after a couple of seconds) or sustained. Sustained oscillations, also called forced oscillations, happen when a generator (or a group of generators) has imperfect or inaccurate real-time control and it continuously injects active or reactive power that in turn excite the oscillatory modes. This type of oscillation can be stopped only when the generator is either taken offline or its controls are adjusted. Oscil-

lations can be observed from many different quantities such as voltage magnitude, frequency, and power injection. Oscillations propagate throughout the system and can be observed from many different buses around the source. An important task for power system operators is to identify the source of the oscillation, that is, the bus at which the generator is exciting the system and causing the oscillatory behavior. Once the source is localized, corrective actions can be taken by the operators to fix the issue—taking offline or controlling the generator. This is one reason we perform the ego PMU identification step.

We propose a method to localize the source of an oscillation that fully integrates within the visualization platform and leverages many of its capabilities. When looking at bus voltage magnitudes during the oscillatory events in our dataset, it can be seen that the oscillations are the most prominent and clean at buses close to the source. As we get further, the voltages and the oscillations become more noisy. This is due to the fact that as you get further from the source, the effect of the oscillation is damped and can be attributed to other phenomena and disturbances that are happening simultaneously in the system. Based on this observation, we perform an FFT analysis on the voltage signals of PMU from different locations. By comparing the FFT magnitude at the oscillation frequency, we can get an idea on how close each bus is to the source. An example of this type of analysis and its implementation within the platform is described in the following sections.

5.3 Case studies

To help demonstrate PMUVis, we show how it can be used to analyze oscillatory power grid events in two case studies. Low-frequency oscillations are always present in large, interconnected transmission grids [47]. Wide-area system monitoring using synchrophasor data from PMUs is often used to control these oscillations using

anomaly detection techniques [48]. System operators must monitor the system's ability to damp such oscillations, and reduce power transfer if required, while maintaining awareness of relevant events and information on other parts of the grid not directly under their control.

5.3.1 Case study: forced oscillation

We first analyze a historical long-duration forced oscillation event. The specific steps are shown in Figure 5.4.

(1) Selecting the event in PMUVis' control panel highlights its corresponding ego PMU #122 in the network panel. Per the operator report, a ∼2 hour-long oscillation was noticed at the Yearling substation (where PMU #122 is located) with a beginning timestamp of 20:44. (2) To create a subgraph suitable for analyzing the oscillation's propagation, additional PMUs are selected via the add hop functionality and manual selection/deselection. (3) In total, 15 PMUs are selected—all within 1–3 hops of PMU #122. In Figure 5.4, the graph's display settings are toggled to highlight the subgraph's equipment specifications (types of line, loads, etc.).

(4) To see an overview of the oscillation, we load the heatmap for a ∼4.5 hour time period starting at the event's beginning, displaying the
signal-to-noise ratio (SNR) for the VPm attribute (positive sequence voltage magnitude) for each selected PMU aggregated into 15-minute blocks. Rows are ordered based on PMU hop distance and similarity to the ego PMU #122 (the top row). Color fluctuations across a row indicates a PMU is experiencing varying SNR values. Several PMUs close to PMU #122 have increased fluctuation in the left half of heatmap (during the ∼2 hours that the oscillation occurred). We select timestep 21:40, which shows a sharp increase in SNR for PMU #122, as a place for further investigation.

(5) To examine the oscillation with non-aggregated PMU data, we select four

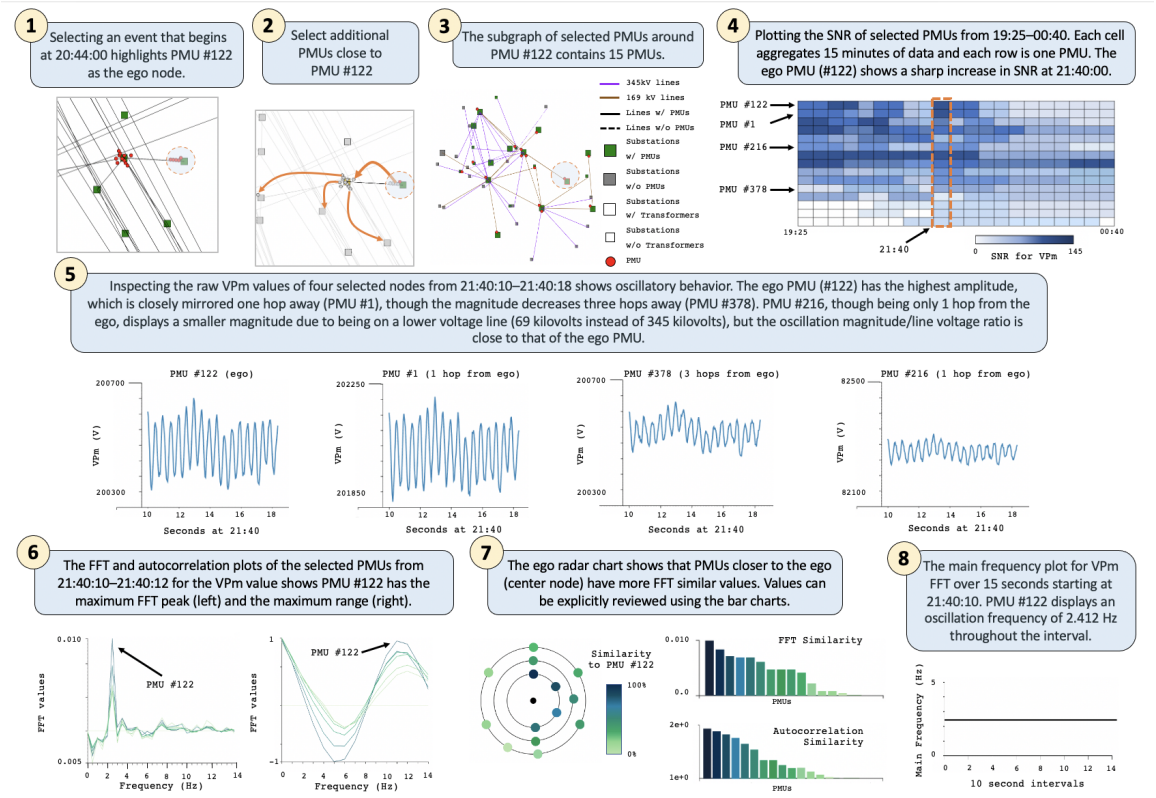

Figure 5.4: In the Forced Oscillation case study, (1) PMU #122 is selected as the event's ego PMU. (2,3) After selecting additional PMUs and (4) identifying 21:40 as a timestep for subsequent analysis, (5-8) additional charts show how the oscillation propagates out from the ego PMU to nearby PMUs. V: volts, VPm: positive sequence voltage magnitude. Figure courtesy of Anjana Arunkumar.

PMUs from the heatmap and display their raw VPm values in the line chart panel. Oscillatory behavior is clearly visible in the ego PMU #122 and in PMU #1 (one hop away). PMU #378, which is three hops away, shows a reduced oscillation with smaller magnitude. As PMU #216 is on a lower voltage line, despite being one hop from PMU #122, it also shows a smaller magnitude oscillation; however, the ratio between its oscillation magnitude and voltage rating is close to that of the ego PMU.

(6) We next load the 15 selected PMUs in the ego panel. The FFT chart confirms that PMU #122 has the highest FFT and autocorrelation peaks, helping to confirm it is at the oscillation's source. (7) Comparing the ego PMU to other selected PMUs using the ego radar chart and similarity bar charts shows that, generally speaking,

PMUs closer to the ego (i.e., with fewer hops) tend to be more similar. This importantly demonstrates the oscillation's effects lessen as they egocentrically propagate from the source throughout the network. (8) Finally, the main frequency plot shows a steady main frequency of ∼2.4Hz, indicating the oscillation at the Yearling substation is indeed a constant, long-duration event.

5.3.2 Case study: damped transitory oscillation

The first case study considered a long-duration, sustained oscillation. Here, we consider a short, damped transitory oscillation event that was not included in our dataset of historical operator reports, but was instead discovered while analyzing the previous case study. Specific steps are shown in Figure 5.5.

While exploring different configurations with the heatmap, we noticed that PMU #169 had an unexplained color fluctuation. (1) Selecting this PMU as the new ego with a selection of 8 additional PMUs, and reloading the heatmap at a granularity of one-second-per-cell for the IBm attribute (current magnitude of phase B) clearly shows a fluctuation for PMU #169 that also seems to affect PMU #278 (one hop away).

(2) This is also seen in the line charts panel, where the oscillation is clearly visible in PMU #169 and with a smaller magnitude in PMU #278. The oscillation is difficult to see in PMU #19, located three hops away.

Examining the oscillation in the event panel shows this is a well-damped transitory current oscillation—one that lasts for approximately 2 seconds. (3) High overlap of FFT values is seen for immediate neighbours of the ego PMU; on further analysis, we note that neighboring PMUs are located on low voltage lines (69kV), and are connected to step-down transformers, accounting for the highly-localized oscillation (which is also symmetrical over IAm and ICm). (4) The main frequency plot (using 10 second time windows) shows only a single spike. Such damped, transitory power oscil-

lations are exhibited by current signals when transient events such as line/generator tripping or high load fluctuations occur [49]. These events tend to quickly die out, with the system returning to an "at rest" state comprised of very slow oscillations that can be filtered out using a high-pass filter.

Current signals contain decaying DC components, and exhibit such damped, transitory power oscillations when transient events such as 'earth faults' occur [49], i.e., line/generator tripping or high load fluctuation. The main frequency oscillation values for the ego PMU, calculated over 10 second time windows (for the time interval chosen to construct raw IBm time charts), show that the system is almost completely at rest — i.e., comprised of very slow oscillations that can be filtered out using a high-pass filter—until the transient current oscillation intensifies. At this time, we see a sharp increase in the main frequency of that PMU; the event quickly dies out as the oscillation damps, with the main frequency returning to its original state. High overlap of FFT values is seen for immediate neighbours of the ego PMU; on further analysis, we note that neighboring PMUs are located on low voltage lines (69kV), and are connected to step-down transformers, accounting for the highly-localized oscillation (which is also symmetrical over IAm and ICm).

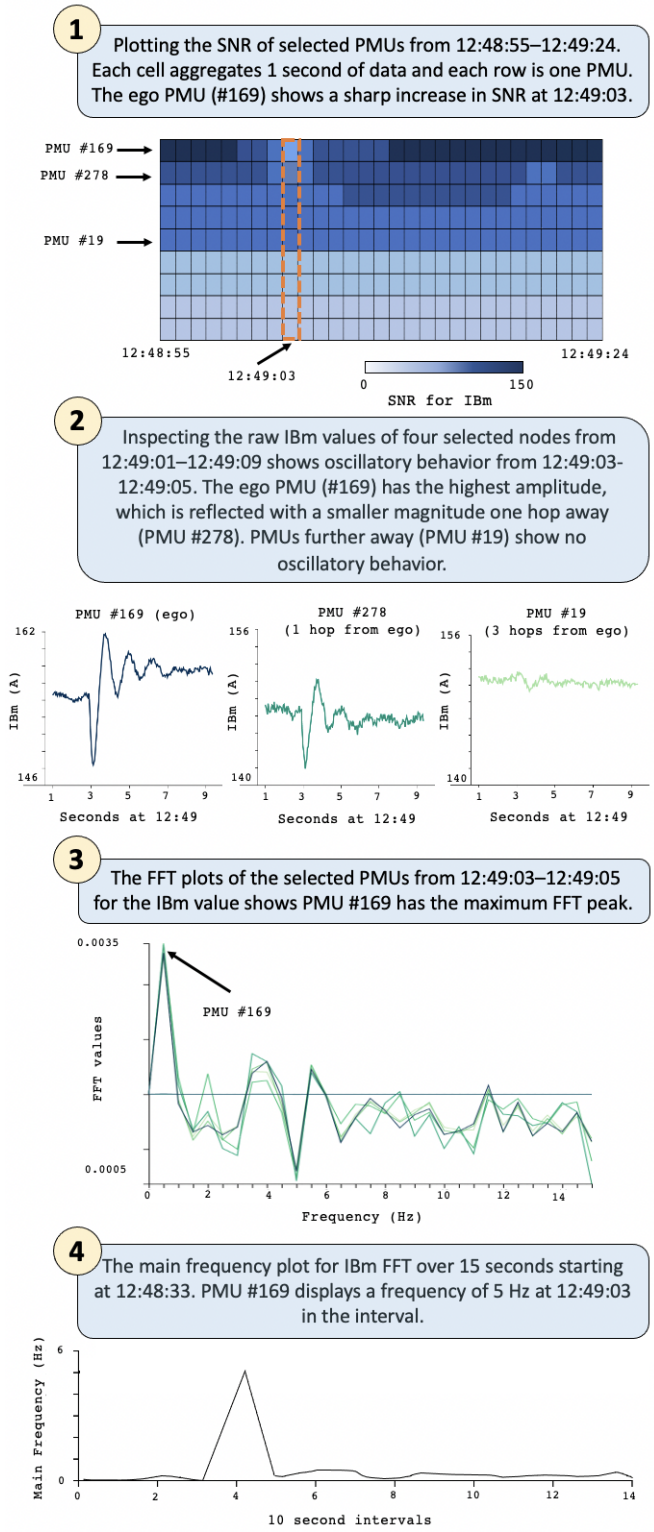

Figure 5.5: In the Damped Transitory Oscillation case study, (1) PMU #169 (top row of the heatmap) experiences a transitory (sub-second) oscillation, which can be seen in the (2) line charts, (3) the FFT plot, and (4) the main frequency chart. A: amperes, IBm: current magnitude of phase B. Figure courtesy of Anjana Arunkumar.

Chapter 6

SYNTHETIC LOAD GENERATION USING GENERATIVE ADVERSARIAL NETWORKS

6.1 Background

The PCA-based generative model described in Chapter 4 presents three main limitations: 1) the length of the synthetic data is limited to the length of the real dataset used, 2) the generated data only captures the main load behaviors and might be missing more nuanced patterns, and 3) the spatial correlation is implemented via ad-hoc correction factors.

In this chapter, we present a more advanced generative model which uses a ML technique called conditional generative adversarial network (cGAN) [50]; this represents a powerful and flexible framework for the training of a generative model. Simple GANs [51] and conditional GANs have been used in the literature for the generation of PMU voltage data [52], renewable energy profiles [53], and residential energy consumption [54] but not for transmission-level load data. We train a cGAN to generate realistic, synthetic week-long time-series load profiles at a resolution of one sample per hour. The generation of synthetic data can be conditioned on specific labels indicating the season of the year or the type of load profile desired in order to meet the user's specific requirements.

As we will see in the rest of the chapter, conditional generative adversarial networks not only overcome all the limitations of the PCA-based approach, but they also present some unique advantages.

6.2 Generative adversarial networks

6.2.1 Basic GAN

Generative adversarial networks are a novel ML framework in which a generative model (or generator) is trained by making it compete against a discriminator. The goal of the generator G is to capture the distribution of the real data p_r, while the discriminator D is trained to distinguish the real data from the synthetic data produced by the generator. The generator is trained to learn a mapping $G(\boldsymbol{z}; \theta_g)$ from a known noise distribution p_z to p_g, where G is a differentiable function represented by a multilayer neural network with parameters θ_g and $\boldsymbol{z}$ is a noise vector sampled from p_z. Given a data sample $\boldsymbol{x}$, the discriminator determines the probability $D(\boldsymbol{x}, \theta_d)$ that the sample came from the real data distribution p_r rather than from the generator p_g. The training of D and G is represented by a two-player minimax game with the following objective function:

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{6.1}$$

Here, the discriminator is maximizing the likelihood of data x when sampled from p_r and minimizing it when sampled from p_g, while the generator has the opposite goal of maximizing the likelihood of the samples from p_g. The optimal solution is obtained when the discriminator assigns a probability of 0.5 to all samples, meaning that it cannot distinguish between real and generated data.

6.2.2 Conditional GAN

Conditional generative adversarial networks (cGAN) are an improvement on the basic GAN framework which allow for a more targeted generation of synthetic data. The conditioning is performed by labelling the real data and then providing this label

$\boldsymbol{y}$ as a further input to both the generator and the discriminator. By doing this, the generator learns the conditional distribution p_g over $\boldsymbol{x}|\boldsymbol{y}$ and the generation process can be guided by requesting synthetic data belonging to a specific class. The final structure of a generic cGAN can be seen in Fig. 6.1.

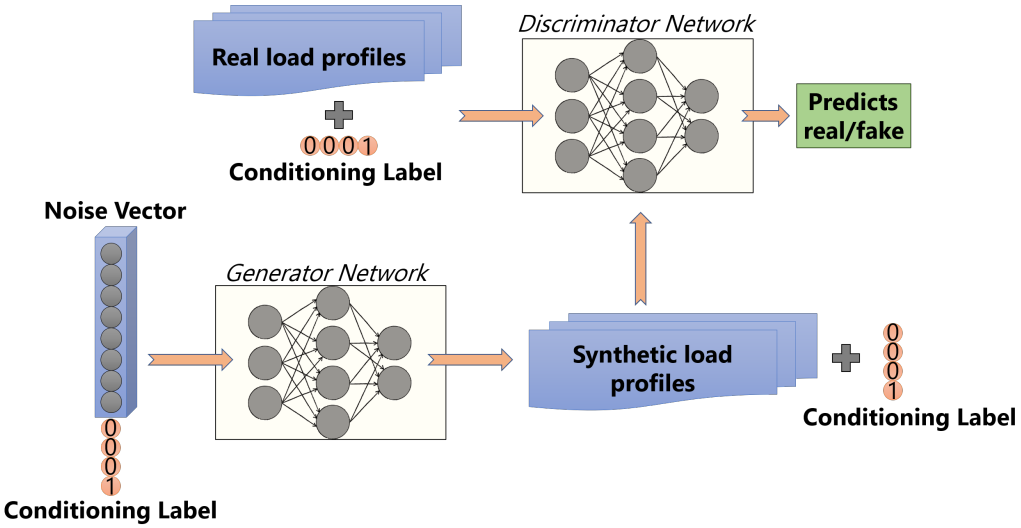

Figure 6.1: Structure of a conditional GAN.

6.3 Dataset description

As described in Chapter 5, the foundation of this project is a large dataset of real PMU data obtained from a utility in the USA. In particular, based on the system topology and the location of the measurement devices, we identified 12 load buses whose lines are entirely monitored by PMUs. The net injection at these buses represents the load demand and this allowed us to compute the active and reactive power of the 12 loads with a resolution of 30 samples per second for two consecutive years (2017 and 2018). As discussed in the introduction, the focus of this work is the generation of week-long profiles at a resolution of 1 sample per hour for a total of 168 hours. The raw, PMU-speed load data is then downsampled by computing the hourly load average and broken into weeks. When combining all weekly profiles from all 12

loads, the final dataset is a 1158x168 matrix. The total number of profiles (1158) is lower than the theoretical maximum (2 years x 12 loads x 52 weeks = 1248) because profiles containing any data dropout have been discarded. Each week-long profile is normalized by dividing it by the average load over the week; the entire dataset is further normalized between 0 and 1 for the training of the cGAN.

6.4 Load characteristics

Different factors influence the way system loads behave over time. To appropriately generate realistic synthetic load profiles for a given application, these elements need to be captured and modeled by the GAN. When looking at the week-long time-series data described in the previous section, two main driving factors can be identified: time of the year, and type of load.

The differences between load profiles due to seasonal changes in energy consumption can be easily visualized. Figure 6.2 shows four week-long profiles for a load, across the four different seasons. In winter and fall, the load pattern presents two daily peaks, one in the morning and one in the afternoon. During spring and especially summer, the load is more regular, with a large spike during the day and dips at night. This type of behavior can be observed in a more or less pronounced manner across all the loads in our dataset. For this reason, the season of the year to which a profile belongs is an important indicator (label) of the expected profile.

At the transmission level, each load represents the aggregate demand of one or more distribution feeders. Thus, the behavior of a load is given by the sum of all the customers at the distribution level that it serves. Because of this, the second main factor that determines the temporal profile of a load is its composition in terms of residential, commercial, and industrial portions since each of these types of loads tend to have very distinctive patterns. In our dataset, we have observed two classes of loads

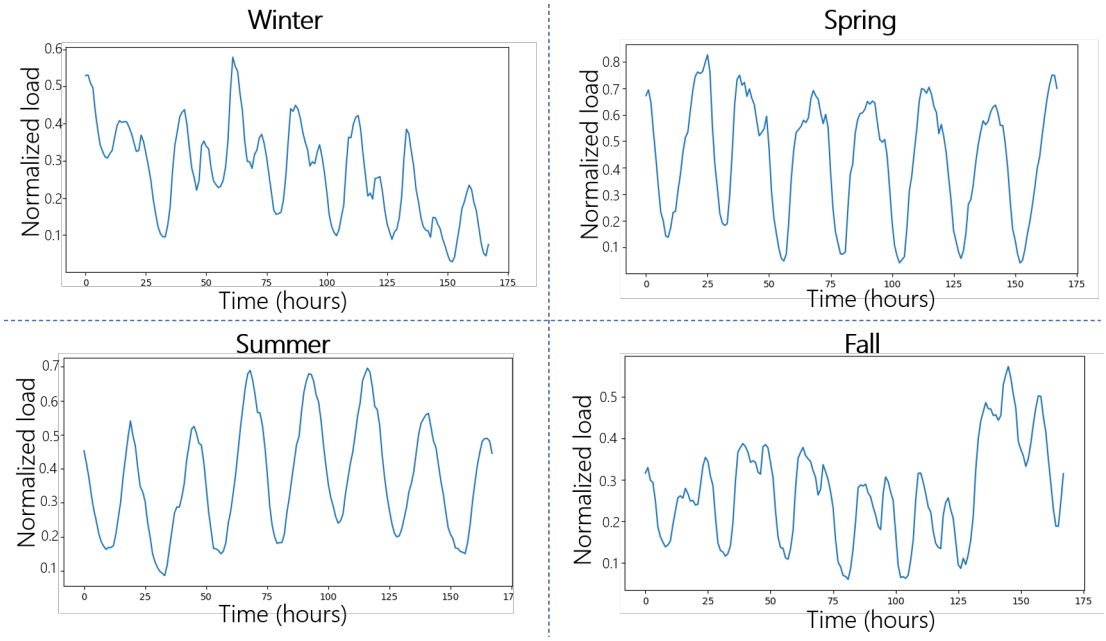

Figure 6.2: Examples of real load profiles; different seasons present different patterns.

with very distinct behaviors: loads that are mainly residential and/or commercial and loads that are mainly industrial. Figure 6.3 shows some selected examples: on the top and bottom left are mainly residential loads from winter and summer respectively, while on the right are winter and summer profiles of a mainly industrial load. As we can see, loads that are mainly residential have very regular and predictable patterns, whereas industrial loads do not necessarily follow recognizable daily patterns.

We used a k-means clustering algorithm to label each load as mainly residential or mainly industrial. When using 3 clusters, two main groups of loads are identified, each containing five and six loads, while one single load is grouped separately. Figure 6.4 shows the percentage composition for each of the twelve loads in terms of commercial, industrial, and residential components. The last column indicates the cluster to which each load is assigned using 3-means clustering. We observed that the loads where the top two factors in terms of percentage composition are residential and commercial are clustered as one, while the loads in which the industrial component is first or second

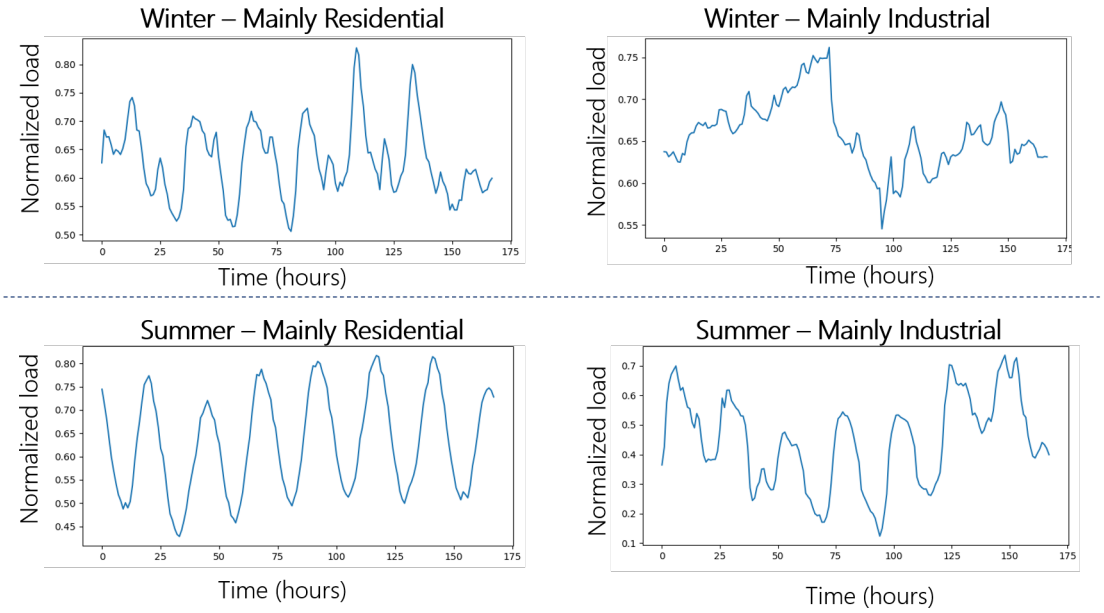

Figure 6.3: Examples of real load profiles. Top row: mainly residential winter load (left) and mainly industrial winter load (right). Bottom row: mainly residential summer load (left) and mainly industrial summer load (right).

are grouped as another cluster. Thus, for training purposes, six loads are labeled as mainly residential and six as mainly industrial.

6.5 cGAN for synthetic load profiles

6.5.1 cGAN model

In this section, we will describe the implementation of the cGAN and its training process. Convolutional neural networks (CNNs) are chosen for the discriminator and the generator for their ability to learn multiple spectral properties of the data. While similar in size and complexity, the two models present some differences.

The discriminator receives two inputs: first, the raw time-series is processed by two convolutional layers, then the flattened output is concatenated to its label and fed to three fully-connected layers. The output of the discriminator is a scalar indicating if a profile is real (1) or fake (0). In the generator, the two inputs are the load

Load	Composition (%)			Cluster
	Commercial	Industrial	Residential	
1	39	23	38	1
2	35	13	52	1
3	n/a	n/a	n/a	1
4	16	3	81	1
5	17	5	78	1
6	33	4	63	1
7	n/a	n/a	n/a	0
8	12	24	64	2
9	21	39	40	2
10	5	71	24	2
11	31	38	31	2
12	3	78	19	2

Figure 6.4: Percentage load composition of the computed loads and results of the k-means clustering.

label and a 25-dimensional Gaussian noise vector [1] . These are concatenated and fed to three fully connected layers and the output is up-sampled via three transposed convolution layers. The final output is a synthetic load profile whose characteristics match the input label. The overall architecture of the cGAN is illustrated in Fig. 6.5.

As we have seen in the previous section, two characteristics of loads are used as labels for the conditional GAN: the season and whether a load is mainly residential or industrial. The factors are represented as one-hot encoded vectors, i.e., the seasons are represented via the following four labels: (1 0 0 0) for winter, (0 1 0 0) for spring, (0 0 1 0) for summer, and (0 0 0 1) for fall. Similarly, the load type is encoded as: (0 1) for mainly residential and (1 0) for mainly industrial.

The training process is performed by iteratively training the discriminator to distinguish between real and generated data and the generator to create realistic-looking profiles. It is to be noted that the discriminator is trained twice at every iteration in order to give it an advantage against the generator; this forces the generator to produce better samples. Figure 6.6 shows how the training process progresses as the

[1]The dimension of the noise vector depends on the GAN architectures chosen and the desired output vector length.

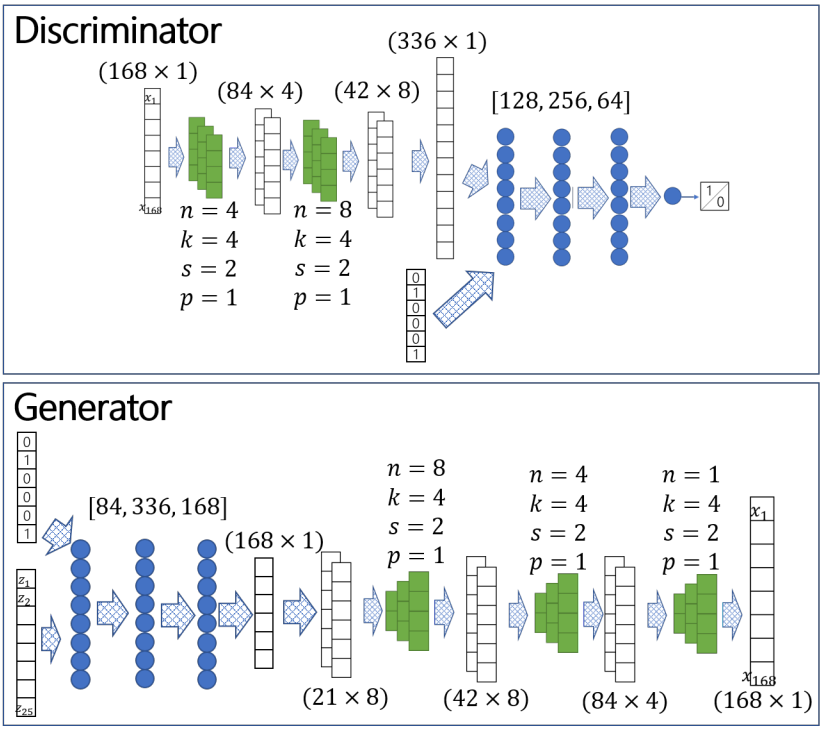

Figure 6.5: Structure of the cGAN used for the generation of week-long profiles.

epochs proceed. The three curves represent the average discriminator prediction at each epoch for three sample datasets: real data used during training (blue), real data never used during training (validation data, green), and fake data created by the generator (orange). We can see that at the beginning the discriminator easily distinguishes between real and fake data, assigning high values to both real datasets and low value to the generated data. As training progresses, the generator improves and the discriminator is unable to differentiate between the two data sources. At around 3000 epochs, the training converges: the discriminator assigns very similar values to all three datasets. It is interesting to notice that some overfitting is happening (the blue curve reaches 0.53) but it is not very significant. More importantly, the discriminator assigns the same values to both the generated data and the validation data; this means the output of the generator matches the real data.

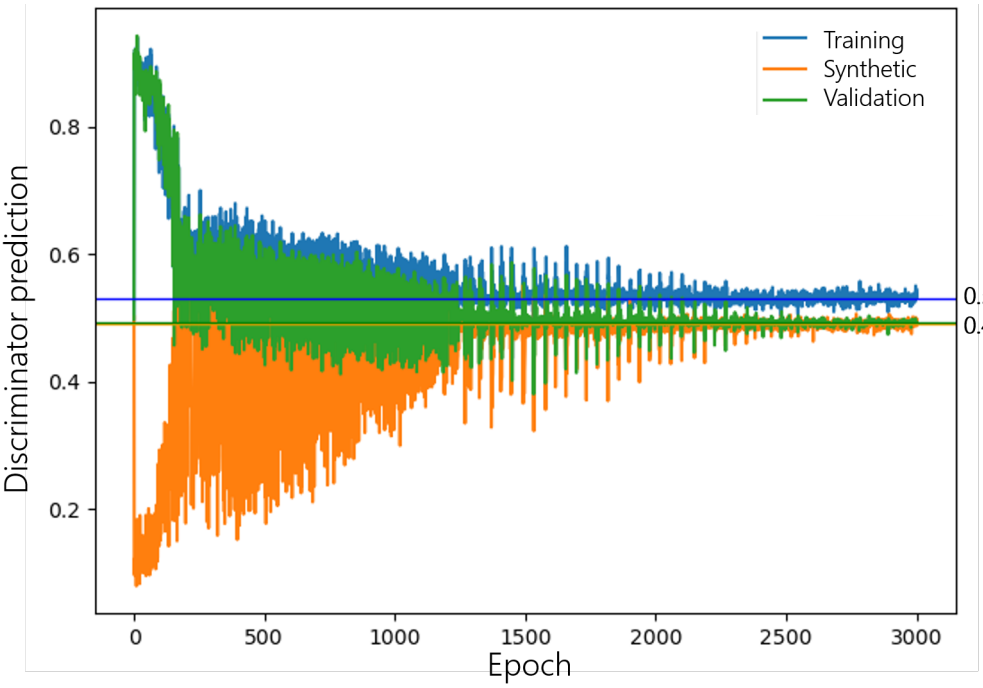

Figure 6.6: Training progress of the cGAN based on the predictions of the discriminator at each epoch. The blue curve shows the average prediction over a batch of real training profiles. The green curve represents real validation data and the orange one predictions on generated data.

6.5.2 Data generation

Once the training process is terminated, the generator can be used to create any number of synthetic profiles. Based on the required data type, the user only needs to define the appropriate label and generate a noise vector according to the predetermined distribution; feeding these to the generator will result in a synthetic load profile. As an example, in Fig. 6.7 two synthetic summer profiles (right) are compared to two randomly selected real profiles (left) of the same label. The blue profiles (top) correspond to a mainly residential load and the green plots (bottom) to a mainly industrial load. It is important to notice that while the synthetic profiles present all the same characteristics as real data, they do not simply repeat real profiles.

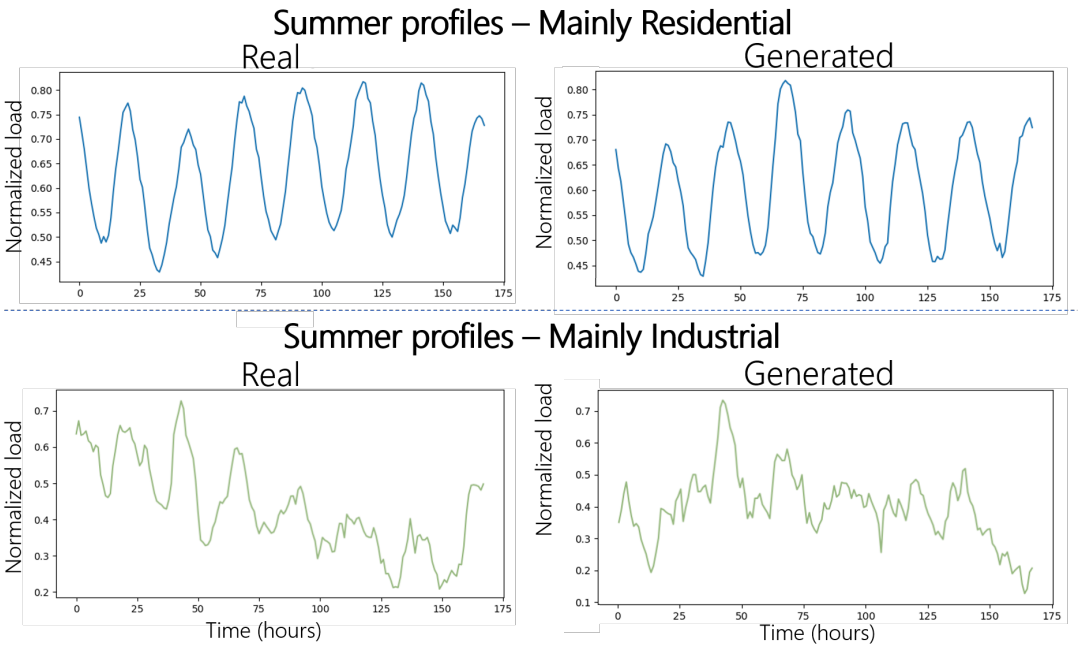

Figure 6.7: Comparison between some real summer profiles (left) and generated profiles (right). Top: mainly residential load. Bottom: mainly industrial load.

6.6 Evaluation of synthetic data

While visual inspection does not yield clear differences between real and synthetic profiles, a quantitative analysis is required to verify that the generator captures all of the characteristics and behaviors present in the real data. In this section, we present multiple tests to validate the quality of the synthetic data.

6.6.1 Wasserstein distance

As explained in Section 6.2, the goal of the generator is to learn a mapping function from the known noise distribution to the distribution of real data. Training is successful when the distribution of generated data matches that of the real data. Wasserstein distance is a measure of distance between two distributions and it can be used to quantitatively assess how close the distributions of generated and real data are.

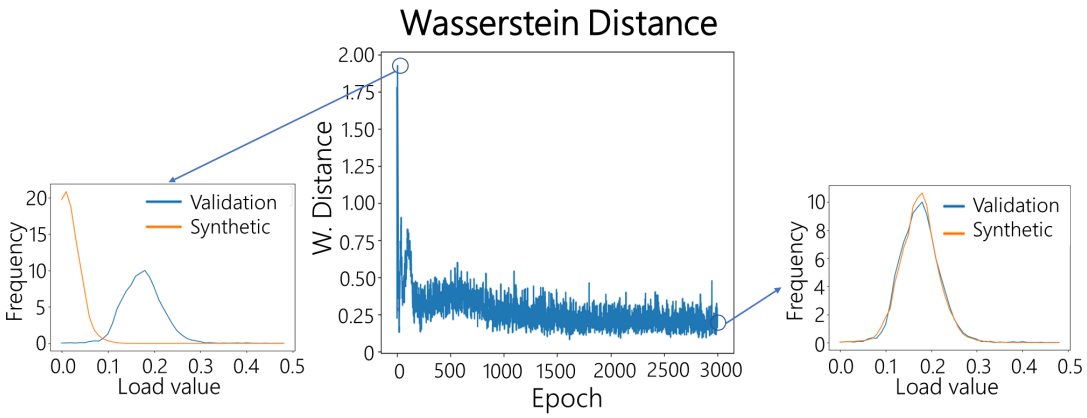

Figure 6.8: Center plot: Wasserstein distance between real and generated data as a function of the epochs. Side plots: comparison between the distribution of real data (blue) and generated data (orange) at epoch 0 (left) and epoch 3000 (right).

The center plot in Fig. 6.8 shows the Wasserstein distance computed during training at each epoch between a batch of generated data and a batch of validation data. It can be seen that as the training progresses, the distance between the distributions tends to zero. This can be further seen by looking at the two smaller plots on either side. The plots to the left and right show the histograms (empirical distributions) of the real data (blue) and that of the generated data (orange), at epoch 0 (left plot) and at epoch 3000 (right plot), respectively. While initially the two distributions are very different, at the end of training the generated data almost perfectly matches the distribution of real data.

6.6.2 Power spectral density

An important characteristic of time-series load data is its periodicity. Because loads are tied to the routine and behavior of people, they present different recurring patterns. One way to identify these periodicities is by looking at the power spectral density of the time-series data. Figure 6.9 shows the power spectral density for real data (blue) and generated data (orange). It can be seen that the two plots match very closely, confirming that the generated data captures the periodic behavior of

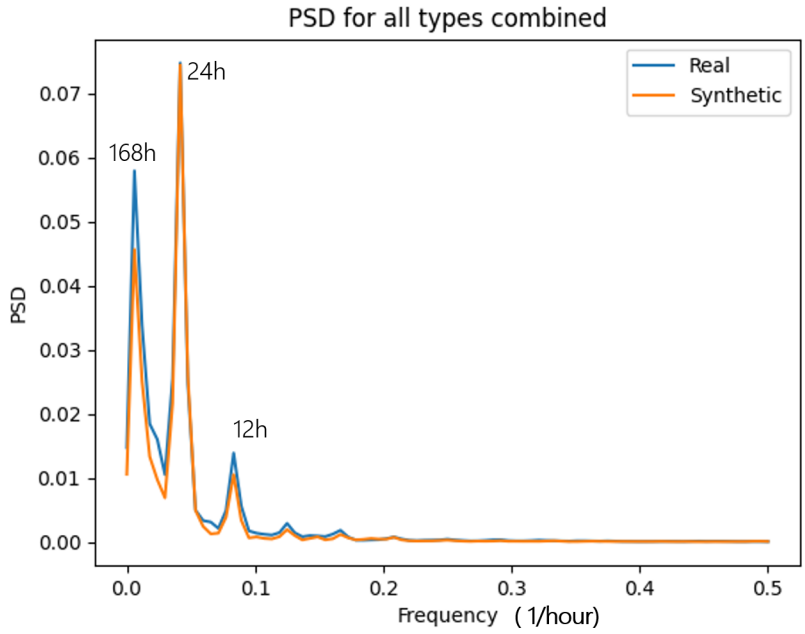

Figure 6.9: Comparison between the power spectral density of real data (blue) and generated data (orange).

real data. It is also interesting to look at the various peaks that appear in spectral density: in particular, the highest peak, which occurs at a frequency of 0.04/hour, corresponds to a time period of 24 hours, thus representing the daily load cycle.

6.6.3 Forecasting application

The main goal of this work is to create a mechanism for the generation of realistic synthetic load data that can be used by researchers when real data is either not available or not rich enough. In the next two sections we present the results of two example applications that show that the synthetic data successfully captures the behavior of real data and that it can be used for downstream applications.

One of the most common uses of time-series load data is the development of the forecasting algorithms needed for power system operations and markets. While many different techniques are used, often in combination, one of the latest advancements

Table 6.1: Comparison of the forecasting error between generated and real load data, for summer and fall residential profiles.

Load label	**Testing data**	**Percentage Error**	
		Mean	Std. Dev.
Summer - Residential	Synthetic	4.37	5.26
	Real	5.30	5.17
Fall - Residential	Synthetic	5.82	7.10
	Real	5.92	5.17

in ML-based forecasting is a class of recurrent neural networks called long short-term memory (LSTM). Because feedback loops are present, the LSTM architecture is able to process sequences of data (such as time-series data) maintaining a memory of the previous inputs. To verify the quality of our load data generator, we trained an LSTM on a batch of synthetic data and then tested the learnt model on the real data.

An LSTM network with three layers and 48 units per layer is trained to predict the value of a load at one point in time, given the previous 48 hours (48 points). This model is trained on two separate datasets independently: synthetic mainly residential summer profiles and synthetic mainly residential fall profiles. Each of the datasets consists of 1200 week-long profiles generated using the trained cGAN according to their respective labels. To evaluate the performance of the LSTM, for each of the two load types, the trained models are used to predict the load values of two batches of profiles: new generated data and real data of the same class. The percentage error between the forecasted value and the actual value is computed for each profile in a batch and the first and second moments are computed. Table 6.1 summarizes the results of the forecasting test for the two types of loads (summer and fall residential). In both cases we can see that even though the model was trained only on synthetic data, the error when applied to real data is very comparable. In general, this suggests

that a user could train a ML model on our synthetic data and be confident that it would still capture the characteristics of real data.

6.6.4 Optimal Power Flow

The synthetic data is also tested to verify that the generated profiles can be correctly mapped to a power system model. One way to check this is to ensure that all the resulting load cases form a feasible AC power flow.

This test is performed by first generating individual, week-long profiles for each load in the Polish test case: this system model has 2383 buses and 1822 loads. Two datasets are generated: one corresponding to a winter week and one for a summer week. Each of these profiles is mapped to the Polish system loads: since the base case of the Polish system is a peak case, the profiles are matched so that the peak of each profile corresponds to the base case value. AC optimal power flow (OPF) is then run on each case corresponding to each of the 168 hours of the week. The results showed that OPF converged in every case to a solution with bus voltages and generator outputs within their predefined limits.

The trained generative model and an Appendix with extra figures and results can be found at [55].

Chapter 7

GENERATION OF MULTI-RESOLUTION SYNTHETIC LOAD DATA

7.1 Introduction

One of the two main goals of the work presented in this dissertation is the design of generative models for bus-level time-series load data. As described in the introduction, such data is crucial for virtually every power system study: loads depend on factors which are external to the electrical grid but they are arguably the main driver of the system's behavior. Depending on the specific applications, the requirements in terms of load data characteristics vary greatly. For example, for studies which have long time horizons, such as transmission expansion planning or market related research, data for very long time scales is required (months or years) but at very low sampling rates. On the other hand, for dynamics or stability studies, only few seconds or minutes of load data are needed but at much higher sampling resolutions.

In chapter 6, it was shown that GANs offer a flexible and powerful learning framework for the modeling of time-series load data. Moreover, the large PMU database described in Chapter 5 contains high-resolution data (30 samples/second) for multiple consecutive years; this allows to capture load behaviors at very different time-scales. In this chapter, we combine our previous results to design a multi-resolution generative scheme which can generate synthetic load data for any use case or downstream application. The goal is to develop a tool capable of generating large amounts of time-series load data at any time-resolution and for lengths of time ranging from a few seconds to a year.

To achieve this, we propose a scheme to down-sample the raw load data at dif-

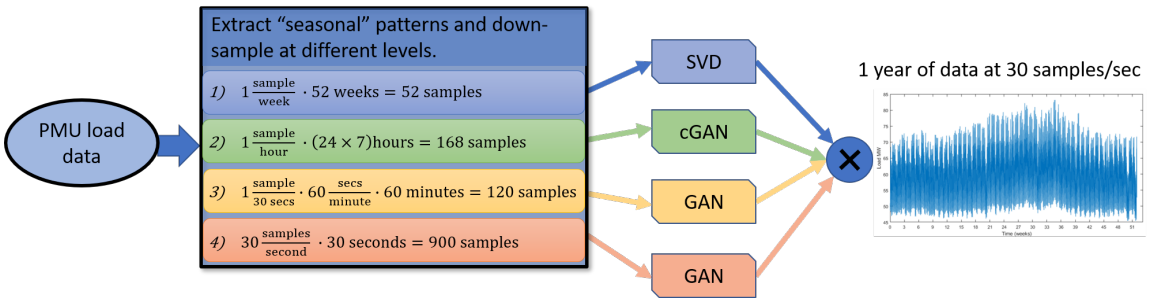

Figure 7.1: Overview of the generative scheme.

ferent levels of resolution and length; each level is designed to capture different load behaviors and characteristics. Subsequently, individual generative models are trained and optimized for each of the aggregation levels. Finally, we develop a process to combine profiles from different levels in order to obtain the desired sampling resolution. The successfully trained generative scheme has been compiled into a python-based graphical application which can be used to create any amount of realistic time-series load data according to user-defined requirements.

7.2 Overview

7.2.1 Multi-resolution generative scheme

Figure 7.1 shows an overview of the complete generative scheme used to achieve the generation of multi-resolution and multi-length synthetic load profiles. The process consists of four steps:

1. The raw PMU data (voltages and currents) is used to compute load consumption at different buses;

2. The PMU-speed load data is down-sampled at four different levels: 1) year-long profile at 1 sample/week, 2) week-long profile at 1 sample/hour, 3) hour-long profile at 1 sample/30 seconds, and 4) 30-second-long profiles at 30 samples/second;

3. A generative model is trained individually for each level;

4. The synthetic data generated using the trained models is combined to obtain any desired time-length and sampling resolution.

The four aggregation levels have been chosen as a compromise between having profiles of comparable size at each level while dividing the data in such a way that each level captures different and independent behaviors. The top level captures yearly seasonality, the second level captures the daily load changes, while the third and fourth levels model faster load variations. Regardless of the choice of aggregation levels, we propose a scheme to generate data at any arbitrary resolution from the highest limit of 30 samples/second to any lower sampling rate.

Moreover, as described in the previous chapter, it is important to train the generative models to capture the distinctive behaviors due to the load type (mainly industrial versus mainly residential) and the time of the year. However, these differences in patterns are only observed over longer time periods (that is, only in the higher aggregation levels); thus, as we will describe later, only the generative models of the top two levels are designed to generate profiles conditioned on these labels.

7.2.2 Dataset description

While in Chapter 6 the focus was on week-long profiles sampled at 1 sample/hour, here we are interested in capturing all time-scales and time-resolutions made available by our PMU dataset. For this reason, leveraging the PMUVis platform, we computed the load value of the twelve available loads for two consecutive years (2017 and 2018) at the highest resolution available of 30 samples/second. At this resolution, one year of data corresponds to roughly 9.5×10^8 points. This highlights the fact that it would be very impractical to learn a single generative model that is able to capture the

characteristics of such a dataset. Instead, as described in the introduction to this chapter, we can leverage the fact that loads present distinct behaviors at different time scales to train separate models for different time-resolutions. In addition to making the problem more tractable, this approach also provides higher flexibility for the generation of datasets of specific lengths and resolutions based on the different down-stream applications for which the load data is required. In the next section, we describe the process of aggregating and down-sampling the load profiles at different levels and we present the specific generative model chosen.

7.3 Load Modeling

7.3.1 Year-long profiles

Year-long profiles capture the monthly and seasonal patterns. For a load-bus, one year of data is down-sampled by taking the average load every week; the resulting profile is 52 samples long. Since 12 loads are available over two years, a total of 24 year-long profiles are obtained. At this time-scale and resolution, it is possible to observe clear differences between the profiles of mainly residential loads and mainly industrial loads. Figure 7.2 shows a comparison between two randomly selected profiles belonging to each of the two load classes; each profile starts on the 1^{st} week of January. It can be noted how mainly residential loads present a very regular pattern through the year, with demand increasing during summer (highest peak) and winter. On the other hand, mainly industrial loads are less strongly dependent on the time of the year, and they vary less. Given these distinctive behaviors, the generative model for this level must be designed so that synthetic data can be created based on the load type label.

Unlike at the lower levels, at this level the number of real year-long profiles is

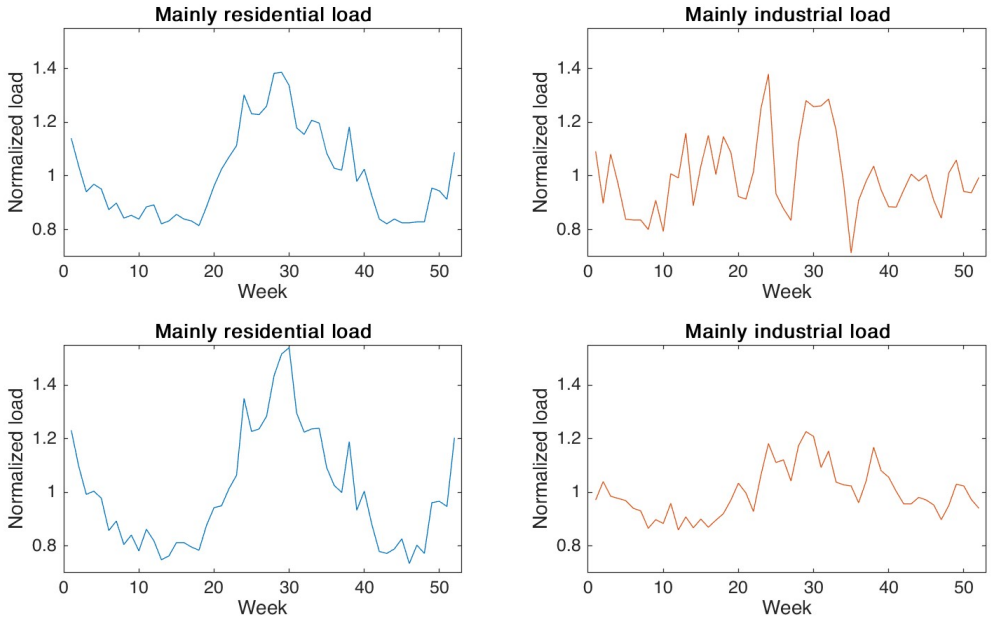

Figure 7.2: Comparison between real mainly residential year-long profiles (blue, left) and mainly industrial year-long profiles (red, right).

not sufficient for the training of a GAN. As we described in the previous chapter, six out of the total twelve loads are classified as mainly residential and the other six are mainly industrial. For this reason, at the top level, only twelve real profiles are available for each class (six loads per class over two years). Training a model based on complex neural networks on such a small dataset would result in overfitting. Moreover, simpler models are sufficient to capture the main behaviors of these loads and they should be preferred given their higher interpretability. Thus, the year-long profiles are modeled using the PCA-based approach presented in Chapter 4.

Two independent models are learned for the two types of loads (mainly residential and mainly residential). Each dataset has size of 12×52 (12 profiles, each with 52 samples). Singular value decomposition is performed on each set of loads and, given the limited size, all 12 features are used. The distribution of the 12 columns of the coefficient matrix U is modeled as a Gaussian distribution. Figure 7.3 shows a comparison between real and generated profiles for mainly residential loads. Figure 7.4

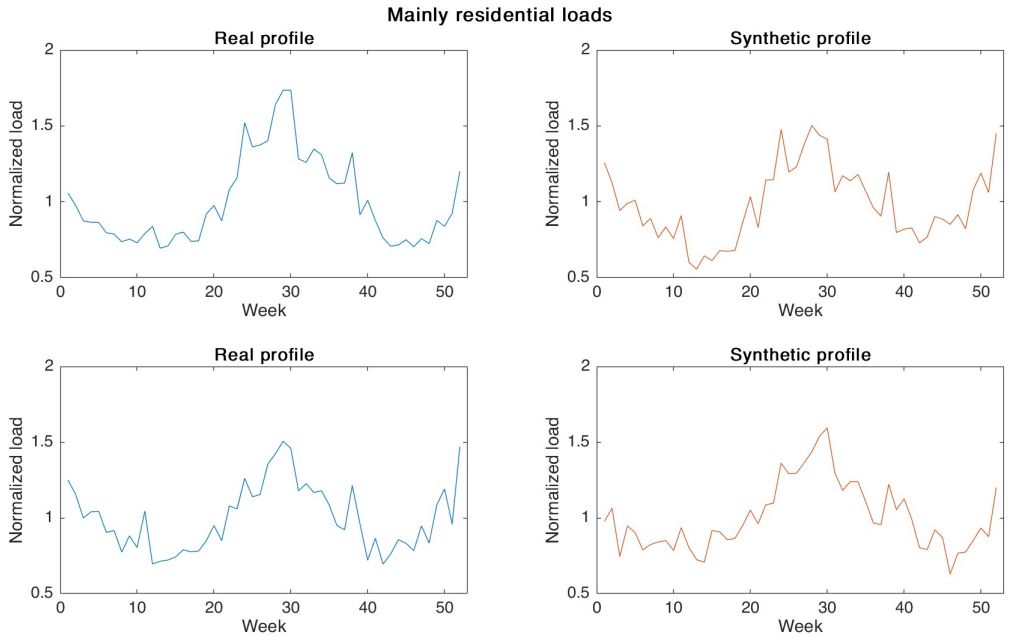

Figure 7.3: Comparison between two real mainly residential year-long profiles (blue, left) and two generated year-long profiles (red, right).

shows the same comparison but for mainly industrial loads.

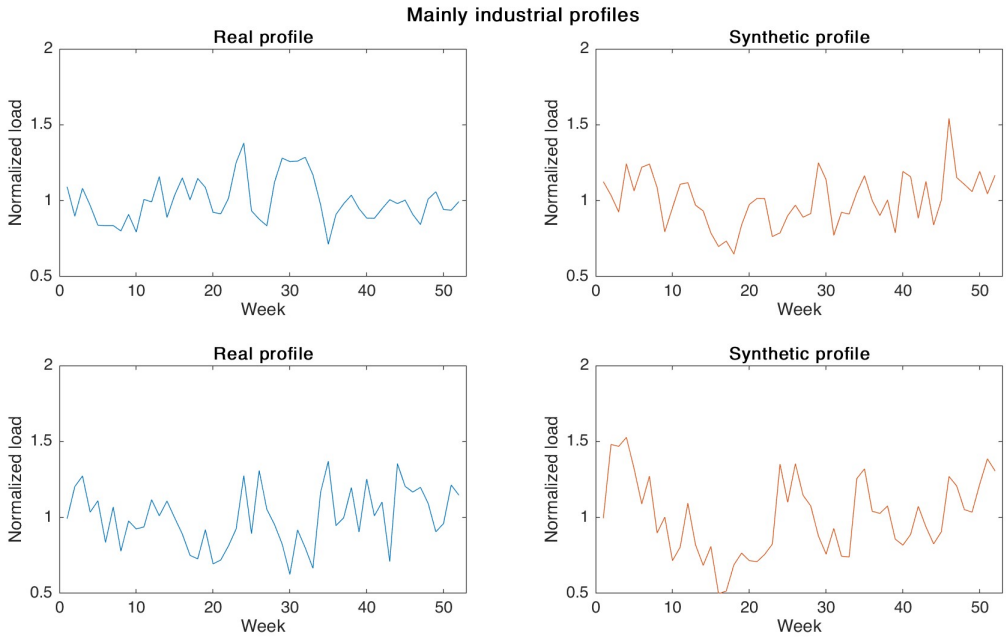

Figure 7.4: Comparison between two real mainly industrial year-long profiles (blue, left) and two generated year-long profiles (red, right).

7.3.2 Week-long profiles

As described in the previous chapter, the profiles at this level are one week long and sampled at 1 sample/hour. Each sample represents the average load over the corresponding one hour period. Each week-long profile is further normalized by dividing it by the average weekly load; this means that each week-long profile is normalized based on the value of the corresponding point in the year-long level.

To model the load profiles in this level, we use the cGAN described in Chapter 6 where the generation is conditioned on the load type (mainly residential or mainly industrial) and the season.

7.3.3 Hour-long profiles

The profiles at this level are sampled at 1 sample/30 seconds for a length of one hour; this results in each sequence being 120 samples long. Each point is computed as the average load over the 30 second period. The normalization of each profile is performed by dividing by the average load over the one hour period and by de-trending the resulting sequence. The de-trending step is necessary because, at this resolution and time length, the load data presents two main patterns: slow, large-amplitude trends and smaller, faster load variations. The slower trends represent the overall change in load consumption from one hour to the other; these are fully captured by the hourly data from the upper level and for this reason they do not need to be modeled at this level. By de-trending the signals, we extract the faster time-scale load changes which happen at a sub-minute scale.

Figure 7.5 show the steps followed to down-sample and normalize the raw load data in order to obtain hour-long profiles sampled at 1 sample/30 seconds. Figure 7.5-a) shows one week worth of raw load data sampled at 30 samples/second. The first

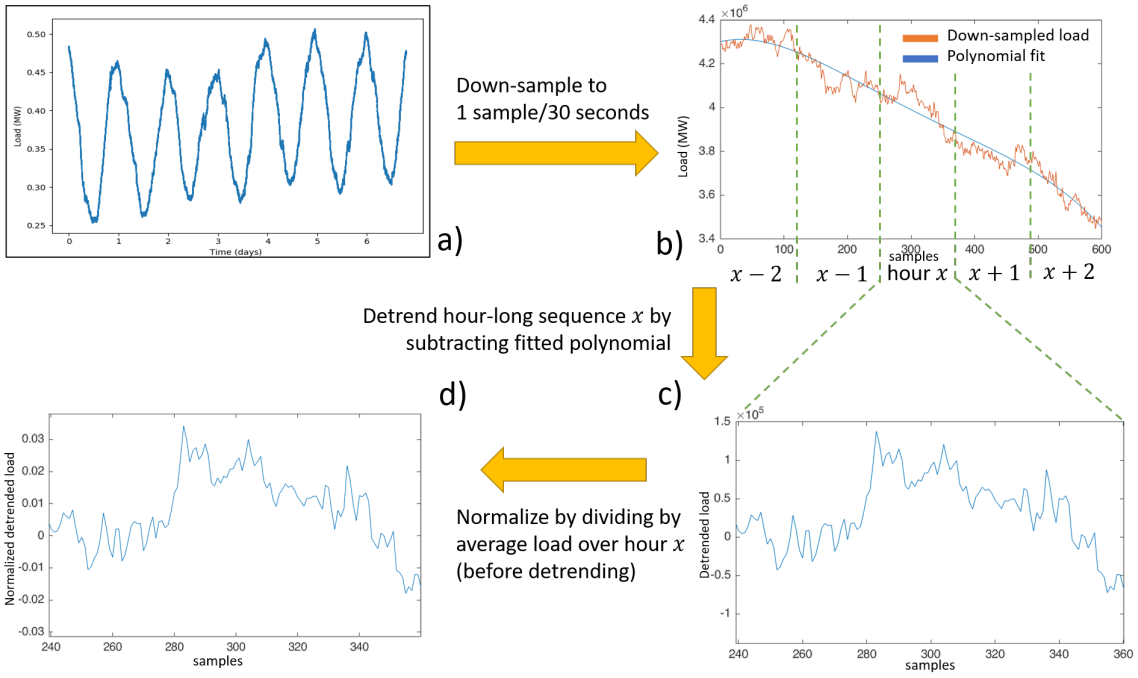

Figure 7.5: Disaggregation process for the hour-long profiles.

step consists in down-sampling this data by taking the average load value over 30 second periods to obtain the required resolution for this level. After this, each hour worth of data is further processed independently. For example, given hour x, the final hour-long sequence is obtained by first de-trending the data and then normalizing it. The de-trending step is illustrated in Figure 7.5-b. The plot shows the down-sampled load data for five consecutive hours: two hours before hour x and two hours after hour x. Based on this time interval, a 4^{th} degree polynomial is fitted to the data; this represents the overall trend of the load. Next, the load data for hour x is de-trended by subtracting from it the fitted polynomial; the resulting profile is shown in Figure 7.5-c. The last step consists in normalizing the hour-long profile by dividing it by the average load over hour x (calculated before the de-trending step).

Unlike at the week-long level, the loads at this resolution do not present marked differences in behavior due to the type of load or time of the year. For this reason, we use a regular GAN instead of the conditional GAN from the previous chapter. Fig-

ure 7.6 shows the architecture of the GAN used to model and generate the hour-long profiles. It can be noted that both the generator and discriminator have a simpler structure compared to the cGAN for the week-long profiles. Here, only one convolutional layer is used since the profiles do not have as strong temporal correlation. The size and characteristics of the input noise are the same as those used for the cGAN.

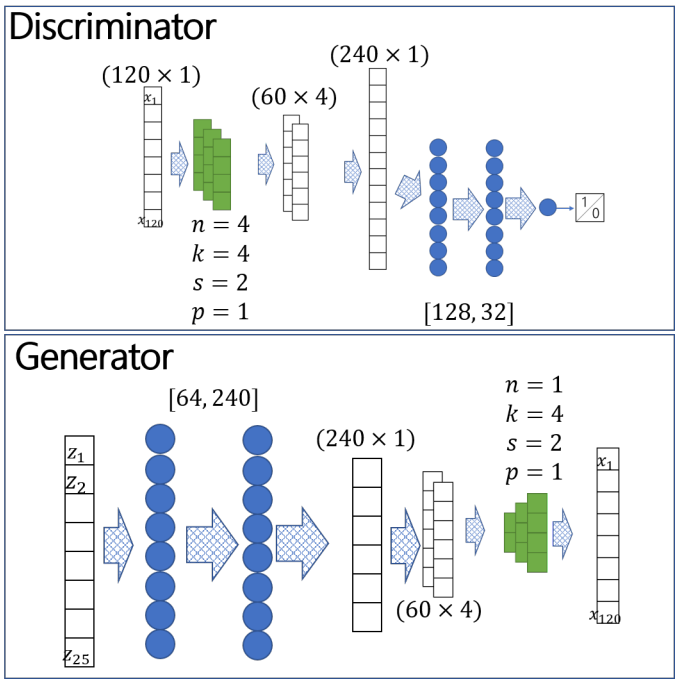

Figure 7.6: Hour GAN.

The experience gained during the design of the cGAN from Chapter 6 allowed for a faster tuning of the GAN for the hour-long profiles. In particular, most of the hyperparameters (such as those related to the loss function, learning rate and decay rate) were directly transferred from the cGAN and only required minimum adjusting. The same training strategy is also used here: the discriminator is updated twice as often as the generator in order to achieve a more robust training convergence. Figure 7.7 shows the convergence of the training process of the final GAN. Similarly to what was observed for the cGAN, the training starts with the discriminator easily distinguishing between real and generate profiles; after a few epochs, the generator improves to the

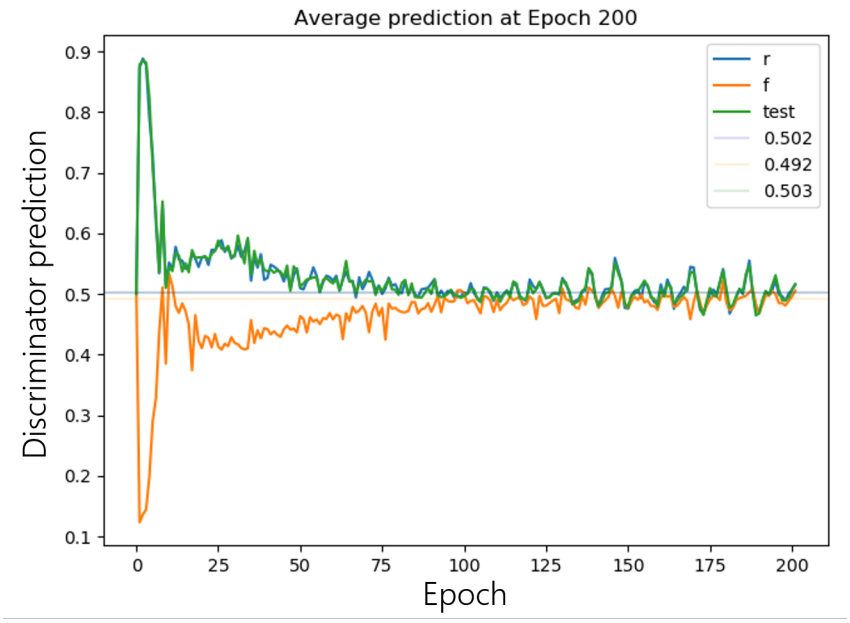

Figure 7.7: Training of the hour GAN.

point where the discriminator cannot tell the data apart. It is to be noted that in this particular case, the total number of epochs required for convergence is smaller than in the case of the cGAN, which converged after about 3000 epochs. This is due to the fact that the dataset of real hour-long profiles is much larger than in the case of week-long profiles: the former has almost 190,000 samples, while, as mentioned in the previous chapter, the latter has 1158 (which is a factor of approximately 168). Thus, given similar batch sizes, the hour-long GAN is updated many more times in an epoch than the cGAN.

Finally, the synthetic hour-long data resulting from the trained GAN was successfully validated following the same procedure as described for the week-long profiles in the previous chapter. For a visual comparison, figure 7.8 shows randomly selected examples of real and synthetic hour-long profiles.

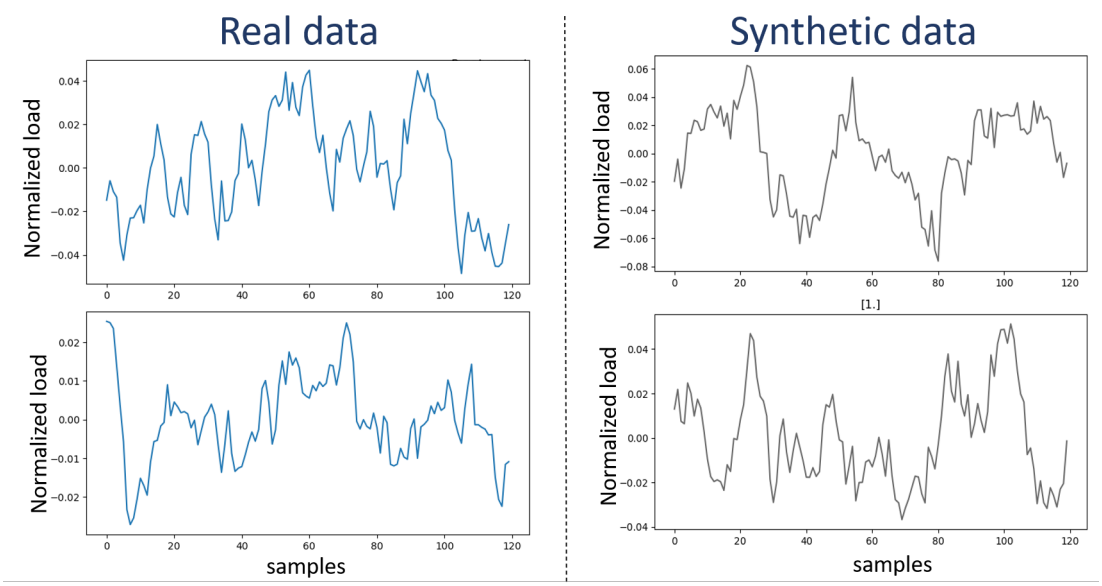

Figure 7.8: Comparison between some real hour-long profiles and some synthetic hour-long profiles.

7.3.4 30-second-long profiles

At the lowest level, the load profiles are 30 seconds long and sampled at the PMU rate of 30 samples/second. As this data represents such a short length of time, no overall trend is observed unlike the hour-long profiles at the level above. For this reason, each profile is simply normalized by dividing it by its average and no de-trending is performed.

To model these profiles, a similar GAN architecture to the one described in the previous section is adopted. The main difference lies in the overall size of the generator and discriminator models. Because each 30-second-long profile is made of 900 points, the GAN for this level requires a higher complexity compared to the one used for the hour-long profiles which only had 120 points each. For this reason, the generator now has three fully connected hidden layers and two transpose-convolutional layers. The final architecture of the GAN is shown in Fig. 7.9. Moreover, the hyper-parameters and training process are kept the same as the GAN for the hour-long profiles. Finally,

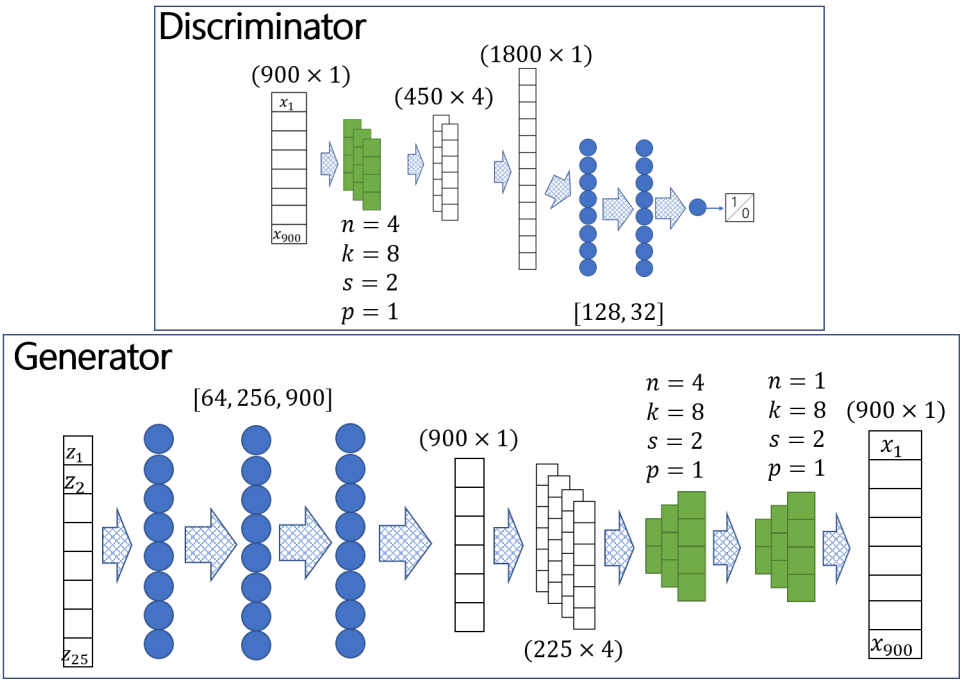

Figure 7.9: 30-second GAN.

Fig. 7.10 shows a comparison between some real and synthetically generated 30-second-long profiles.

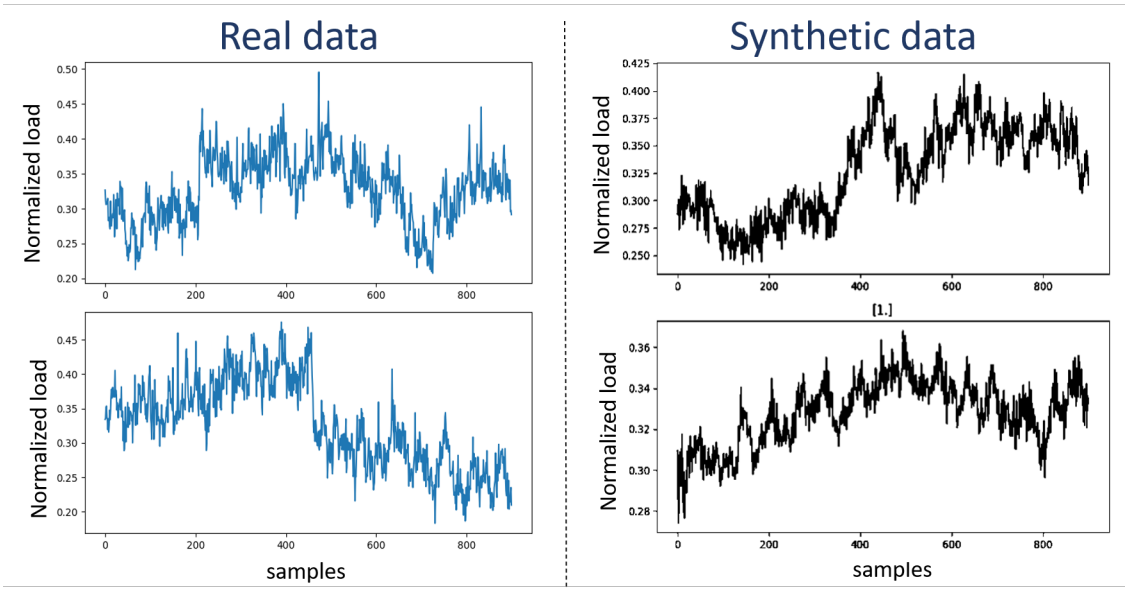

Figure 7.10: Comparison between some real 30-second-long profiles and some synthetic 30-second-long profiles.

7.4 Multi-resolution synthetic data

Having trained generative models for each load aggregation level, it is now possible to generate synthetic data for any time-length and any time-resolution by combining multiple profiles together. In particular, to obtain longer times, several profiles at a given level can be concatenated together. Further, to obtain a specific resolution, profiles at different levels can be combined together. In Sections 7.4.1 and 7.4.2 these two types of aggregation are described and some examples are shown. Finally, Section 7.4.3 presents a method to obtain any specific user-defined time-resolution without being limited to the predefined resolutions of the four original disaggregation levels.

7.4.1 Concatenating load profiles

At each level, a trained generative model creates profiles of fixed length, based on the length of the real profiles used for training. To obtain longer time durations, multiple profiles from a given level need to be concatenated together. For example, if four weeks of data are required, the cGAN for Level 2 is used to generate four individual week-long profiles which are then concatenated together to obtain the final time-length. Depending on the specific scenario, attention needs to be paid to ensure that the correct load characteristics are maintained; for example, the load type and season labels must be consistent when generating the four week-long profiles.

While this approach is very straightforward and it naturally follows from the load disaggregation structure we chose initially, it presents one challenge: ensuring continuity at the seam between two concatenated profiles. Because consecutive profiles are generated independently, concatenating them could result in some degree of mismatch between the end of one profile and the start of the next one.

For the profiles at Level 2 (week-long profiles), a linear filter is used to smoothen the seam between two profiles. The linear filter is applied to the last two points of one profile and the first two points of the following one. Moreover, the filter is chosen to have a size of five; that means that each point is adjusted as a linear combination of itself and the previous two and following two values, that is

$$x_k^{\text{new}} = \sum_{i=-2}^{2} \beta_i x_{k+i} \tag{7.1}$$

where, x_k is the sample to which the filter is applied, x_k^{new} is the new, filtered value and β is the vector of filter weights. The specific values of the filter are learned from the real load dataset by solving the following least-square problem:

$$X_{[-2,-1,1,2]}\ \beta_{[-2,-1,1,2]} = 0.5 X_k \tag{7.2}$$

where, $X_{[-2,-1,1,2]}$ is a matrix where each row represents the two previous values and two following values of an entry x_k and $\beta_{[-2,-1,1,2]}$ are the corresponding filter values. X_k is the column vector of all values x_k and using the factor 0.5 corresponds to fixing the value of $\beta_0 = 0.5$.

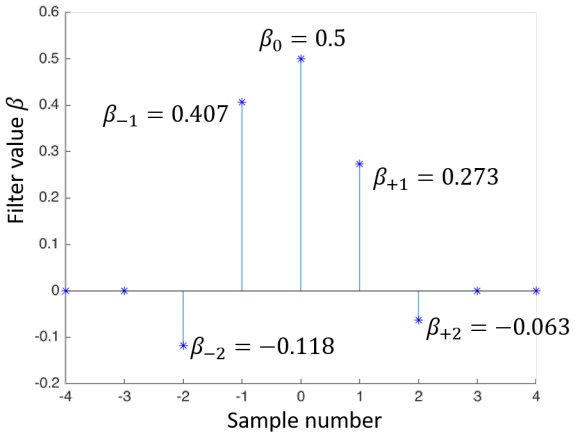

Figure 7.11: Values of the linear filter applied to the concatenation of week-long profiles.

Figure 7.11 shows the filter values learned for the week-long profiles, as a function of the sample number. Figure 7.12 shows two examples of the discontinuity that can

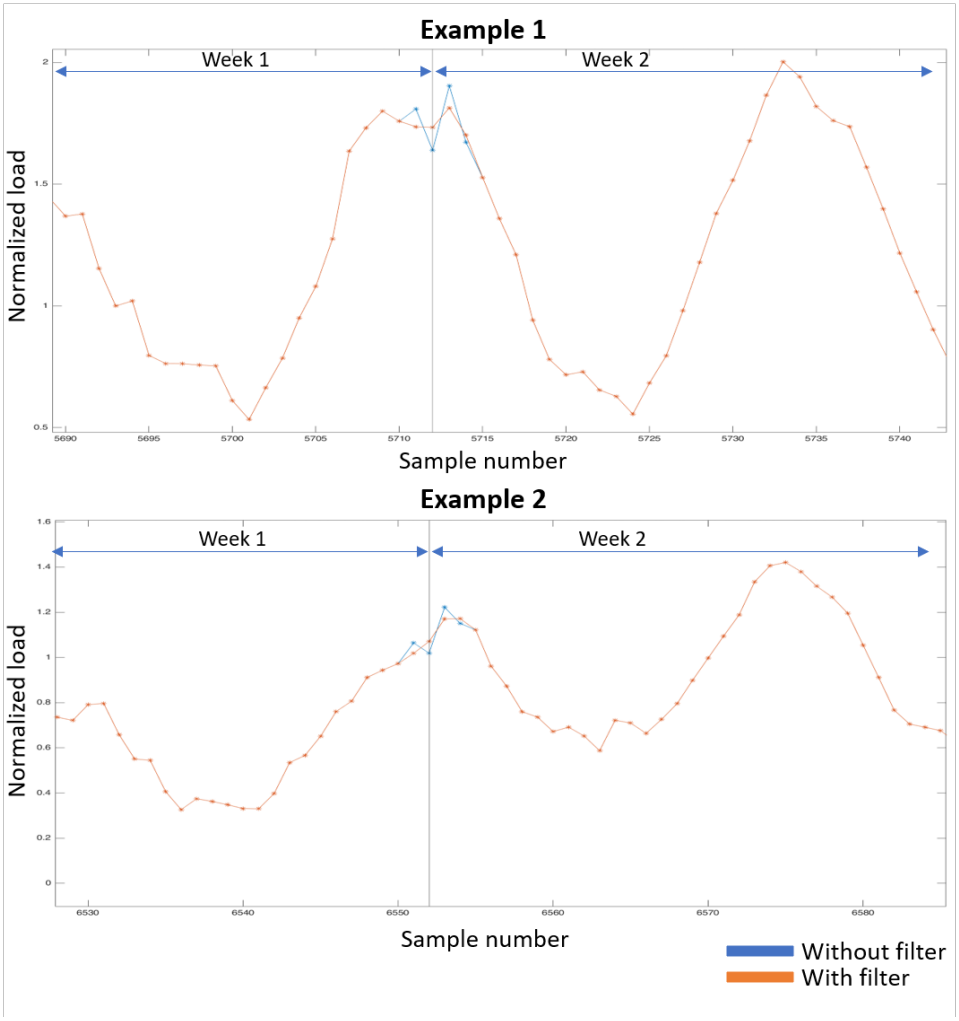

Figure 7.12: Worst-case examples of discontinuity between two concatenated summer profiles. The blue curve shows the load values before the filter is applied and the orange curve shows the filtered values which eliminate the discontinuity. The vertical line indicates the time coordinate of the last sample of the previous profile.

be observed when concatenating two profiles from Level 2. In each plot, the blue points represent the load values before applying the filter; the red curve shows the values after the linear filter is used. It can be seen that without the filter there is a sharp discontinuity at the seam between the two weeks; applying the filter results in a more regular load behavior. It is to be noted that the two examples shown represent worst case scenarios, where the discontinuity is relatively large; in most cases, the concatenation does not result in a detectable discontinuity.

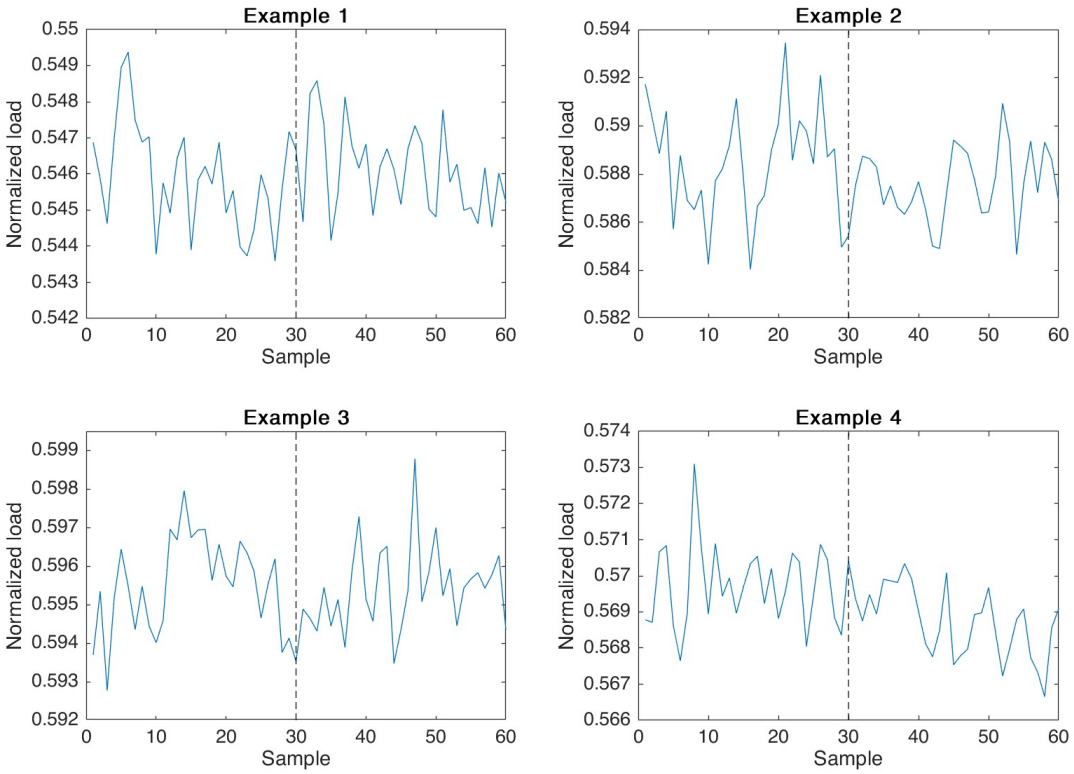

Figure 7.13: Examples of concatenation of 30-second-long profiles from Level 4. Each plot shows two seconds of data at 30 sample/second obtained by concatenating two separate profiles. The first 30 samples represent the end of one profile and the second 30 samples are the beginning of the following profile. The vertical line indicates the last sample of the previous profile. In all cases, it can be seen that the concatenation does not produce any recognizable discontinuity.

At Levels 3 and 4, it was observed that concatenating profiles does not yield the same discontinuity issues as observed for Level 2. This behavior is to be expected because at these time scales the load fluctuations are much smaller compared to the hourly changes in the week-long profiles and the behavior of the loads is less regular and predictable. For these reasons, the load change between two consecutive profiles at Level 3 or 4 is indistinguishable from the normal load behavior. To better illustrate this, Fig. 7.13 shows four separate examples of concatenation of profiles from Level 4. Each plot shows two seconds of data at 30 sample/second obtained by concatenating two separate profiles. The first 30 samples represent the end of one profile and the

Table 7.1: Comparison of the percentage difference between consecutive samples in real and generated data, for Level 3 and Level 4.

Aggregation Level	**Dataset**	**Percentage Difference**	
		Mean [%]	Std. Dev. [%]
Level 3	Real	0.59	0.49
	Synthetic	0.47	0.33
Level 4	Real	0.17	0.14
	Synthetic	0.13	0.27

second 30 samples are the beginning of the following profile. In all cases, it can be seen that the concatenation does not produce any recognizable discontinuity. While visual inspection of several generated profiles provided a qualitative confirmation of this phenomena, a statistical analysis is performed to further verify this observation. In particular, the percentage difference between the last sample of a profile and the first sample of the following profile is measured for a dataset of real data and a dataset of synthetically generated data. Table 7.1 shows a comparison of the mean and standard deviation of the percentage difference observed in the real and synthetic datasets, for both profiles at Level 3 and Level 4. The results of this statistical analysis confirm that a linear filter is not required to concatenate profiles from the two bottom levels since no significant difference can be observed between the seams of real profiles and generated profiles.

7.4.2 Combining profiles at different resolutions

In the previous section, we showed how synthetic load data of any length can be obtained by concatenating multiple profiles. In order to capture behaviors at different time-scales and to generate data at a specific resolution it is necessary to combine

profiles from different levels. For example, if we are interested in hourly load data over an entire year, first 52 week-long profiles must be concatenated (following the correct seasons) to get the desired resolution (1 sample/hour) and secondly, they must be combined with one year-long profile from the first level to capture the long-term seasonal behavior of the load. The process of combining different levels follows directly from the disaggregation scheme which was adopted initially to down-sample the load profiles at different resolutions. The steps for the three possible combinations are as follows.

(i) *Combining Levels 1 and 2.* The profiles from Level 1 (year-long profiles at 1 sample/week) capture the slow and long term changes in energy demand across months and years, while Level 2 (week-long profiles at 1 sample/hour) captures the daily load behavior. The real dataset of week-long profiles was normalized by dividing each profile by their average weekly load; at the same time, each point in the year-long profiles was computed as the average demand during any given week. This means that to combine the two levels together, each week-long profile must be scaled by the corresponding value in the year-long profile. The scaling is performed by multiplying the week-long profile at week i, defined as X_2^i, by the factor $x_{1,i}/\text{mean}(X_2^i)$, where $x_{1,i}$ is the value corresponding to the i^{th} week from the year-long profile X_1. By doing so, the week-long profile is effectively normalized so that its average equals the corresponding value of the year-long profile.

(ii) *Combining Levels 2 and 3.* For resolutions up to 1 sample every 30 seconds, the profiles from Level 2 need to be combined with profiles from Level 3 (hour-long profiles at 1 sample/30 seconds). The process of combining the hour-long profiles with a week-long profile consists of two steps: 1) scaling the hour-long profiles and 2) adding the polynomial trend. As we showed in Section 7.3.3, the last step in creating the hour-long profiles consisted in normalizing them by dividing by the average hourly

load. For this reason, when generating the synthetic data, we first scale each hour-long profile by the corresponding value in the week-long profile following the same procedure as with Levels 1 and 2 above. Secondly, a polynomial trend is added to each hour-long profile in order to capture the slower load variations that can be observed from the week-long profile. The trend is computed by fitting a 4^{th} degree polynomial to the points in the week-long profile, from two hours before the hour of interest to two hours after.

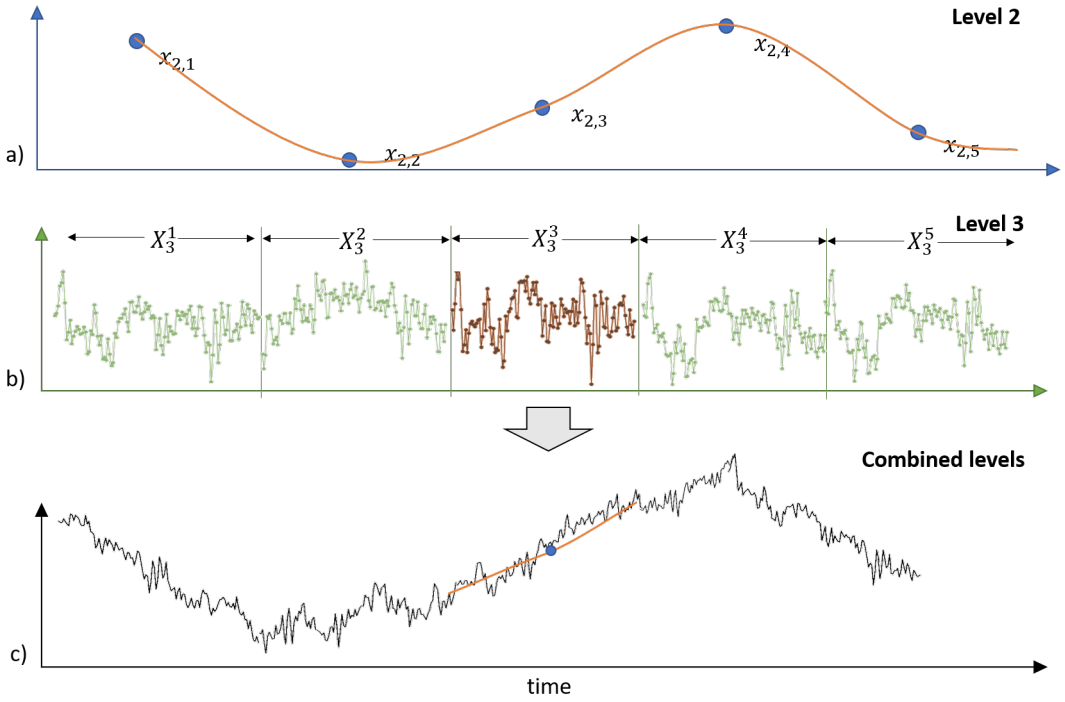

Figure 7.14: Illustration of the process to combine profiles from Level 2 and 3. Plot a) shows 5 hours of data at Level 2, where each point represents the average hourly load. The orange curve represents the fitted 4-degree polynomial for hour number 3 which is obtained by fitting the data representing 2 hours before and two hours after the hour of interest (hour 3 in this case). Plot b) shows 5 hour-long profiles which have been concatenated to obtain 5 hours worth of data a 1 sample every 30 seconds. Plot c) shows the result of combining the profiles together.

This particular process is illustrated in Fig. 7.14: in this example, 5 hours of data are generated by combining 5 hours from Level 2 and 5 hour-long profiles from Level 3 (please note that these plots are for illustration purposes and they do not

show real examples of generated data). Since each point in plot a) represents the average hourly load, each of the 5 hour-long profiles are first scaled based on the corresponding average values and they are then concatenated to obtain five hours of consecutive data. To capture the overall behavior observed in the Level 2 profile, the polynomial trend must be added. For example, for the 3^{rd} hour-long profile, a polynomial curve (shown in orange in plot a)) is fitted to the five points representing two hours before and two hours after hour 3. This trend is then added to the hour-long profile and the resulting time-series is shown in plot c). By repeating this process for each hour, the overall envelope of the load is modeled. At this point, it is also possible to demonstrate the validity of the polynomial trend approach by comparing it to a simpler, linear approach. Figure 7.15 shows six hours of data obtained by combining one profile from Level 2 and six profiles from Level 3. The blue curve represents the data combined using the polynomial approach, while in orange is the result of combining the profiles using a linear trend computed between two consecutive hours. It can be seen that using a 4^{th} degree polynomial yields a very smooth and natural behavior; on the other hand, the linear approach results in very sharp load changes where different hour-long profiles are concatenated together.

(iii) *Combining Levels 3 and 4.* The process of combining these two levels is the same as in case (i). The 30-second-long profiles of Level 4 were originally obtained by normalizing the real data by dividing them by their average value; at the same time, each point in the Level 3 profiles represents the average load over a 30 seconds period. Thus, the generated 30-second-long profiles are scaled by the corresponding value in the hour-long profiles to which they are combined with.

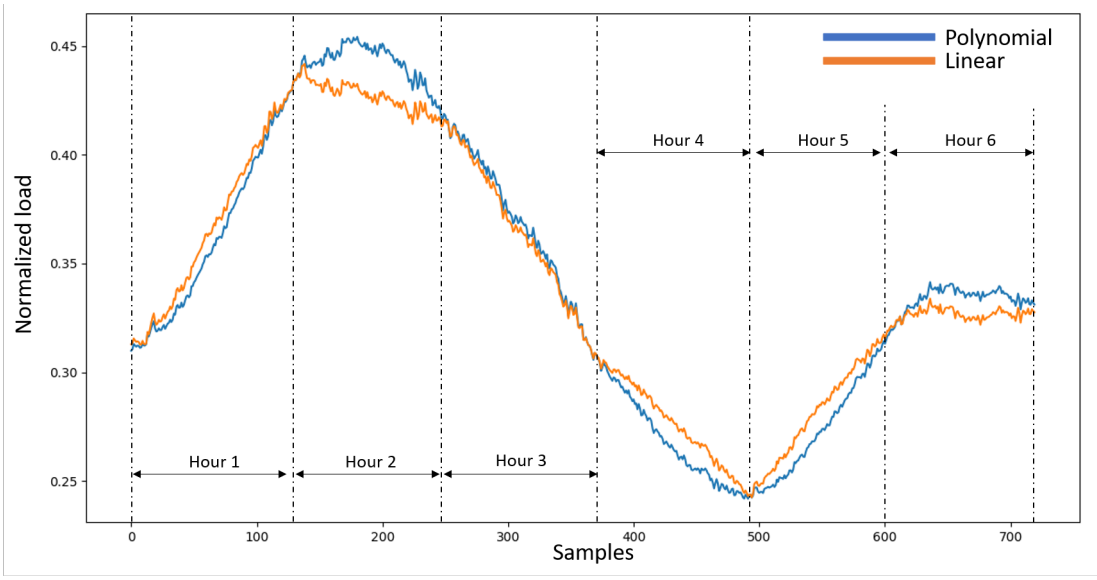

Figure 7.15: Combining Levels 2 and 3: comparison between polynomial and linear approach.

7.4.3 Generating load profiles at arbitrary resolutions

To make the proposed multi-resolution generative scheme applicable to the largest number of use cases and applications, a user should not be limited to the four predetermined sampling resolutions which were used to disaggregate the data. To obtain a specific resolution, first, a synthetic load profile is generated by aggregating profiles from different levels as described in Section 7.4.2 in such a way that the resulting time-series will have a resolution which is higher than that requested; then, the generated data is down-sampled to the exact resolution necessary. For example, a common sampling rate for load measurements is 1 sample/10 minutes as specified by the standard IEC 61000-4-30 [56]. As this resolution falls between that of the week-long profiles (1 sample/hour) and hour-long profiles (1 sample/30 seconds), the synthetic data will be generated by aggregating profiles down to Level 3 and the resulting output is then down-sampled to achieve the correct resolution. The down-sampling can be achieved by computing the average load over the desired sampling period as well as

other statistics such as minimum or maximum value. The choice of down-sampling metric depends on the specific resolution needed as well as the user requirements. For example, if daily load values are needed, the synthetic data will be generated at the week-long level at 1 sample/hour and then the values over a 24 hour period will be aggregated. Depending on the analysis for which the data is needed, the maximum daily value could be considered in order to provide a worst case scenario or the mean or minimum values could be computed to represent lighter load conditions.

7.4.4 Validation of synthetic data

In Section 7.3, the generative models for each level have been validated by performing several tests on the synthetic data generated by them. These tests showed that the synthetic load profiles retain the characteristics of the real data and they can successfully be used for power system applications. The last step in validating the compete multi-resolution generative scheme consists in verifying that it can correctly create realistic load data at any resolution. By testing synthetic data at a sampling rate different from any of the four that were initially chosen for the four aggregation levels, we can effectively validate the techniques used for the aggregation of profiles from different levels as well as the re-sampling process to obtain a specific resolution.

The validation procedure consists in training a load forecasting algorithm solely on synthetically generated data and then testing it on the real data, similarly to what described in Section 6.6.3 for the week-long profiles. In this case, we test time-series load data sampled at 1 sample/10 minutes: to generate this data, profiles from Level 2 and 3 need to be combined together and then down-sampled to the chosen resolution. An LSTM network with four layers and 36 units per layer is trained to predict the load value at a given point in time, based on the previous six hours of data (that is, the previous 36 samples). The algorithm is trained on a dataset of about 8000

Table 7.2: Comparison of the forecasting error between generated and real load data, sampled at 1 sample/10 minutes.

Testing Dataset	**Percentage Error**	
	Mean	Std. Dev.
Synthetic	2.15	2.35
Real	2.08	1.58

six-hours-long profiles synthetically generated using the LoadGAN application. The trained model is then tested on 1000 new synthetic profiles as well as 1000 profiles from real load data; to evaluate the performance, the absolute percentage difference between the actual and predicted value is computed. From the results shown in Table 7.2, it can be noted that the characteristics of the forecasting error when tested on the two datasets match very closely. This means, that although the model was only trained on synthetic data, it effectively captures the behavior of the real data.

7.5 LoadGAN - open source application

All the code developed as part of this project is freely available online at [55]. This GitHub repository contains the code used to create and train the generative models for each of the levels as well as the fully trained models which can be directly used to generate new synthetic load data. Moreover, we created an application for the generation of multi-resolution time-series data, called LoadGAN, which can be used via a graphical user interface (GUI).

LoadGAN allows a user to generate unique synthetic data tailored to a specific application by leveraging all of the features of the generative scheme described in this chapter. Figure 7.16 shows the settings which are available for the targeted generation of synthetic load data. Multiple profiles can be generated simultaneously by specifying the number of mainly residential and mainly industrial loads in a system.

The sampling resolution of the synthetic data is entered as number of samples over sampling period; as an example, Fig. 7.16 shows a resolution of 1 sample/10 minutes, as specified by IEC 61000-4-30. As described in the previous section, the aggregation to a specific resolution can be achieved by computing the mean, minimum or maximum value over the sampling period; the user can select the desired aggregation type. The temporal length of the generated data can be specified from a few seconds to one entire year. Finally, the user can select the season to be used as label for the generation of the week-long profiles. When creating a full year of data, this setting will be overridden and the data will be generated based on the correct sequence of seasons, starting in winter (January 1st). The synthetic data generated is saved in .csv format and the destination folder and file name can be modified by the user. The application also computes the estimated file size based on the currently selected settings before the data is generated.

7.6 Use-cases for synthetic load data

Load data is crucial in a variety of power system studies, both in traditional applications as well as for the development of data-driven algorithms. In both cases, load data either represents the main variable of interest (for example in load forecasting or scenario generation studies) or it is required for the creation of realistic simulations.

Examples of traditional power system studies that rely on time-series load data are listed below along with an explanation of how LoadGAN can be used to generate the necessary data.

- Transmission expansion planning: the problem of TEP consists in identifying the optimal strategy to improve the transmission network in order to meet some predetermined future goals [57]. Generally, the objective is to predict and adjust to the changes in load demand and generation portfolio with a multi-

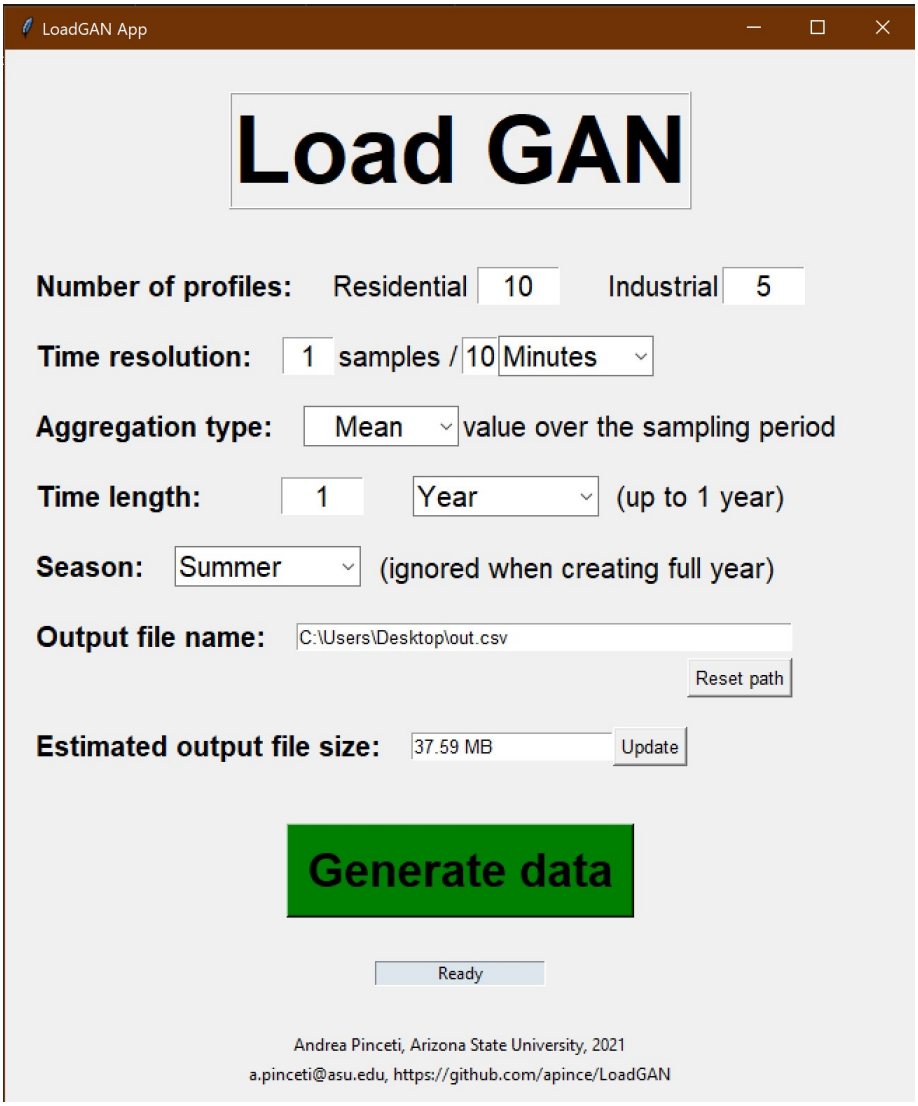

Figure 7.16: Interface of the LoadGAN application.

year horizon [58, 59]. A key component of such studies is the ability to generate meaningful scenarios which can capture the variability and uncertainty of the future energy demand [60, 61, 62]. LoadGAN can be used to create large dataset of synthetic load data for multiple years and under different conditions. These datasets, which capture individual bus-level load behaviors, could represent the basis for the development of limited, representative scenarios. Moreover, the proposed framework can be easily improved by training it on additional

data under specific conditions: for example, the models can be conditionally trained to learn the behavior of loads with increasing penetration of consumer photovoltaic generation or electric vehicle charging.

- Market related studies: the availability of rich, bus-level load data is crucial for research in the field of electricity markets. For example, studies on the design of effective demand response mechanisms require realistic time-series load data to model the system behavior during many days or weeks of operation and identify opportunities for load shifting or load curtailment [63, 64].

- Stability and dynamics studies: in order to accurately study power system stability and assess its dynamic behavior, time-series load data should be used. Varying the loads during this type of simulations allows for a more realistic setup, where the behavior of the system reflects the continuous changes due to the generators responding to load changes [65, 66, 67].

In terms of data-driven and machine learning applications that leverage large datasets of load data, some examples are:

- Load forecasting: one of the most established applications of machine learning to power systems is load forecasting. Among the extensive literature covering this topic, several different methods are used to predict the future system loads: neural networks [68, 69, 70, 71], support vector models [72, 73, 74, 75], long-short term memory models [76, 77, 78], as well as combinations of them [79, 80]. Depending on the use cases, load forecasting is performed at different time horizons: from short-term (few minutes to hours ahead) to long-term (day ahead to weeks or months). Also, it can be done at aggregate levels (system net load) or at the individual bus level. LoadGAN provides rich datasets which span all the time-horizons of interest for load forecasting problems. The synthetically

generated data can be used to either train or test forecasting algorithms as well as a way to enrich limited datasets of real load data. (at different time horizons)

- Large-scale simulations for machine learning applications: as the number of use cases for data driven-based algorithms increases, the availability of large amounts of power system data becomes crucial. Many applications are developed and tested based on synthetic data generated via dynamic simulations. For example, extensive measurement datasets based on simulated contingencies and faults can be used to train event detection and classification algorithms [81, 82, 83]. As described in the beginning chapters, cyber-security is a field which heavily relies on data driven techniques to identify vulnerabilities and propose countermeasures [84, 85, 86]. Data is also often used to develop PMU-based applications for monitoring and controlling the electrical grid [87, 88, 89]. In all these scenarios, realistic time-series load data can be used to simulate and better capture the dynamical behavior of power systems and provide different system conditions.

7.7 Ongoing work

The generative scheme proposed in this chapter is currently being used to generate synthetic load data for the following independent research projects.

7.7.1 Generation of synthetic eventful PMU data under time-varying load condition

As part of a collaborative effort with the research group of Dr. Le Xie from Texas A&M University, we are developing a machine-learning based approach for the generation of eventful synthetic PMU data. To capture the dynamical behavior of a power

system, we propose to use ordinary differential equation (Neural ODE), a data-driven approach that uses neural networks to learn the parametrization of the state equations during events such as oscillations or faults. Because the system dynamics are highly dependent on the initial conditions (that is, the steady state operating point), in order to create diverse and realistic eventful PMU data, time-series load data is necessary to model different operating states. Thus, in this framework, LoadGAN is being used to generate large datasets of synthetic time-varying load data at different resolutions; individual profiles are created and mapped to the loads in the test systems. Each time-stamp within these datasets represents a unique operating condition which allows for the training and modeling of the Neural ODE under different scenarios. As a result of combining these fast-scale models of the dynamical behavior of a system with time-varying load conditions, realistic synthetic PMU data can be generated for relatively long time horizons (minutes or hours) while capturing the effects of a large number of system events.

7.7.2 Synthetic dataset for real-time event detection and classification

The second project for which LoadGAN is currently being used deals with the detection and classification of power system events based on real-time PMU data. This work aims at designing machine learning algorithms to extract meaningful features from a set of measurements and subsequently using these as signatures for the classification of different types of events. These models have initially been tested on the dataset of real PMU data described in Chapter 5, which contains information about 70 recorded network events. To improve the algorithm and to better characterize its performance, a larger and richer dataset of events is needed. As described in Section 7.6, an accepted approach for the study of data-driven algorithms in power system is to generate synthetic datasets via simulation tools. In this case, using well-

established synthetic networks (such as the synthetic Texas model [24]) and PSS/E simulation software, hundreds of system contingencies and outages can be simulated and realistic dynamic data can be generated to capture the system response. As explained in the previous section, the dynamics of a power system are highly dependent on the operating conditions. Thus, LoadGAN is being used to create a large dataset of different load conditions by generating individual time-series profiles for each load in the system. This approach allows for the training and testing of the classification algorithms under varying system states, making them more robust and effective in real-world use.

7.8 Conclusion

In Chapters 4 and 6 two methods for the generation of realistic time-series load data are introduced. In particular, in Chapter 4 we demonstrated a learning algorithm based on principal component analysis which can be used to extract typical temporal patterns from a real load dataset; these patterns can be linearly combined to generate new time-series data. The final generative model includes weighting factors which guarantee that the spatial characteristics that were observed in the real data are maintained in the synthetic data. Finally, we have verified that all the resulting cases represent valid power system operating conditions. One of the drawbacks of this generative model is the fact that the length of the synthetic data in terms of time is limited to the length of the real data that is available. In our case, the data is generated at hourly interval for one consecutive week. Also, the user does not have any control on the generation process, meaning that it cannot request specific types of profiles. These limitations are overcome by the second proposed approach, based on GANs and described in Chapter 6. This second method to generate synthetic transmission load data at a bus level leverages conditional generative adversarial

networks which allow for the selection of the time of the year and type of load for which to generate time-series load profiles. Extensive testing is performed and we have verified the validity of our method and quality of the generated data.

Building on these models, in this Chapter a complete scheme for the generation of time-series load data at any resolution is proposed. To capture load behaviors at varying time-scales, the generative framework includes multiple models (GANs, cGANs, and PCA-based models) which appropriately combined can result in load data of any time length and resolution. Each of these models is extensively tested and the aggregation scheme is carefully designed to produce high-quality, realistic time-series data. All of these components have been compiled into a simple open-source graphical interface that, along with the trained generative models, allows an end-user to quickly generate any amount of synthetic load data based on selectable parameters.

More broadly, the proposed generative scheme represents a flexible framework which can be tuned to capture an even larger number of interesting load characteristics. For example, as mentioned in the previous section, the generation of representative load scenarios is crucial to ensure the reliability of our fast-evolving electrical grid. The ability of the proposed generative scheme to learn and subsequently generate data conditioned on different load characteristics could be leveraged for the efficient modeling of loads with varying degrees of penetration of electrical vehicles and/or rooftop solar.

REFERENCES

[1] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *Ccs*, vol. 14, no. 1, pp. 1–33, 2009.

[2] Y. Yuan, Z. Li, and K. Ren, "Modeling Load Redistribution Attacks in Power Systems," *IEEE Transactions on Smart Grid*, 2011.

[3] J. Zhang and L. Sankar, "Physical system consequences of unobservable state-and-topology cyber-physical attacks," *IEEE Transactions on Smart Grid*, vol. 7, no. 4, pp. 2016–2025, 2016.

[4] A. Sanjab and W. Saad, "Data Injection Attacks on Smart Grids With Multiple Adversaries: A Game-Theoretic Perspective," *IEEE Transactions on Smart Grid*, vol. 7, pp. 2038–2049, jul 2016.

[5] L. Xie, Y. Mo, and B. Sinopoli, "Integrity Data Attacks in Power Market Operations," *IEEE Transactions on Smart Grid*, vol. 2, pp. 659–666, dec 2011.

[6] PJM, "PJM metered load data."

[7] L. Liu, M. Esmalifalak, Q. Ding, V. A. Emesih, and Z. Han, "Detecting false data injection attacks on power grid by sparse optimization," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 612–621, 2014.

[8] J. Zhao, G. Zhang, and R. A. Jabr, "Robust Detection of Cyber Attacks on State Estimators Using Phasor Measurements," *IEEE Transactions on Power Systems*, vol. 32, no. 3, pp. 2468–2470, 2017.

[9] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. V. Poor, "Machine Learning Methods for Attack Detection in the Smart Grid," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 8, pp. 1773–1786, 2016.

[10] J. Zhang and L. Sankar, "Physical System Consequences of Unobservable State-and-Topology Cyber-Physical Attacks," *IEEE Transactions on Smart Grid*, vol. 7, no. 4, pp. 2016–2025, 2016.

[11] Z. Chu, J. Zhang, O. Kosut, and L. Sankar, "Evaluating power system vulnerability to false data injection attacks via scalable optimization," in *2016 IEEE International Conference on Smart Grid Communications, SmartGridComm 2016*, pp. 260–265, 2016.

[12] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. September, pp. 1–58, 2009.

[14] G. L. Prajapati and A. Patle, "On performing classification using SVM with radial basis and polynomial kernel functions," in *Proceedings - 3rd International Conference on Emerging Trends in Engineering and Technology, ICETET 2010*, pp. 512–515, 2010.

[15] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[16] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics.*, pp. 431–441, September 1963.

[17] A. Pinceti, L. Sankar, and O. Kosut, "Load Redistribution Attack Detection using Machine Learning: A Data-Driven Approach," in *IEEE Power and Energy Society General Meeting*, 2018.

[18] V. Joshi, J. Solanki, and S. K. Solanki, "Statistical methods for detection and mitigation of the effect of different types of cyber-attacks and parameter inconsistencies in a real world distribution system," in *2017 North American Power Symposium, NAPS 2017*, 2017.

[19] J. J. Yu, Y. Hou, and V. O. Li, "Online False Data Injection Attack Detection with Wavelet Transform and Deep Neural Networks," *IEEE Transactions on Industrial Informatics*, 2018.

[20] Y. Huang, J. Tang, Y. Cheng, H. Li, K. A. Campbell, and Z. Han, "Real-time detection of false data injection in smart grid networks: An adaptive CUSUM method and analysis," *IEEE Systems Journal*, 2016.

[21] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, January 1967.

[22] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection : A survey," *ACM Computing Surveys*, vol. 41(3), July 2009.

[23] PJM, "PJM metered load data."

[24] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid Structural Characteristics as Validation Criteria for Synthetic Networks," *IEEE Transactions on Power Systems*, vol. 32, pp. 3258–3265, Jul 2017.

[25] H. Li, A. L. Bornsheuer, T. Xu, A. B. Birchfield, and T. J. Overbye, "Load modeling in synthetic electric grids," in *2018 IEEE Texas Power and Energy Conference, TPEC 2018*, 2018.

[26] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, pp. 12–19, Feb 2011.

[27] A. Pinceti, L. Sankar, and O. Kosut, "Data-Driven Generation of Synthetic Load Datasets Preserving Spatio-Temporal Features," in *IEEE Power and Energy Society General Meeting*, 2019.

[28] L. Sankar, O. Kosut, and K. Hedman, "A verifiable framework for cyber-physical attacks and countermeasures in a resilient electric power grid."

[29] R. Podmore, "Digital computer analysis of power system networks," *PhD Thesis, University of Canterbury, Christchurch, New Zealand*, 1972.

[30] IncSys, Inc., "IncSys - power system simulation software."

[31] R. Khodadadeh, "Designing a software platform for evaluating cyber-attacks on the electric power grid," *Master's Thesis*, 2019.

[32] A. Pinceti, "Nearest neighbor attack detection." `https://github.com/apince/EMS_FDI_NearestNeighborAttackDetection`, 2019.

[33] A. Abusorrah, A. Alabdulwahab, Z. Li, and M. Shahidehpour, "Minimax-Regret Robust Defensive Strategy Against False Data Injection Attacks," 2017.

[34] H. Shayan and T. Amraee, "Network Constrained Unit Commitment Under Cyber Attacks Driven Overloads," *IEEE Transactions on Smart Grid*, p. 1, 2019.

[35] T. M. Cover and J. A. Thomas, *Elements of information theory*, ch. 13. Wiley-Interscience, 1991.

[36] CAISO, "CAISO historical EMS hourly load."

[37] C. Grigg *et al.*, "The IEEE reliability test system-1996. A report prepared by the reliability test system task force of the application of probability methods subcommittee," *IEEE Transactions on Power Systems*, vol. 14, pp. 1010–1020, Aug 1999.

[38] A. Khotanzad, R. Afkhami-Rohani, T.-L. Lu, A. Abaye, M. Davis, and D. J. Maratukulam, "ANNSTLF - A neural-network-based electric load forecasting system," *IEEE Transactions on Neural Networks*, vol. 8, pp. 835–846, July 1997.

[39] I. Nadtoka, S. Vyalkova, and F. Makhmaddzonov, "Maximal electrical load modeling and forecasting for the Tajikistan power system based on principal component analysis," in *2017 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pp. 1–4, May 2017.

[40] L. Xiao-fei and S. Li-qun, "Power system load forecasting by improved principal component analysis and neural network," in *2016 IEEE International Conference on High Voltage Engineering and Application (ICHVE)*, pp. 1–4, Sept 2016.

[41] F. M. Bianchi, E. D. Santis, A. Rizzi, and A. Sadeghian, "Short-term electric load forecasting using echo state networks and PCA decomposition," *IEEE Access*, vol. 3, pp. 1931–1943, 2015.

[42] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[43] Z. Chu, A. Pinceti, R. S. Biswas, O. Kosut, A. Pal, and L. Sankar, "Can predictive filters detect gradually ramping false data injection attacks against pmus?," in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6, 2019.

[44] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pp. 330–339, 2010.

[45] M. P. Andersen and D. E. Culler, "Btrdb: Optimizing storage system design for timeseries processing," in *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pp. 39–52, 2016.

[46] "PingThings Predictive Grid." `https://www.pingthings.io`.

[47] K. Prasertwong, N. Mithulananthan, and D. Thakur, "Understanding low-frequency oscillation in power systems," *International Journal of Electrical Engineering Education*, vol. 47, no. 3, pp. 248–262, 2010.

[48] Y. Chen, *Early Anomaly Detection and Classification with Streaming Synchrophasor Data in Electric Energy Systems*. PhD thesis, 2015.

[49] R. K. Mai, L. Fu, Z. Y. Dong, B. Kirby, and Z. Q. Bo, "An adaptive dynamic phasor estimator considering dc offset for pmu applications," *IEEE Transactions on Power Delivery*, vol. 26, no. 3, pp. 1744–1754, 2011.

[50] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[51] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[52] X. Zheng, B. Wang, and L. Xie, "Synthetic dynamic PMU data generation: A generative adversarial network approach," in *2019 SGSMA Conference*, pp. 1–6, 2019.

[53] Y. Chen, P. Li, and B. Zhang, "Bayesian renewables scenario generation via deep generative networks," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, 2018.

[54] Z. Wang and T. Hong, "Generating realistic building electrical load profiles through the generative adversarial network (GAN)," *Energy and Buildings*, vol. 224, 2020.

[55] A. Pinceti, "Synthetic load GAN." `https://github.com/apince/LoadGAN`, 2021.

[56] IEC 61000-4-30:2015, "Electromagnetic compatibility (EMC) - Part 4-30: Testing and measurement techniques - Power quality measurement methods," standard, International Electrotechnical Commission, 2015.

[57] G. Latorre, R. D. Cruz, J. M. Areiza, and A. Villegas, "Classification of publications and models on transmission expansion planning," *IEEE Transactions on Power Systems*, vol. 18, no. 2, pp. 938–946, 2003.

[58] H. Zhang, V. Vittal, G. T. Heydt, and J. Quintero, "A mixed-integer linear programming approach for multi-stage security-constrained transmission expansion planning," *IEEE Transactions on Power Systems*, vol. 27, no. 2, pp. 1125–1133, 2012.

[59] R. A. Jabr, "Robust transmission network expansion planning with uncertain renewable generation and loads," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4558–4567, 2013.

[60] M. Sun, J. Cremer, and G. Strbac, "A novel data-driven scenario generation framework for transmission expansion planning with high renewable energy penetration," *Applied Energy*, vol. 228, pp. 546–555, 2018.

[61] M. Sun, F. Teng, I. Konstantelos, and G. Strbac, "An objective-based scenario selection method for transmission network expansion planning with multivariate stochasticity in load and renewable energy sources," *Energy*, vol. 145, pp. 871–885, 2018.

[62] Y. Liu, R. Sioshansi, and A. J. Conejo, "Hierarchical clustering to find representative operating periods for capacity-expansion modeling," *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3029–3039, 2018.

[63] M. Parvania, M. Fotuhi-Firuzabad, and M. Shahidehpour, "Optimal demand response aggregation in wholesale electricity markets," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 1957–1965, 2013.

[64] H. Aalami, M. P. Moghaddam, and G. Yousefi, "Demand response modeling considering interruptible/curtailable loads and capacity market programs," *Applied Energy*, vol. 87, no. 1, pp. 243–250, 2010.

[65] M. Koivisto, K. Das, F. Guo, P. Sørensen, E. Nuño, N. Cutululis, and P. Maule, "Using time series simulation tools for assessing the effects of variable renewable energy generation on power and energy systems," *WIREs Energy and Environment*, vol. 8, no. 3, p. e329, 2019.

[66] V. S. N. Arava and L. Vanfretti, "A method to estimate power system voltage stability margins using time-series from dynamic simulations with sequential load perturbations," *IEEE Access*, vol. 6, pp. 43622–43632, 2018.

[67] E. Vittal, M. O'Malley, and A. Keane, "A steady-state voltage stability analysis of power systems with high penetrations of wind," *IEEE Transactions on Power Systems*, vol. 25, no. 1, pp. 433–442, 2010.

[68] Z. Deng, B. Wang, Y. Xu, T. Xu, C. Liu, and Z. Zhu, "Multi-scale convolutional neural network with time-cognition for multi-step short-term load forecasting," *IEEE Access*, vol. 7, pp. 88058–88071, 2019.

[69] K. Nose-Filho, A. D. P. Lotufo, and C. R. Minussi, "Short-term multinodal load forecasting using a modified general regression neural network," *IEEE Transactions on Power Delivery*, vol. 26, no. 4, pp. 2862–2869, 2011.

[70] K. Methaprayoon, W. Lee, S. Rasmiddatta, J. R. Liao, and R. J. Ross, "Multistage artificial neural network short-term load forecasting engine with front-end weather forecast," *IEEE Transactions on Industry Applications*, vol. 43, no. 6, pp. 1410–1416, 2007.

[71] T. Senjyu, H. Takara, K. Uezato, and T. Funabashi, "One-hour-ahead load forecasting using neural network," *IEEE Transactions on Power Systems*, vol. 17, no. 1, pp. 113–118, 2002.

[72] E. Ceperic, V. Ceperic, and A. Baric, "A strategy for short-term load forecasting by support vector regression machines," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4356–4364, 2013.

[73] H. Jiang, Y. Zhang, E. Muljadi, J. J. Zhang, and D. W. Gao, "A short-term and high-resolution distribution system load forecasting approach using support vector regression with hybrid parameters optimization," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3341–3350, 2018.

[74] Q. Hu, S. Zhang, M. Yu, and Z. Xie, "Short-term wind speed or power forecasting with heteroscedastic support vector regression," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 1, pp. 241–249, 2016.

[75] E. E. Elattar, J. Goulermas, and Q. H. Wu, "Electric load forecasting based on locally weighted support vector regression," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 4, pp. 438–447, 2010.

[76] X. Shao and C. S. Kim, "Multi-step short-term power consumption forecasting using multi-channel lstm with time location considering customer behavior," *IEEE Access*, vol. 8, pp. 125263–125273, 2020.

[77] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2019.

[78] M. Tan, S. Yuan, S. Li, Y. Su, H. Li, and F. He, "Ultra-short-term industrial power demand forecasting using lstm based hybrid ensemble learning," *IEEE Transactions on Power Systems*, vol. 35, no. 4, pp. 2937–2948, 2020.

[79] M. Alhussein, K. Aurangzeb, and S. I. Haider, "Hybrid cnn-lstm model for short-term individual household load forecasting," *IEEE Access*, vol. 8, pp. 180544–180557, 2020.

[80] Dong-Xiao Niu, Qiang Wanq, and Jin-Chao Li, "Short term load forecasting model using support vector machine based on artificial neural network," in *2005 International Conference on Machine Learning and Cybernetics*, vol. 7, pp. 4260–4265 Vol. 7, 2005.

[81] W. Wang, L. He, P. Markham, H. Qi, Y. Liu, Q. C. Cao, and L. M. Tolbert, "Multiple event detection and recognition through sparse unmixing for high-resolution situational awareness in power grid," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1654–1664, 2014.

[82] S. Liu, Y. Zhao, Z. Lin, Y. Liu, Y. Ding, L. Yang, and S. Yi, "Data-driven event detection of power systems based on unequal-interval reduction of pmu data and local outlier factor," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1630–1643, 2020.

[83] R. Yadav, A. K. Pradhan, and I. Kamwa, "Real-time multiple event detection and classification in power system using signal energy transformations," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 3, pp. 1521–1531, 2019.

[84] A. S. Musleh, H. M. Khalid, S. M. Muyeen, and A. Al-Durra, "A prediction algorithm to enhance grid resilience toward cyber attacks in wamcs applications," *IEEE Systems Journal*, vol. 13, no. 1, pp. 710–719, 2019.

[85] V. Venkataramanan, P. S. Sarker, K. S. Sajan, A. Srivastava, and A. Hahn, "Real-time federated cyber-transmission-distribution testbed architecture for the resiliency analysis," *IEEE Transactions on Industry Applications*, vol. 56, no. 6, pp. 7121–7131, 2020.

[86] S. Xin, Q. Guo, H. Sun, B. Zhang, J. Wang, and C. Chen, "Cyber-physical modeling and cyber-contingency assessment of hierarchical control systems," *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2375–2385, 2015.

[87] Y. Meng, Z. Yu, N. Lu, and D. Shi, "Time series classification for locating forced oscillation sources," *IEEE Transactions on Smart Grid*, vol. 12, no. 2, pp. 1712–1721, 2021.

[88] Q. Zhu, J. Chen, L. Zhu, D. Shi, X. Bai, X. Duan, and Y. Liu, "A deep end-to-end model for transient stability assessment with pmu data," *IEEE Access*, vol. 6, pp. 65474–65487, 2018.

[89] V. Malbasa, C. Zheng, P. Chen, T. Popovic, and M. Kezunovic, "Voltage stability prediction using active machine learning," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 3117–3124, 2017.