

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

**ENERGY DEMAND FORECASTING IN FOG COMPUTING BASED MICROGRIDS
USING ENSEMBLE LEARNING**



M.Sc. THESIS

Tuğçe KESKİN

Department of Computer Engineering

Computer Engineering Programme

JUNE 2021

**ENERGY DEMAND FORECASTING IN FOG COMPUTING BASED MICROGRIDS
USING ENSEMBLE LEARNING**



M.Sc. THESIS

**Tuğçe KESKİN
(504171529)**

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Gökhan İNCE

JUNE 2021

**SİS BİLİŞİMİ TABANLI MİKRO ŞEBEKELERDE TOPLULUK ÖĞRENME İLE
ENERJİ TALEP TAHMİNİ**

YÜKSEK LİSANS TEZİ

**Tuğçe KESKİN
(504171529)**

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Gökhan İNCE

HAZİRAN 2021

Tuğçe KESKİN, a M.Sc. student of ITU Graduate School of ScienceEngineering and Technology 504171529 successfully defended the thesis entitled “**ENERGY DEMAND FORECASTING IN FOG COMPUTING BASED MICROGRIDS USING ENSEMBLE LEARNING**”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Asst. Prof. Dr. Gökhan İNCE**
Istanbul Teknik Üniversitesi

Jury Members : **Asst. Prof. Dr. Gökhan İNCE**
Istanbul Technical University

.....
Asst. Prof. Dr. Mehmet Tahir SANDIKKAYA
Istanbul Technical University

.....
Asst. Prof. Dr. Ziya Cihan TAYŞI
Yildiz Technical University

Date of Submission : **11 June 2021**

Date of Defense : **7 July 2021**





To my family,



FOREWORD

This thesis is made as a master project, as part of the requirements for the awarding of a degree in Master of Science in Engineering at the department of Computer Engineering at the Istanbul Technical University. I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Asst. Prof. Dr. Gökhan İNCE, my advisor for his countless hours of reflecting, reading, encouraging, and most of all patience throughout the entire process.

I would like to express my gratitude to my dear husband, Mert KESKİN, who contributed immensely to the completion of my thesis with his endless love and support.

I would also like to thank my family and friends for their support, patience, and faith in me.

JUNE 2021

Tuğçe KESKİN
Computer Engineer



TABLE OF CONTENTS

	<u>Page</u>
FOREWORD.....	ix
TABLE OF CONTENTS.....	xi
ABBREVIATIONS	xiv
LIST OF TABLES	xv
LIST OF FIGURES	xvii
SUMMARY	xix
ÖZET	xxiii
1. INTRODUCTION	1
2. LITERATURE REVIEW.....	5
2.1 Internet of Things	5
2.2 Smart Grid	5
2.2.1 Internet of Things in Smart Grid	6
2.2.2 Microgrids	8
2.2.3 Challenges for Smart Grids and Microgrids in Information Processing	10
2.3 Computing Platforms.....	11
2.4 Short-term Load Forecasting	15
3. BACKGROUND TECHNOLOGIES.....	19
3.1 Modeling Information Management in Microgrids.....	20
3.1.1 Edge Devices	20
3.1.2 Docker	23
3.1.3 Cloud	24
3.2 Ensemble Learning	24
3.2.1 ARIMA and SARIMA	25
3.2.2 SVR	27
3.2.3 LSTM	28
4. PROPOSED FRAMEWORK.....	31
4.1 System Architecture	31
4.1.1 Edge Layer.....	33
4.1.1.1 Clients.....	33
4.1.1.2 Networking and Communication.....	34
4.1.2 Fog Layer.....	35
4.1.2.1 Cluster Topology.....	35
4.1.2.2 Application Architecture	35
4.1.2.3 Application Deployment.....	36

4.1.3 Cloud Layer	36
4.1.4 Operating the Framework.....	37
4.2 Application Design.....	41
4.2.1 Services.....	42
4.2.1.1 Data Management Service	42
4.2.1.2 Training Service.....	42
4.2.1.3 Inference Service	43
4.2.2 Energy Demand Forecasting	43
4.2.2.1 Data Preprocessing	43
4.2.2.2 Ensemble Model for Short-term Load Forecasting	44
4.2.3 Implemented Features	46
4.2.3.1 Preprocessing Raw Sensor Data	46
4.2.3.2 Tuning Model Hyperparameters	47
4.2.3.3 Saving Best Models	50
4.2.3.4 Predicting Hourly Data	52
5. EXPERIMENTS AND RESULTS	55
5.1 Experimental Setup	55
5.1.1 Software and Hardware	55
5.1.2 Dataset	55
5.1.3 Training Parameters.....	56
5.1.4 Evaluation Methods and Metrics	57
5.1.4.1 Evaluation Methods of Prediction Accuracy	57
5.1.4.2 Evaluation Methods of Timing	58
5.2 Results	59
5.2.1 Results on Prediction Accuracy.....	59
5.2.2 Results on Timing.....	64
6. CONCLUSION	67
REFERENCES.....	73
CURRICULUM VITAE.....	81



ABBREVIATIONS

IoT	: Internet of Things
STLF	: Short-term Load Forecasting
VSTLF	: Very Short-term Load Forecasting
AI	: Artificial Intelligence
SVR	: Support Vector Regression
LSTM	: Long Short Term Memory
ARIMA	: Autoregressive Integrated Moving Average
SARIMA	: Seasonal Autoregressive Integrated Moving Average
MAE	: Mean Absolute Error
MAPE	: Mean Absolute Percentage Error
RMSE	: Root Mean Square Error
TCP	: Transmission Control Protocol
LAN	: Local Area Network
Wh	: Watt-hour
AMI	: Advanced Metering Infrastructure
RES	: Renewable Energy Sources
EV	: Electric Vehicles
M2M	: Machine-to-Machine
HAN	: Home Area Network
NAN	: Neighborhood Area Network
WAN	: Wide Area Network
SCADA	: Supervisory Control and Data Acquisition
ANN	: Artificial Neural Networks
RNN	: Recurrent Neural Networks
SVM	: Support Vector Machines
ML	: Machine Learning
AWS	: Amazon Web Services
EC2	: Elastic Compute Cloud
IaaS	: Infrastructure as a Service
MLP	: Multilayer Perceptron
AIC	: Akaike's Information Criterion
ACF	: Autocorrelation Function
PACF	: Partial Autocorrelation Function
HTTP	: Hyper Text Transfer Protocol
MQTT	: Message Queuing Telemetry Transfer
REST	: Representational State Transfer
LSTM_C	: LSTM _{continuous}
LSTM_B	: LSTM _{bundled}
SVR_C	: SVR _{continuous}
LSTM_B	: SVR _{bundled}

LIST OF TABLES

	<u>Page</u>
Table 3.1 : Hardware Platforms Cost Comparison.....	23
Table 4.1 : Data set.....	48
Table 4.2 : Hourly Data set.....	54
Table 5.1 : Comparison of Prediction Errors.....	60
Table 5.2 : Comparison of Execution Times	65



LIST OF FIGURES

	<u>Page</u>
Figure 2.1 : Smart Grid Architecture	6
Figure 2.2 : Microgrid Architecture	9
Figure 3.1 : SVR Kernel Mapping	28
Figure 3.2 : LSTM Memory Block	29
Figure 4.1 : Fog-based Reference Architecture	32
Figure 4.2 : System Architecture	33
Figure 4.3 : The Proposed Model Overview	45
Figure 4.4 : Ensemble Learning Pipeline.....	46
Figure 4.5 : Sequence Diagram for Preprocessing Raw Sensor Data	47
Figure 4.6 : Sequence Diagram for Tuning Model Hyperparameters.....	49
Figure 4.7 : Sequence Diagram for Saving Best Models	51
Figure 4.8 : Sequence Diagram for Predicting Hourly Data.....	53
Figure 5.1 : Autocorrelation Function of Historical Load Data.....	56
Figure 5.2 : Prediction Results of Compared Models in Two Consecutive Days .	60
Figure 5.3 : MAPE Results of the Ensemble Model against the Best and Worst Performance	61
Figure 5.4 : Number of Aggregated Houses versus MAPE Results	62
Figure 5.5 : MAPE Results: Forecasting the Aggregated Load versus Aggregating the Load Forecasts	63



ENERGY DEMAND FORECASTING IN FOG COMPUTING BASED MICROGRIDS USING ENSEMBLE LEARNING

SUMMARY

The information systems and communication technologies enable the information exchange between customers and utility providers in smart grid systems. The power flows through the subsystems (i.e., power generation, transmission, distribution, consumption), and the information flows through the networks (i.e., home area network, neighborhood area network, wide area network).

Nowadays, smart grid systems have been evolving from centralized grids into microgrids. The concept of microgrid has been introduced to manage the distributed power generation, and power supply operations with increased observability and control. Microgrids can improve grid efficiency since they are closer to the demand and incorporate renewable energy resources. Moreover, they can achieve better grid reliability as they can manage the generated and consumed power themselves. In microgrids, the goal is to maximize the use of renewable energy sources, e.g., wind or solar, instead of conventional energy resources, e.g., coal, oil, or natural gas. The development of intelligent decision-making mechanisms for energy supply in microgrids will enable more effective usage of energy resources. For this reason, utility suppliers should be able to identify the energy demand of customers as quickly and accurately as possible and respond to this demand in a short time. In this context, short-term load forecasting enables energy providers to anticipate customer demand, plan the energy generation, and minimize power outages. The power consumption data received from smart meters help utility suppliers implement energy demand management tools in microgrids.

Cloud computing can support information processing in smart grids by providing on-demand storage, networking, and computing services. However, the cloud-centric architecture has become a bottleneck for the decentralized and data-driven microgrids evolving from centralized smart grids. Fog computing, which is used interchangeably with edge computing, has emerged to reduce the burden of processing and storing big data produced by devices in edge networks. Therefore, fog computing facilitates the deployment of intelligent energy demand management tools in smart grid systems. The fog layer acts as a local data center storing the microgrid energy demand information and taking care of energy demand forecasting.

The number of studies regarding the use of fog computing in smart grids has increased. This indicates that fog computing-based smart grid applications will be employed widely. The aspects of proposed systems should be tested, such as application architecture design and deployment, infrastructure and network management, resource management, and scalability. Test environments help to minimize the costs and deployment time. Although the number of studies addressing the applicability of

fog computing in smart grids has increased, the subject is still fresh. There is also a lack of frameworks that allow modeling smart grid applications developed using fog computing.

As one of the main purposes of this thesis, we designed a fog-based modeling framework for researchers and developers who aim to develop, test, or evaluate smart grid applications. Fog computing is an appropriate paradigm to build distributed, latency-aware, and privacy-preserving energy demand applications in microgrid systems. In our framework, these distributed local grids, i.e., microgrids, utilize the fog nodes as information management centers that collect data from Internet of Things (IoT) devices such as smart meters and execute the energy demand forecasting application. Therefore, we introduced a 3-tier modeling framework comprising edge, fog, and cloud layers for microgrid architecture. In the edge layer, we simulated the functions of data generating entities in a neighborhood such as smart meters and weather stations using edge clients. In the fog layer, we modeled a local information management center through a fog cluster. We used 3 Raspberry Pi devices acting as fog nodes in the setup of the fog cluster. We also proposed an application deployment model using resource-efficient Docker container technology.

In this thesis, we also proposed an ensemble learning approach for short-term load forecasting using support vector regression (SVR) and long-short term memory (LSTM). We used the fog-based framework to deploy the application of the proposed ensemble learning approach. Firstly, we designed and implemented the microservices with particular responsibilities, namely data management, training, and inference service. These services are callable through application programming interfaces (APIs) to execute the ensemble learning pipeline.

We utilized the publicly available REFIT electrical load dataset to build the energy demand forecasting model. The dataset includes power measurements from 20 homes in Loughborough, UK, as well as air temperature measurements. We used the load, temperature, and temporal (i.e., month, day of the week, and hour) features in model training. We compared the performances of the trained models on the unbiased test set. We used mean absolute percentage error (MAPE), root mean squared error (RMSE), and mean absolute error (MAE) as the evaluation metrics. The proposed ensemble model gave the most accurate results among the sub models, with RMSE of 132.5 Wh, MAPE of 19.9%, and MAE of 103.5 Wh.

To evaluate the timing performance of the fog nodes compared to the cloud, we studied the network latency and execution time of the ensemble model training. We measured the latency using Transmission Control Protocol (TCP) for both scenarios, where the data is propagated within the job requests from the master fog node to worker nodes or the cloud node. As a result, the average latency was lower between the master and worker fog nodes located in a Local Area Network (LAN). The average latency between the master fog node and the cloud node is higher due to multi-hop communication. The execution time in fog and cloud layers was also evaluated. We trained multiple SVR and LSTM models in Docker containers concurrently in the fog nodes. In the first scenario, we ran the training service with the same number of replicas, only changing the number of fog nodes (i.e., one master fog node, one master fog node and one worker fog node, or all fog nodes). As a result, when we further spread the containers into more fog nodes, the execution time decreased. In the second scenario, we just updated the training service configuration to define more

replicas, keeping the number of fog nodes constant. It provided a reduced execution time. As expected, training in the cloud node yielded the best execution time because of more powerful processors. In conclusion, we observed that the fog framework is promising for the execution of complex models such as the proposed ensemble model. Furthermore, it can help achieve execution times approaching the cloud with its distributed and cost-effective resources.





SİS BİLİŞİMİ TABANLI MİKRO ŞEBEKELERDE TOPLULUK ÖĞRENME İLE ENERJİ TALEP TAHMİNİ

ÖZET

Bilgi sistemleri ve iletişim teknolojileri, akıllı şebeke sistemlerinde müşteriler ve hizmet sağlayıcılar arasında bilgi alışverişini sağlar. Güç, alt sistemler (yani, enerji üretimi, iletimi, dağıtımı, tüketimi) aracılığıyla akarken, bilgi ise ağlar (yani, ev alan ağı, komşu alan ağı, geniş alan ağı) üzerinden akar.

Günümüzde akıllı şebeke sistemleri, merkezi şebekelerden mikro şebekelere dönüşmektedir. Dağıtılmış güç üretimini ve artan gözlemlenebilirlik ve kontrol ile güç kaynağı operasyonlarını yönetmek için mikro şebeke kavramı tanıtılmıştır. Mikro şebekeler, talebe daha yakın oldukları ve yenilenebilir enerji kaynaklarını içerdiği için şebeke verimliliğini artırabilir. Ayrıca, üretilen ve tüketilen gücü kendileri yönetebildikleri için daha yüksek şebeke güvenilirliği elde edilmesine yardımcı olabilirler. Mikro şebekelerde amaç, kömür, petrol veya doğal gaz gibi geleneksel enerji kaynakları yerine rüzgar veya güneş gibi yenilenebilir enerji kaynaklarının kullanımını en üst düzeye çıkarmaktır. Mikro şebekelerde enerji temini için akıllı karar verme mekanizmalarının geliştirilmesi, enerji kaynaklarının daha etkin kullanılmasını sağlayacaktır. Bu nedenle şebeke tedarikçileri, müşterilerin enerji talebini mümkün olan en hızlı ve doğru şekilde tespit edebilmeli ve bu talebe kısa sürede cevap verebilmelidir. Bu bağlamda, kısa vadeli yük tahmini, enerji sağlayıcılarının müşteri talebini tahmin etmesine, enerji üretimini planlamasına ve elektrik kesintilerini en aza indirmesine olanak tanır. Akıllı sayaçlardan alınan güç tüketimi verileri, kamu hizmeti tedarikçilerinin mikro şebekelerde enerji talebi yönetimi araçlarını uygulamalarına yardımcı olur.

Bulut bilişim, isteğe bağlı depolama, ağ oluşturma ve bilgi işlem hizmetleri sağlayarak akıllı şebekelerde bilgi işlemeyi destekleyebilir. Ancak bulut merkezli mimari, merkezi akıllı şebekelerden gelişen merkezi olmayan ve veri odaklı mikro şebekeler için bir darboğaz haline gelmiştir. Kenar bilişim ile birbirinin yerine geçerek kullanılabilen sis bilişim, kenar ağlarda cihazların ürettiği büyük verilerin işlenmesi ve saklanması yükünü azaltmak için ortaya çıkmıştır. Bu nedenle, sis bilişim, akıllı şebeke sistemlerinde akıllı enerji talebi yönetimi araçlarının dağıtımını kolaylaştırır. Sis katmanı, mikro şebeke enerji talebi bilgilerini depolayan ve enerji talebi tahminini gerçekleştiren yerel bir veri merkezi görevi görür.

Sis bilişimin akıllı şebekelerde kullanımına ilişkin çalışmaların sayısı artmıştır. Bu da sis hesaplama tabanlı akıllı şebeke uygulamalarının yaygın olarak kullanılacağını göstermektedir. Önerilen sistemlerin uygulama mimarisi tasarımı ve dağıtımı, altyapı ve ağ yönetimi, kaynak yönetimi ve ölçeklenebilirlik gibi yönleri test edilmelidir. Test ortamları, maliyetleri ve dağıtım süresini en aza indirmeye yardımcı olur. Sis bilişimin akıllı şebekelerde uygulanabilirliğini ele alan çalışmaların sayısı artmış olsa

da konu hala tazedir. Sis bilişimi kullanılarak geliştirilen akıllı şebeke uygulamalarının modellenmesine izin veren çerçevelerin eksikliği gözlenmektedir.

Bu tezin temel amaçlarından biri olarak, akıllı şebeke uygulamalarını geliştirmeyi, test etmeyi veya değerlendirmeyi amaçlayan araştırmacı ve geliştiriciler için sis tabanlı bir çerçeve modeli tasarladık. Sis bilişim, mikro şebeke sistemlerinde dağıtılmış, gecikmeye duyarlı ve gizliliği koruyan enerji talebi uygulamaları oluşturmak için uygun bir paradigmadır. Çerçevemizde, bu dağıtılmış yerel şebekeler, yani mikro şebekeler, sis düğümlerini, akıllı sayaçlar gibi Nesnelerin İnterneti (IoT) cihazlarından veri toplayan ve enerji talebi tahmin uygulamasını yürüten bilgi yönetim merkezleri olarak kullanır. Bu nedenle, mikro şebeke mimarisi için kenar, sis ve bulut katmanlarından oluşan 3 katmanlı bir modelleme çerçevesi sunduk. Kenar katmanında, kenar istemcilerini kullanarak akıllı sayaçlar ve hava istasyonları gibi bir mahalledeki veri üreten varlıkların işlevlerini simüle ettik. Sis katmanında, bir sis kümesi aracılığıyla yerel bir bilgi yönetim merkezi modelledik. Sis kümesinin kurulumunda sis düğümü görevi gören 3 adet Raspberry Pi cihazı kullandık. Ayrıca, kaynak açısından verimli Docker konteynır teknolojisini kullanan bir uygulama dağıtım modeli önerdik.

Bu tezde, destek vektör regresyonu (SVR) ve uzun-kısa süreli bellek (LSTM) kullanarak kısa süreli yük tahmini için bir topluluk öğrenme yaklaşımı da önerdik. Önerilen topluluk öğrenme yaklaşımının uygulamasını dağıtmak için sis tabanlı çerçeveyi kullandık. İlk olarak, veri yönetimi, eğitim ve çıkarım servisi gibi belirli sorumlulukları olan mikro servisleri tasarladık ve uyguladık. Bu servisler, topluluk öğrenme ardışık düzenini yürütmek için uygulama programlama arabirimleri (API'ler) aracılığıyla çağrılabilir.

Enerji talebi tahmin modelini oluşturmak için halka açık REFIT elektrik yükü veri setini kullandık. Veri seti, Loughborough, İngiltere'deki 20 evden alınan güç ölçümlerinin yanı sıra hava sıcaklığı ölçümlerini içerir. Model eğitiminde yük, sıcaklık ve zamansal (yani ay, haftanın günü ve saat) özniteliklerini kullandık. Eğitilmiş modellerin performanslarını tarafsız test seti üzerinde karşılaştırdık. Değerlendirme ölçütleri olarak ortalama mutlak yüzde hatası (MAPE), ortalama karesel hata (RMSE) ve ortalama mutlak hata (MAE) kullandık. Önerilen topluluk modeli, 132,5 Wh RMSE, % 19,9 MAPE ve 103,5 Wh MAE ile alt modeller arasında en doğru sonuçları verdi.

Sis düğümlerinin buluta kıyasla zamanlama performansını değerlendirmek için topluluk modeli eğitiminin ağ gecikmesini ve yürütme süresini inceledik. Gecikmeyi, verilerin ana sis düğümünden çalışan düğümlere veya bulut düğümüne iş istekleri içinde yayıldığı her iki senaryo için İletim Kontrol Protokolü'nü (TCP) kullanarak ölçtük. Sonuç olarak, bir yerel alan ağında bulunan ana ve çalışan sis düğümleri arasındaki ortalama gecikme daha düşüktür. Ana sis düğümü ile bulut düğümü arasındaki ortalama gecikme, çok sekmeli iletişim nedeniyle daha yüksektir. Ayrıca sis ve bulut katmanları için yürütme süresi de değerlendirildi. Sis düğümlerinde aynı anda Docker konteynırlarında birden fazla SVR ve LSTM modelini eğittik. İlk senaryoda, eğitim hizmetini aynı sayıda replika ile çalıştırarak yalnızca sis düğümlerinin (yani bir ana sis düğümü, bir ana sis düğümü ve bir çalışan sis düğümü veya tüm sis düğümleri) sayısını değiştirdik. Sonuç olarak, konteynırları daha fazla sis düğümüne yaydığımızda yürütme süresi kısaldı. İkinci senaryoda, sis düğümlerinin sayısını sabit tutarak, sadece servis yapılandırmasını daha fazla sayıda replika tanımlayacak şekilde güncelledik. Bu

durum, azalan bir yrtme sresi saėladı. Beklendiėi gibi, bulut dėmnde eėitim, daha gçl iřlemciler nedeniyle en iyi yrtme sresini saėladı. Sonu olarak, sis çerevesinin nerilen topluluk modeli gibi karmařık modellerin yrtlmesi iin umut verici olduėunu ve daėıtık ve uygun maliyetli kaynaklarıyla buluta yaklařan yrtme srelerinin elde edilmesine yardımcı olabileceėini gzlemledik.





1. INTRODUCTION

The smart grid is a communication network that allows collecting and analyzing data from different components of the electricity grid to predict power demand and supply for better power management [1]. Recently, microgrids, which are small-scale and decentralized versions of smart grids, are used in electrical systems. These networks enable using distributed power generators and renewable energy sources. Microgrids are closer to demand than traditional grids, resulting in transmission reduction and efficiency gains [2]. To ensure the interaction between microgrids and the distribution network, the demand and generation needs in the microgrid must be determined. Therefore, the development of short-term load forecasting (STLF) applications, from hours to weeks, is a priority research subject [3].

Microgrids require computing platforms for the storage and processing of information. Cloud-based applications are recommended as useful solutions for demand management efficiency due to their massive storage and computing mechanisms [4]. However, the cloud computing paradigm alone is insufficient to meet the needs of decentralized and data-driven microgrid architecture and should include a complementary paradigm. For example, the communication delay resulting from distant cloud servers is unacceptable for some *real-time applications* in microgrids. Setting prices in the electricity market according to the consumption demands of consumers requires a more accurate short-term demand forecast and thus the best purchase prices in real-time [5]. Energy demand forecasting is also important for a reliable power supply. Since electrical energy cannot be stored for future use, generating as much energy as needed in real-time is important for energy efficiency [6]. Besides, central cloud systems are not sufficient for the distributed nature of microgrids integrated with advanced metering infrastructure (AMI), renewable energy sources (RES), and electric vehicles (EVs). There is an increasing need to run *distributed intelligence*, in other words, intelligent optimization algorithms on distributed processing nodes to manage increased variability in the smart grid [7].

Moreover, Internet of Things (IoT) data collected usually contains sensitive or private information about the users' consumption behavior. Sending raw data to a public cloud accessible to third parties will raise privacy and security concerns. Therefore, a distributed architecture that protects *security and privacy* is needed [8].

Fog computing resolves latency problems since it allows processing data near to the data source. Furthermore, fog nodes are efficient computing resources due to their virtualization capabilities. They play a role in processing IoT data for short-term tasks and sending the processed data to the cloud system for long-term analysis [9]. Hence, fog computing is an appropriate paradigm to build distributed, latency-aware, and privacy-preserving energy demand applications in microgrid systems.

Research on the integration of smart grids and fog computing has come to the fore in recent years [2, 5, 7, 8]. However, there is a lack of frameworks using low-latency and distributed fog nodes to validate the artificial intelligence-based smart grid applications in the literature. As the key contribution of this thesis, we proposed a fog-based modeling framework for researchers and developers who aim to develop, test, or evaluate smart grid applications. Therefore, we used Raspberry Pi devices to build a fog cluster. We also described an application deployment model based on resource-efficient Docker container technology for the framework.

For demand response management, the energy generation must be planned based on users' energy demand. Therefore, STLF, from hours to weeks, plays a crucial role in predicting the energy demand based on users' historical consumption data. The methods of energy consumption prediction can be roughly presented in three categories [10,11]: 1) statistical-based approaches, e.g., autoregressive integrated moving average (ARIMA), seasonal autoregressive integrated moving average (SARIMA); 2) machine learning approaches, e.g., support vector regression (SVR), random forest regression (RFR), extreme gradient boosting; 3) deep learning approaches, e.g., artificial neural networks (ANNs), long-short term memory based recurrent neural network model (LSTM based RNN). As the prediction capabilities can differ among the mentioned learning methods, the ensemble learning approach is recently used to obtain better prediction results for energy consumption prediction [6, 12–14]. Ensemble learning takes the results of multiple predictions produced by the models. Then, it calculates

an outcome value based on a statistical approach, such as the median, average, and weighted average of the prediction values.

In this thesis, we also proposed an ensemble learning approach for energy demand forecasting. We deployed the proposed energy demand forecasting application with its software components and performed a set of experiments. We aimed to demonstrate the potential of distributed and low-latency fog nodes in executing complex models such as the proposed ensemble model.

In Chapter 2, we explained the relationship between the smart grid and IoT. Then, we discussed the interest of energy demand management with the emergence of microgrids to manage the supply operations. We also introduced the information processing challenges in smart grids and microgrids. Then, we reviewed the studies related to the computing platforms used in grid information management. Moreover, we reviewed the studies about short-term load forecasting for energy demand management applications.

In Chapter 3, we defined the problem formulation about testing the microgrid applications. We also introduced the computing technologies that we used for modeling the microgrid information management system. We also explained the learning methods that we used in the proposed energy demand forecasting application.

In Chapter 4, we introduced the layers of the proposed fog-based framework for modeling the microgrids and operation of the proposed framework. We also presented the design phases of the energy demand forecasting application, including the services, ensemble method, and implemented features.

In Chapter 5, we talked about the results of our experiments on the fog-based framework regarding prediction accuracy in particular cases, timing properties, namely network latency and execution time.

In Chapter 6, we evaluated the results of the thesis and discussed the future plans.

2. LITERATURE REVIEW

2.1 Internet of Things

IoT is a revolutionary paradigm rapidly growing in information and communication technology with the advent of wireless communications [15]. IoT accelerates the transformation of traditional devices by leveraging fundamental technologies such as embedded devices, communication technologies, Internet protocols and sensor networks [16]. IoT is a real-world model of the Internet, comprising every uniquely identifiable object. This vision refers to ubiquitous computing [17]. IoT enables smart objects to communicate with each other for information sharing and decision-making. Moreover, IoT establishes a broad network between people, devices, and machines. Cisco estimates that there will be 29.3 billion connected devices in use around the world by 2023. The machine-to-machine (M2M) devices will account for 51% of all networked devices around the world [18]. The growth of IoT gains speed along with the reduction in the size and price of IoT devices as well as improvements in data analytics. IoT introduces various protocols and empowers services such as smart homes, wearables, intelligent transportation systems, smart cities, smart grids, and healthcare [16].

IoT has the following characteristics [19]:

1. Comprehensive sense: Collecting data of objects through sensors
2. Reliable transmission: Accurate data transfer via communication networks
3. Intelligent processing: Analyzing big data using computing techniques such as cloud computing to control objects

2.2 Smart Grid

Smart grid is a communication network to collect and analyze data from distinctive components of the electricity grid to forecast power demand and supply that leads to

better power management [1]. The smart grid contains four fundamental subsystems: power generation, transmission, distribution, and utilization.

2.2.1 Internet of Things in Smart Grid

IoT services can support the smart grid functionalities by incorporating communication technologies, sensors, actuators and smart meters. Smart grids are built upon two types of communication flow. The first flow occurs from sensors and appliances to smart meters, while the second flow is between smart meters and energy suppliers' data centers. For the first flow, power line communications or wireless technologies such as ZigBee, Bluetooth, and ultra-wideband are used. The second flow is accomplished through cellular networks such as 3G and 4G or the Internet [17, 20]. Employed IoT applications enhance real-time and bidirectional data reading among the electricity corporation, customers, and power equipment in smart grid [19, 21, 22].

Fig. 2.1 shows smart grid architecture presenting power subsystems, power flow and information flow. Smart grid is formed by three categories of networks which are a wide area network (WAN), a neighborhood area network (NAN) and a home area network (HAN). The power flows through the subsystems and the information flows through the networks [23].

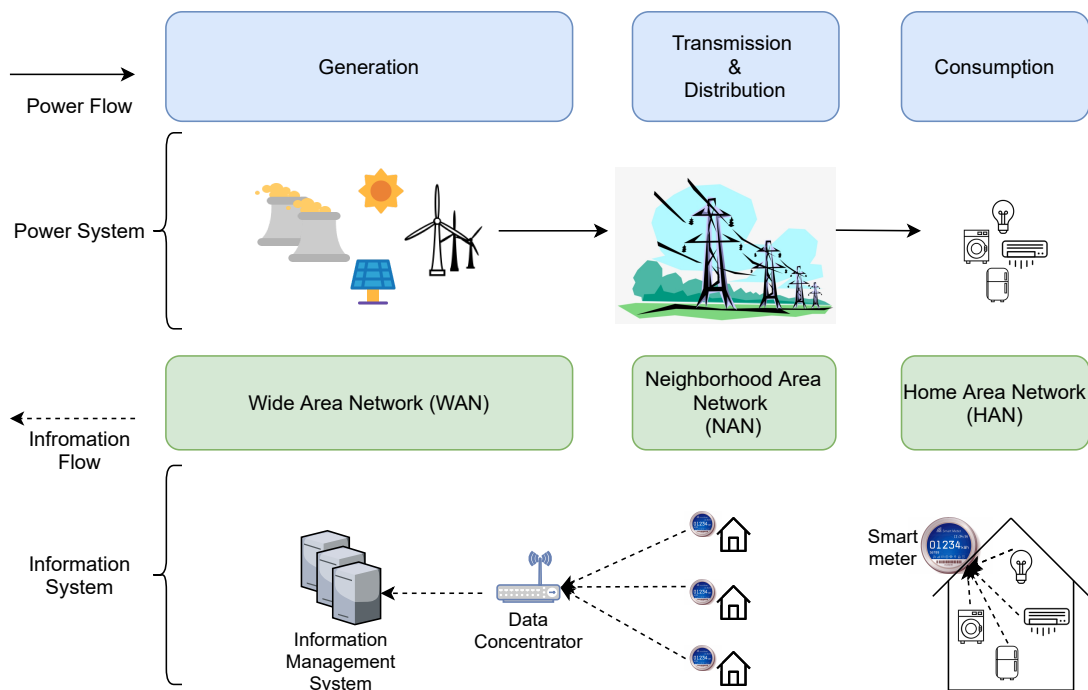


Figure 2.1 : Smart Grid Architecture

HAN is deployed within residential units or industrial plants. Smart devices, home appliances, electric vehicles, and renewable energy sources such as solar panels can connect via a HAN. Electrical appliances send the power consumption data to smart meters; therefore, consumers' power demands are managed in this network. Bluetooth, family standards of IEEE 802.11, power line communication, or ZigBee technologies are utilized for home area networks. Furthermore, NAN is the second layer of the smart grid, consisting of smart meters located on different HANs. NAN facilitates communication between distribution substations and field electrical devices for the distribution layer. NAN enables collecting the metering information from multiple HANs and transmits this information to WAN through data concentrators using the communication technologies such as family standards of IEEE 802.11, 3G and 4G wireless cellular networks, and optical networks. WAN is the third layer of a smart grid, and it acts as a bridge for the network gateways or data concentrators. WAN enables communication among power transmission, generation systems, renewable energy sources, and control centers. Long-range and high-bandwidth communication technologies such as fiber optic and wireless cellular networks, e.g., WiMAX, LTE, and LTE advanced, can be used in WAN [7, 17, 23].

To accomplish the communication and information flow, a number of IoT architectures have been proposed for smart grid systems. Layered IoT architectures are widely used (e.g., 3-layer or 4-layer) in the literature. Lloret et al. [24] proposed a 3-layers IoT architecture when classifying all components in smart grids. Layer 1 includes smart meters, network devices, and communication protocols to enable smart metering via the Internet. Layer 2 is formed by the devices receiving data on the utility side, while layer 3 is designed for intelligent systems to deliver information to the decision systems and billing systems. In particular works, 3-layer IoT architecture is extended to 4-layers architecture to represent the smart grids better. Wang et al. [25] defined a 4-layers IoT architecture for the smart grid system consisting of the terminal layer, field network layer, remote communication layer, and master station system layer. Remote terminal units and smart meters in the terminal layer gather information from IoT devices and transmit this information to the field network layer. The field network layer can use wired or wireless communication based on IoT device type to transfer the collected data to a remote communication network layer. Wired (e.g., optical) and

wireless (e.g., 2G,3G, LTE) networks support remote communication network layer. This layer acts as middleware between IoT devices and the master station system layer. The master station system layer aims at controlling and managing smart grid functions, including generation, transmission, distribution, utilization.

Since the layered architectures are generic for many IoT applications, the specific aspects of smart grid, such as the networks (i.e., HAN, NAN, WAN) and functionalities (i.e., power generation, transmission, distribution, consumption) are covered by the alternative architectures for smart grids in the literature [22]. For example, cloud-based smart grid architectures utilize cloud computing as a key design component for data storage and computation in smart grids. Pan et al. [26] used a cloud-based smart grid architecture in their proposal of an energy-efficient IoT testbed that can be employed in a wide range of buildings and help energy savings on different levels. Having used information and communication technologies such as mobile smartphones with location service, distributed control, and cloud computing, the authors involve the individuals and organizations in the energy-saving process to build energy-saving policies and use them in decision making. Mohanty et al. [27] proposed web-enabled smart grid architecture where both non-renewable and renewable energy sources are connected to digital energy meters. The household energy consumption data collected by digital energy meters is firstly sent to IoT gateways. Then the collected data is periodically transmitted to the servers where web services are running. Thus, the end-users can utilize the data for monitoring or further analysis.

2.2.2 Microgrids

Smart grid is proposed as a solution for common electricity grid issues, including unpredictable power outages, consumer frauds, high greenhouse gas and carbon emission, and inflexible electricity prices. For instance, the peak hour energy demand needs to be overestimated to eliminate the power outage risks. Therefore, generating more electricity increases the utility costs [28,29]. However, it has become essential to maintain *efficiency* and *reliability* in smart grids simultaneously. The grid efficiency corresponds to the efficient use of power consumption and power system utilization. It can be improved with various mechanisms, such as renewable energy sources and energy demand management. Besides, the grid reliability is to guarantee the service to

customers at all times. The reliability can be achieved by distributed energy generation, fault detection, and predicting peak demand [5, 17].

Distributed energy generation enables renewable energy sources to go into service. Energy suppliers can place in new locations, especially energy sources near consumption points [3]. The concept of *microgrid* has been introduced to manage the distributed power generation, and power supply operations with increased observability and control [30]. Microgrids are interconnected local smart grids with renewable energy sources and local battery storage [2], as shown in Fig. 2.2. These microgrids are capable of self-generating energy by using wind turbines and solar panels. They can be connected to the central system or isolated when potential risks such as failure and intrusion occur. Microgrids can improve the reliability of local distributions since these grids manage the generated and consumed power themselves. They have their power supply and control systems. Moreover, microgrids exchange information with other elements such as energy suppliers and service providers in the smart grid [4, 31].

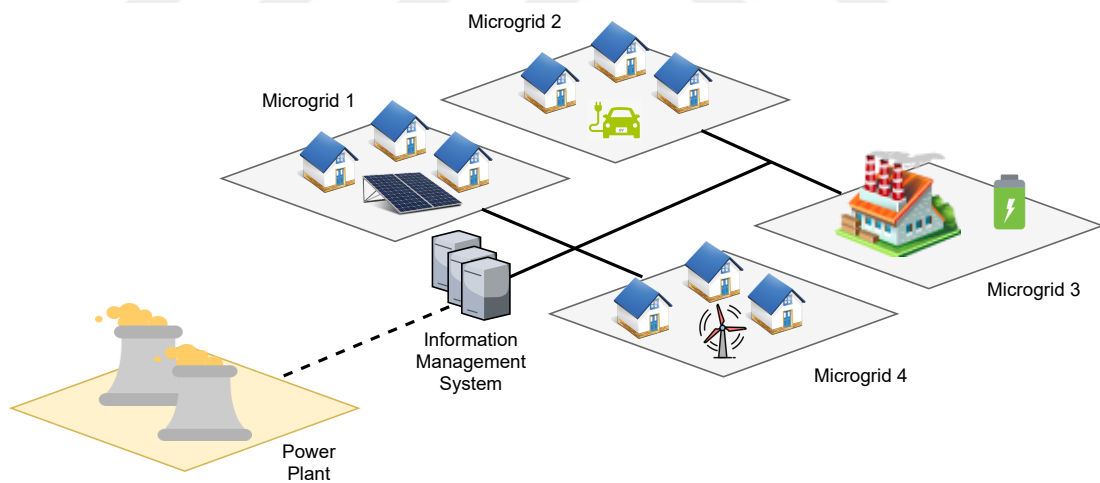


Figure 2.2 : Microgrid Architecture

The benefits of micro-grid architecture are as follow [4, 32]:

1. Electricity distribution is shifted from centralized facilities towards decentralized micro-grids.
2. Micro-grids facilitate the penetration of low-carbon, renewable energy generation.
3. Micro-grids enable efficient and reliable operation, as compared with traditional energy management mechanisms.

4. Advanced real-time energy monitoring systems can be applied.
5. Service reliability can be achieved through micro-grids that can interact with each other in the presence of outages or critical load demand.

Energy demand management is enhanced with the integration of communication technologies and information management systems into smart grids. The implementation of energy demand management tools in microgrids ensures increased reliability and management of the impacts of renewable energy sources.

Studies related to energy demand management in microgrids gain attention [30,33,34]. The referred studies mainly address the topics such as optimizing the demand response operation and energy demand forecasting in microgrids. Wang et al. [30] proposed a genetic algorithm to reduce the operation costs of microgrids, considering the distributed generation, environmental factors, and demand response. Yaprakdal et al. [33] proposed an operational scheduling mechanism in microgrids using two models to forecast daily photovoltaic power generation and electrical demand. Moradzadeh et al. [34] applied a hybrid model of machine learning methods for short-term load forecasting, using the data collected in a rural microgrid in Africa. López et al. [21] developed a NAN model and evaluated M2M communications on short-term and medium to long-term scenarios for demand response.

2.2.3 Challenges for Smart Grids and Microgrids in Information Processing

- Limited-capacity IoT devices: Since the resources of IoT devices, e.g., batteries, memory, and bandwidth, are limited, it is not possible to send all information to the destination. Thus, for efficient resource usage and data acquisition, data fusion techniques should be utilized to collect and aggregate data [17].
- Network delay: It is required to optimize the smart grid network by employing the ideal number of gateway and IoT devices. Unless the number of connections between gateways is reduced, congestion occurs and causes delay and packet loss. Since IoT devices resend the data, the increased re-congestions degrades the grid performance [17].

- **Distributed energy generation:** The usage of distributed generation sources in smart grids makes the control of the grid more challenging. Since renewable distributed generation sources, e.g., wind, solar, have a random nature, predictions of energy demand are dependent on the power generated by renewables. It makes the demand prediction more complicated [3].
- **Processing of big data:** The IoT devices such as sensors and smart meters in a smart grid produce big data, and hence these devices consume too much energy and cause a bottleneck. Smart grid design should allow efficient storage and processing of a vast amount of collected data [17].
- **Security and privacy:** Secure transmission and storage of data are essential for billing purposes and grid control operations. Effective measures and security standardization efforts are required to prevent cyberattacks and to avoid information leakage in smart grids [20].

2.3 Computing Platforms

To address the challenges mentioned in Section 2.2.3, energy companies should have the infrastructure with software and hardware capabilities. To fulfill the specific computational needs of smart grids, cloud-based solutions have already been proposed as mentioned in Section 2.2.1. There are several reasons why smart grid architectures with cloud platforms are widely accepted. Firstly, it leads to improvements in the operation of the Supervisory Control and Data Acquisition (SCADA) system, which is a decision-making element in smart grids. It collects data from the distributed grid assets for monitoring and controlling. The size of smart grids increases, and it is challenging to update and upgrade SCADA systems that are employed in local machines. Thus, cloud computing enables the operating SCADA systems, as it provides on-demand access to the pool of computing resources [23]. Secondly, the integration of IoT to smart grids introduces the necessity of managing, processing, and storing massive amounts of data collected from IoT devices and gateways. Thus, the energy companies utilize unlimited storage and computing resources of cloud platforms to achieve information management. Furthermore, smart grid functionalities such as demand response, distributed energy generation, or dynamic pricing require

heterogeneous architecture. Cloud computing allows heterogeneity by providing different software applications for customers and utility providers [4].

Pau et al. [35] proposed a cloud-based smart metering solution for smart grid applications. This architecture allows interoperability with the abstraction of different level IoT devices in the device integration layer. It also enables flexibility of the communication with the smart meters through publish/subscribe or request/response models and provides interfaces to distribution grid services in the middleware layer. The authors deployed and evaluated the state estimation model to reconfigure the smart grid into the application layer. Atzori and Luigi [36] proposed a cloud-based smart grid architecture concerning reusability, elasticity, and security. To enhance smart grid devices' capabilities and boost new services in a distribution grid, the authors proposed a virtualization middleware that puts together data from different domains such as weather and energy consumption. This architecture allows reusing the data from real-world objects without privacy violation through the access keys. The studies reveal that cloud-based architectures allow a large set of smart grid applications such as state estimation, demand response, non-intrusive load monitoring. Cloud platforms enrich the smart grid systems with scalability, flexibility, reusability, and interoperability due to data storage and computing capabilities. Although cloud computing unlocks new applications for smart grid management, the cloud-centric approach has shortcomings for the information management of smart grid systems that have evolved into distributed microgrids.

The limitations of cloud-centric smart grids are listed below [5]

- The geographically distant cloud servers need multi-hop communication. The yielded communication delay is not acceptable for some real-time applications.
- The raw data transmitted from IoT devices and smart meters to the cloud causes high bandwidth usage and unexpected costs. The imposed load on the communication network causes unreliable latency and response time.
- The accumulated IoT data usually holds sensitive or private information about the user. Sending the raw data directly to a public cloud raises privacy and security concerns as any third party can access them.

The real-time data collected via AMI can give grid operators a more solid understanding of large-scale and highly dynamic smart grid systems. The entities producing massive data such as SCADA systems, smart meters from AMI, and various sensors are scattered across the networks such as HANs and NANs [37]. Therefore, the cloud-centric data processing architectures will not be sufficient to handle the big data explosion. As discussed in Section 2.2.2, microgrids have emerged due to the need for smart grid operations reliability. Thus, a scalable and secure distributed data processing architecture requires to achieve reliable smart grid operations. For this reason, smart grid management needs a complementary computing paradigm to store and process data close to where it is produced. This requirement led to the use of fog computing in the context of the smart grid. Fog computing is an architectural setup for distributed processing where application logic is embedded in remote clouds and across the intermediate infrastructure components such as portable devices, gateways, or smart meters in microgrids [37]. Thus, a proportion of data storage, caching and processing is offloaded to geographically dispersed nodes called fog nodes. Only partially-processed or final results are dispatched to the cloud layer rather than the raw data for the smart grid information processing [5]. Furthermore, the communication cost between the service providers and AMI infrastructures is reduced as they will be close to each other [7].

Recently, the number of studies addressing the integration of fog computing in smart grid management has increased, and various fog-based architectures have also been proposed. Akram et al. [38] presented a 3-tier cloud-fog architecture containing fog nodes geographically dispersed to forward demand requests to the microgrids. The fog layer that gets the user demand information via smart meters in AMI communicates with the microgrid to fulfill the user demands. In this work, to manage user requests related to energy demands in fog nodes, a couple of load balancing techniques, such as Round-Robin, Particle Swarm Optimization, and Ant Colony, were employed and evaluated in terms of response time. Singh and Yassine [9] proposed an IoT data analytics platform that uses the fog computing paradigm for smart grids. As a case study, k-means clustering is applied on time-series energy consumption data of appliances to find out how frequently and when the consumer has used a given appliance. Pre-processing, pattern mining, and classification are performed in the data

analytics engine located in fog nodes. Thus, computationally intensive short-term tasks are offloaded to the fog layer, allowing analysis of large historical data to be concerned in the cloud layer. Yan and Su [39] proposed a reliable, cost-effective, and efficient smart meter infrastructure using fog computing paradigm. They validated the solution on a testbed comprising a cluster of microcomputers. In this decentralized approach, fog nodes called Datanodes are responsible for data storage and processing of smart meter data. The proposed fog computing approach resulted in a lower average processing time compared to conventional centralized approach. When the data size increased, the superiority of the distributed solution in terms of the average processing time was more observable. Barik et al. [40] proposed a fog computing-based approach for information management in microgrids. In order for the energy stored in the battery energy storage system to be used during critical situations in the grid, a mechanism that will ensure the communication between the grid assets, i.e., substations, microgrids, and consumers, is required. The data from data storage can also be sent to the cloud layer when needed. Distributed energy sources such as solar panels or electric vehicles are used in the energy storage system. To assist smart energy management and demand reduction, Tom et al. [41] applied the short-term load prediction using time series analysis in a load dataset and did the discriminant analysis to find the appliances taking the biggest share during peak demand hours. They proposed to use a fog-based architecture to perform forecasting and determinant analysis, as fog nodes provide efficiency with distributed data processing where these steps have to be done for every house in the city. Okay and Ozdemir [8] proposed a fog computing-based smart grid model and data aggregation approach to ensure privacy. The smart meters label the appliance level consumption data as private. The smart meters encrypt and authenticate this data with a key shared between the customer and cloud server before transmitting the data to the target fog node. Nevertheless, the total consumption and production data are labeled as public by smart meters. In the fog layer, the fog nodes are responsible for aggregating public and private data separately to reduce stored and transmitted data.

These studies have focused on different aspects of fog computing in smart grids, such as grid optimization, security, or load analysis. Fog computing-based mechanisms can support a large set of information management services in the era of microgrids due

to their compatibility with distributed systems. Therefore, fog computing is promising for the reliability and efficiency of grid systems.

2.4 Short-term Load Forecasting

As mentioned in Section 2.2.2, the improvements in microgrids operations allow sustainable energy systems in the microgrids. Load forecasting is considered as one of the most significant solutions to energy management in power systems, particularly in microgrids [34]. It is used for scheduling of electricity generation, therefore it is essential to energy suppliers in meeting the energy demands of customers [42].

Regardless of the forecasting model used, energy consumption forecasting can be classified based on *the forecasting horizon* which refers the period of forecasting and *the aim of the forecast* which corresponds to the number of the values to forecast [3].

In terms of horizons, forecasting is divided into four categories which are *very short-term load forecasting* (VSTLF) from second or minutes to several hours, *short-term load forecasting* (STLF) from hours to weeks, *medium-term and long-term load forecasting* (MTLF and LTLF) from months to years [3]. MTLF and LTLF are important when constructing new generation units and setting prices for power companies. SLTF is used to adjust the electricity generation, schedule the maintenance, fuel allocation. On the other hand, VSLTF is used for real-time control, and security evaluation [43]. STLF and VSTLF gain significance with the introduction of the smart grid. Smart meters produce power consumption data belonging to a household, suburb, or region at short time intervals (i.e., minutes to hours). Therefore, the prediction of power consumption for the next couple of days in every smart home can be made. The predicted data allows customers to interpret the usage and determine the motives to optimize the electricity use. Besides, STLF helps utility providers in the scheduling of electricity generation. Forecasting the generated power makes utility providers aware of the power generation in the following days. Thus, they provide pricing information to consumers based on the production. STLF enables intelligent energy demand management with minimized energy waste through predictions [7,44].

Forecasting models also can be categorized based on the number of values to predict, in other words, the *aim of the forecast*. The first category includes the models that

forecast a single value, such as the load value belonging to the next hour or next day. In contrast, the second category aims to make forecasts with multiple values, for example, which demonstrates next-day hourly forecast [3].

The parameters mentioned above are associated with the motivation behind the forecast. When it comes to forecasting models, it is dependent on the decision of the researcher [3]. The methods of energy consumption prediction can be roughly presented in three categories [10], [11], [45]:

- *Statistical-based approaches:* autoregressive integrated moving average (ARIMA) Box–Jenkins model, seasonal autoregressive integrated moving average (SARIMA) model, exponential smoothing models including Holt–Winters model (HW), state space/Kalman filtering model, linear regression model and so on
- *Machine learning approaches:* support vector regression (SVR) model, random forest regression (RFR), extreme gradient boosting (XGB), etc.
- *Deep learning approaches:* artificial neural networks (ANNs) model, long-short term memory based recurrent neural network model (LSTM based RNN), etc.

Regardless of the forecast horizon and the aim of the prediction, energy consumption forecasting approaches can be categorized into linear or non-linear, and univariate or multivariate. When the electricity load forecasting model is univariate, the model is based on only load values. Nevertheless, the electricity load forecasting model can have multiple variables (multivariate) presented in the input data, such as load value, temperature, and human activities.

From the linearity perspective, statistical-based forecasting approaches assume a linear relationship between the input and output variables. If there is precise linearity in the data, these models can deal with the forecasting. These statistical-based methods have been used for many years in electrical systems and signal processing. Although these methods lack flexibility and are poor at handling non-linearities, they are still utilized in STLF methods integrated with other non-linear methods since they are convenient to develop and use [46]. In this context, the hybrid models combining the linear and non-linear methods such as ARIMA-ANN have been proposed [11, 44].

Even though the simplicity of statistical-based methods is an advantage [46], electricity load forecasting with high accuracy is a non-trivial task with non-linearities that require being fed into the forecasting model. Therefore, modern, artificial intelligence-based methods have been developed. Artificial neural networks (ANN) are in this category and widely used for load forecasting [47–49], which helps load profile analysis and load control. ANNs have been proven to be more capable of grabbing non-linear relationships. However, this technique requires a large amount of data to get a sufficient generalization capability. On the other hand, the support vector machines (SVM) are able to solve the non-linear problems with small quantities of training data, but selecting an appropriate kernel function is necessary [50]. The SVMs are used commonly for load forecasting by researchers. Furthermore, the authors in [51, 52] applied parameter optimization for model efficiency.

Recently, deep learning methods have drawn attention. In comparison with shallow learning, deep learning refers to stacking several layers and relying on stochastic optimization to complete machine learning tasks. Therefore, deep learning models improve the learning ability with the abstraction provided by a varying number of layers [53]. Instead of manually designing models, these powerful models can automatically learn the hidden relationship between prediction and historical data when large enough training data is given [50]. As one of the deep learning models, recurrent neural networks (RNN) is powerful for the time dependency of a data sequence, commonly employed in forecasting tasks. To overcome the gradient vanishing in long-term components of data, long short term memory (LSTM) cell with an extra gate was introduced [53]. Kong et al. [53] proposed LSTM based load forecasting to address the complexity of residential load forecasting. Hossain and Mahmood [54] evaluated the effect of the input sequence length, weather and time-related features on the LSTM accuracy in one-step and multi-step ahead predictions. Zheng et al. [55] proposed gated recurrent units (GRU) with a simpler architecture that produces similar accuracy with LSTM.

According to the literature review, energy demand management in a microgrid can increase the awareness of energy consumption. Therefore, electricity consumers and producers have proper programming for balance, intraday, and day-ahead markets, ensuring grid efficiency and reliability [34]. Both the computing environment and

the forecasting methodology are the main complements of intelligent energy demand management. Regardless of the forecast models, forecasting types can be classified based on a couple of criteria, such as the aim of the forecast (i.e., single, multiple values) and the forecast horizon (i.e., LTLF, MTLF, STLF, VSTLF). Short-term load forecasting, i.e., hours to days ahead, is the most important one for the demand response functionality in an energy management system. STLF requires processing and analyzing higher frequency data which are measured in seconds, minutes, or hours. When the IoT systems expand to smart grid applications, enormous data needs to be processed in near real-time to satisfy the requirements of STLF. One solution is to utilize cloud systems that deliver extensive storage and computing capabilities for these computationally intensive applications. However, it is not easy to achieve near real-time responses from cloud-based applications since they suffer the high latency resulting from data transmission between network devices and cloud servers. Fog computing resolves latency problems since it allows processing data near to the data source. Furthermore, fog nodes are efficient computing resources due to their virtualization capabilities. They play a role in processing IoT data for short-term tasks and sending the processed data to the cloud system for long-term analysis [9]. When it comes to forecasting models, the researcher identifies the model which represents the data closely as much as possible.

3. BACKGROUND TECHNOLOGIES

In compliance with the distributed nature of microgrids, the fog layer localized between IoT devices and the cloud enables the processing and storage of data collected from neighborhoods.

Application architecture design and deployment, infrastructure and network management, mobility, resource management, and scalability make fog computing modeling challenging for researchers and service providers [56]. Recently, researchers have introduced prominent simulation tools for fog computing, combining these properties in different scales [57–60]. However, there is a lack of frameworks that utilize low-latency and distributed fog nodes to validate the artificial intelligence-based smart grid applications in the literature. Therefore, we aimed to make an evaluation setup running the energy demand forecasting scenario with software and hardware components for the referred architecture. We utilize the following concepts in the core of this thesis:

- **Microgrids:** These distributed local grids supply energy to the neighborhoods, providing reliability and efficiency.
- **Internet of Things (IoT):** Smart grids and microgrids are built upon the connected IoT devices such as smart meters, gateways, and data centers for information management.
- **Fog Computing:** Distributed, low-latency, and privacy-preserving computing paradigm enables the smart grid applications, e.g., demand management, in the neighborhoods.
- **Energy Demand Forecasting:** The predicted load data helps customers to reduce their energy consumption, as well as service providers to plan their energy production.

3.1 Modeling Information Management in Microgrids

A low-latency, distributed, and privacy-preserving fog computing architecture is a promising solution to meet computing requirements in microgrids. Fog computing is a layered model consisting of physical or virtual fog nodes located between smart end devices and central cloud services. It provides local computing resources for end devices and network connectivity to central services as needed [61]. Therefore, it enables a suitable infrastructure for energy demand forecasting applications.

Test environments are useful for validating the proposed system architecture and minimizing costs and deployment time. In recent years, the low-cost single-board computers enable the development of home automation solutions together with the rising number of IoT devices and sensors. Although single-board computers have lower processing capabilities in the past, the more powerful alternatives are released to the market with the increasing demand. Moreover, they come into prominence in handling issues such as latency, network bandwidth, and data privacy concerns. For this reason, these devices can even be used as testbeds in scenarios such as medical monitoring and smart cities, as they provide a fast and efficient modeling solution [57].

3.1.1 Edge Devices

The global market for edge computing is estimated to reach USD 22,45 million by 2024. The increasing adoption of the IoT and security concerns when handling private data are driving edge computing market growth [62]. According to analyst firm Gartner, empowered edge is one of the top strategic technology trends. Through 2028, artificial intelligence (AI) capabilities will increase in the edge devices [63]. Even today, the potential of edge processing is considerably high due to increasing performance of edge devices [64]. There are currently emerging hardware platforms released in the market with different sizes, computing capabilities, and costs. Therefore, we evaluated competitor edge devices that we can use as fog nodes in the modeling solution.

In 2017, Intel launched the Movidius Neural Compute Stick, the world's first USB-based machine learning inference kit with Movidius vision processing unit (VPU) technology. It is a self-contained accelerator that brings deep neural network processing capabilities to various host devices at the edge. Also, the company

announced an upper version Movidius Neural Compute Stick 2 in 2018, which is supported by OpenVINO AI toolkit and powered by the latest generation of Intel VPU.

In 2018, Google announced Edge tensor processor unit (TPU) which is purpose-built application-specific integrated circuit (ASIC) designed to run ML models at the edge . Edge TPU Dev Board is an all-in-one development board built on a NXP i.MX 8M system-on-chip with a quad-core ARM Cortex-A53 processor and a Cortex-M4 core along with the Edge TPU coprocessor itself. It allows designing embedded systems requiring fast ML inferencing. The second offering was Edge TPU Accelerator which can be mounted as universal serial bus (USB) device to a host board such as Raspberry Pi and allows accelerated ML in the existing system. It gives the capabilities of Edge TPU coprocessor to the system.

Nvidia Jetson family, which was introduced in 2014, is a series of embedded computing boards with small size and high performance which leads to the acceleration of the machine learning applications. Jetson TK1 is built on Nvidia's Tegra K1 processor with 192-core Kepler GPU in addition to 32-bit quad-core ARM Cortex-A15 CPU. In 2015, the Tegra X1-based Jetson TX1 was launched, featuring a 256-core power-efficient Maxwell GPU and a 64-bit quad-core ARM Cortex-A57 CPU . Jetson TX2, offered in 2017, is twice as powerful as the Jetson TX1 and based on 256-core Nvidia Pascal GPU found in the Tegra X2 system-on-chip (SoC). TX2's CPU complex includes a dual-core Nvidia Denver 2 for high single-thread performance and a quad-core ARM Cortex-A57 geared for multithreading . In 2019, Nvidia introduced a much cheaper board the Jetson Nano which enables the innovation from hobbyists, developers and students. It is not powerful as Tegra X1, however a promising product which has quad-core ARM Cortex-A57 CPU and 128-core Maxwell GPU. Nvidia Jetson family brings the capability to help the users easily move AI and machine learning workloads to the edge. The benchmark results are reported in , which compares the inference performance of Nvidia Jetson Nano against the other popular hardware platforms. Jetson community projects present the collection of the applications that use Nvidia Jetson platform as the edge device in a wide range of use cases embracing the artificial intelligence.

Raspberry Pi, which still is one the most popular microcomputers and was released in 2012, is commonly used in home automation due to widespread availability [65].

There have been variations released since 2012 and Pi's performance has been improved over the years. The Raspberry Pi is a fully featured computer which has Broadcom chip with ARM CPU and Videocore VI GPU at its heart [66]. The Pi 1 and Zero are based on Broadcom chip BCM2835 with 32-bit single-core ARM1176 CPU, while Pi 2 is built on BCM2836 features 32-bit quad-core ARM Cortex-A7 cluster. Later models of Pi 2 and Pi 3 are based on BCM2837 containing 64-bit quad-core ARM Cortex A53. Pi 4 uses the architecture of BCM2711 employing quad-core ARM Cortex-A72 CPU cores. Raspberry Pi is easy to use for beginners rather than the prototype platforms targeted for more advanced users and developers. It is also convenient compared to its alternatives which cannot be used as a standalone computer. Moreover, it is very suitable for the applications which require graphic interface or the Internet which makes it a favored platform in Internet of Things point of view [65].

All of these products lead up the evolvement of the edge platforms with high performance and small footprint. Even though edge devices with AI capabilities are increasingly becoming widespread, AI at the edge is still new and it is hard for researchers to choose the hardware platform which is the best match with the requirements.

In Table 3.1 , Raspberry Pi 4 stands out with its relatively cheap price despite its computing power along with its alternatives. The increased performance in new generation Raspberry Pi 4, which is introduced in 2019, makes this device a very competitive platform when it comes to the machine learning at the edge. Allan in [67], presents the results of the benchmark between Nvidia Jetson Nano, Raspberry Pi 3 and Raspberry Pi 4 that are based on the inferencing speeds of MobileNet v1 and v2 models, powered by TensorFlow machine learning framework. When the model inferencing was executed on Tensorflow Lite which focuses on-device inference instead of traditional TensorFlow, a significant speed increase was observed in Raspberry Pi 4 against Raspberry Pi 3. The inference speed of Pi 4 was even closer to the results of Nvidia Jetson Nano utilizing TensorRT built on CUDA, Nvidia's parallel programming model. This remarkable decrease in inference time turns Raspberry Pi 4 into a competitor to the Nvidia Jetson Nano.

Table 3.1 : Hardware Platforms Cost Comparison

Hardware Platform	Processor	Cost
Intel Neural Compute Stick 2	Intel® Movidius™ Myriad™ X VPU	\$109.95 *
Google Coral USB Accelerator	Google Edge TPU coprocessor	\$84.95 *
Google Coral Development Board	quad-core ARM Cortex-A53 plus Cortex-M4F CPU, Google Edge TPU coprocessor	\$167.95 *
NVIDIA Jetson Nano Developer Kit (V3)	128-core Maxwell™ GPU, quad-core ARM Cortex-A57 CPU	\$99.00 *
Raspberry Pi 4 Model B (4 GB)	quad-core ARM-Cortex-A72 CPU, VideoCore VI GPU	\$59.95 *

* Accessed on 2020-04-29.

3.1.2 Docker

One of the enablers of AI at edge is Docker, which is an emerging container technology. It provides a solution to deploy an application with its dependencies automatically in a self-contained environment. The containers are isolated from each other and are given individual network interface [68]. Since Docker container virtualizes the operating system, but not hardware, it is much lightweight and faster than virtual machines. Containers can be used to build edge cloud on single-board affordable device clusters like Raspberry Pi. Bukhary et al. [69] evaluate Docker container technology as a platform for edge computing. In this work, Docker has been evaluated in terms of deployment and termination, resource and service management, fault tolerance and caching. Authors' conclusion is that Docker represents a good solution to be employed in edge computing contexts. In [70], the design of an IoT gateway that can be efficiently employed also in an edge computing architecture has been proposed. In particular, that study shows how to efficiently and flexibly use Docker containers in order to customize the IoT platform, by offering data processing services. Mendki [71] considered container-based virtualization as a good choice for deploying edge-level video analytics solutions utilizing a deep learning framework. This solution has negligible overhead in terms of CPU processing compared with

the bare metal deployment. Al-Rakhami et al. [68] implemented a human activity recognition scenario utilizing the Docker platform on Raspberry Pi. The study shows that it is possible to apply the intelligent edge application based upon low-cost, lightweight and efficient container platform resulting in reduced delays, more effective communication and faster decision-making. The related works emphasize that using Docker container technologies at the edge and fog computing allows managing different services, including data processing services, which makes migration to edge based approach more convenient.

3.1.3 Cloud

Cloud computing is a way to have access to flexible and scalable computing, storage, and networking resources [72]. The cloud platforms are widely accepted in smart grid architectures as mentioned in Section 2.3. Information flows to global information management systems that rely on cloud infrastructure in general. In our proposed framework, cloud servers can be used where resource-limited fog nodes are insufficient to run microgrid level applications.

The cloud providers such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud offer cloud servers to the customers. These virtual servers are hosted and delivered through computing services called Infrastructure as a Service (IaaS), namely AWS Elastic Compute Cloud¹ (EC2), Azure Virtual Machines², and Google Compute Engine³. These computing services are called Infrastructure as a Service (IaaS) which implies the customers can take virtual machines of any size and configure their networking features and data storage space.

3.2 Ensemble Learning

The predictive ability can differ among the mentioned learning methods, therefore ensemble learning approach is recently used to achieve better prediction results for energy consumption prediction. Ensemble learning takes the multiple prediction results produced by models and calculated an output value based on statistical approaches such as median, average, weighted average of the prediction values [6].

¹<https://aws.amazon.com/ec2/>

²<https://azure.microsoft.com/en-us/services/virtual-machines/>

³<https://cloud.google.com/compute>

Alam et al. [73] have built an ensemble model based on Mahalanobis distance that combines prediction results of 4 learning models, namely, ARIMA model, Recurrent Neural Network model with LSTM cells, Univariate Linear Regression and Multivariate Linear Regression to capture the household energy consumption data. Sarajcev et al. [74] proposed an approach for STLF, which uses aggregate load clustering with ensemble learning. Ensemble of the models including Gradient boosting (GB), SVM and RF are stacked with an “elastic net” linear regression. Yan et al. [13] aimed at short-term gateway load forecasting of multi-energy micro grid and implement information fusion by fetching data from renewable energy access node and gas turbine access node in the connected-grid. Fusion matrix is given as input to decision-tree model comprises of XGBoost and GBRT, and also fed into Seq2Seq network after PCA method is applied to reduce the dimension of input data. Forecasted values from both decision-tree and Seq2Seq networks are fused by weighted average method. Petrican et al. [6] evaluated the performance energy forecasting techniques consisting of SARIMA, MLP, LSTM, SVR and ensemble method on REFIT Electrical Load Measurements data set with respect to the selected features, where ensemble model gives the best prediction results.

We proposed an ensemble STLF model that incorporates SVR as a shallow model and LSTM as a deep learning model in this thesis. We also used SARIMA as the baseline STLF method for experimental work.

3.2.1 ARIMA and SARIMA

The ARIMA is a statistical model and based on the two concepts, the first one is that the forecasts are based on linear functions of the sample observations. Secondly, the purpose is to converge to the simplest models that present a proper description of the observed data. It corresponds to the *parsimony* principle [75]. ARIMA processes, also called as Box-Jenkins models, are described by George Box and Gwilym Jenkins in 1970 [76]. With the recent contributions [76] by Hyndman et al. in 1998, Box-Jenkins modeling procedure is extended to a five-stage process: (1) the data are differenced until there are no obvious trend or seasonality left in the data to satisfy the stationarity condition (*data preparation*); (2) candidate ARIMA model is identified which may have a good fit to the data (*model selection*); there are also some model selection

methods such as Akaike's Information Criterion (AIC); (3) the model coefficients which provide the best fit to the data are estimated (*parameter estimation*); (4) the assumptions are tested and step 2 is applied if the model is found to be inadequate (*model checking*); (5) when the model is selected, estimated and checked, the forecasts are computed based on the model (*forecasting*).

ARIMA model is composed of three parts: the autoregressive (AR), the integrated (I) the moving average (MA). The model is indicated as $ARIMA(p,d,q)$, where AR describes how each observation is a function of the previous p observation (number of time lags), d is the degree of differencing when the time series becomes stationary, and MA states how each observation is a function of the previous q errors [75], [77]. An ARIMA non-seasonal time series can be represented as a combination of past values and past errors [78], [10] as shown in Eq. 3.1:

$$y_t = \theta_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \dots \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \dots - \theta_q \varepsilon_{t-q}, \quad (3.1)$$

where y_t is the actual value and ε_t is the random noise at time t , ϕ_i and θ_j are the coefficients, while p and q are respectively the degrees of autoregressive and moving average parts. Moreover, the order differencing process, denoted as ∇ , is used to transform a non-stationary series to the stationary one [10] and defined in Eq. 3.2:

$$\nabla^d y_t = \nabla^{d-1} y_t - \nabla^{d-1} y_{t-1}. \quad (3.2)$$

Parameter estimation phase in an ARIMA model corresponds to the estimation of p , d , and q values by autocorrelation function (ACF) and partial autocorrelation function (PACF) after the differencing process is applied to the non-stationary series.

Since time series contain a specific periodic component, ARIMA models are enhanced to deal with this seasonality. Therefore, Seasonal Autoregressive Integrated Moving Average Model (SARIMA) model is defined. It enables a deeper understanding for temporal effect corresponding to autocorrelation, trend, and seasonality [79], [78]. The SARIMA process is denoted as the $SARIMA(p,d,q)(P,D,Q)_s$ model, where (P,D,Q) parameters have also the same description with ARIMA, but P and Q represent the AR and MA orders at the seasonal period length of the time series, and D corresponds

to the seasonal differencing. The s parameter corresponds to the periodic trend of the time series. A time series is a SARIMA process with period of S in case d and D are non-negative integers, and differenced series is a stationary autoregressive moving average process [10].

3.2.2 SVR

Support Vector Machine (SVM), a machine learning method, was proposed by Cortes and Vapnik in 1995 [80]. SVM attempts to find the hyperplane that maximizes the margin between two classes and minimizes the absolute error under threshold [81]. The SVR is a regression method that uses the same principles as the SVM. SVR utilizes a high-dimensional feature space obtained with the transformation of original variables and a penalty term added to the error function for penalizing the complexity [82]. To clarify, given $\{(x_i, y_i) | i = 1, 2, \dots, N, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^d\}$, x_i and y_i are the inputs and outputs of the training samples. The following linear regression function in Eq. 3.3 is trained:

$$f(x) = w^T x + b, \quad (3.3)$$

where x is input vector, w is weight vector and b is the offset [52]. The error function is given by Eq. 3.4 [82]:

$$R = \frac{1}{2} \|w\| + c \sum_{i=1}^N |y_i - f(x_i)|. \quad (3.4)$$

The first term in the error function penalizes model complexity. The second term is the ε -insensitive loss function, defined as $|y_i - f(x_i)| = \max\{0, |y_i - f(x_i)| - \varepsilon\}$, where c is penalty factor. It does not penalize errors below ε , allowing smaller deviation than ε to reduce model complexity. For this, the error minimization problem of Eq. 3.5 needs to be solved [82]:

$$\begin{aligned} & \min \frac{1}{2} \|w\| + c \sum_{i=1}^N (\xi_i + \xi_i^*) \\ \text{s.t. } & \begin{cases} f(x_i) - y_i \leq \varepsilon + \xi_i \\ y_i - f(x_i) \leq \varepsilon + \xi_i^* \\ \xi_i \geq 0, \xi_i^* \geq 0, i = 1, 2, \dots, N \end{cases} \end{aligned} \quad (3.5)$$

where ε is the loss function and ξ_i, ξ_i^* represent the slack variables which overcome the infeasible constraints [52]. To solve the optimization problem, the dual problem of the Eq. 3.5 can be represented using Lagrange multipliers α_i and α_i^* , as in Eq. 3.6:

$$f(x) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) x_i^T x + b \quad (3.6)$$

The training vectors producing nonzero Lagrange multipliers are described as *support vectors*, which is the fundamental concept in the SVR approach. To introduce the nonlinearity in the algorithm, This model is extended through the kernel K , as shown in Eq. 3.7:

$$f(x) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) K(x_i^T x) + b \quad (3.7)$$

The penalty factor c in the Eq. 3.5 and kernel function parameter K in the Eq. 3.7 will considerably affect the prediction performance of the SVM [52]. The kernel mapping of SVR is demonstrated in Figure 3.1:

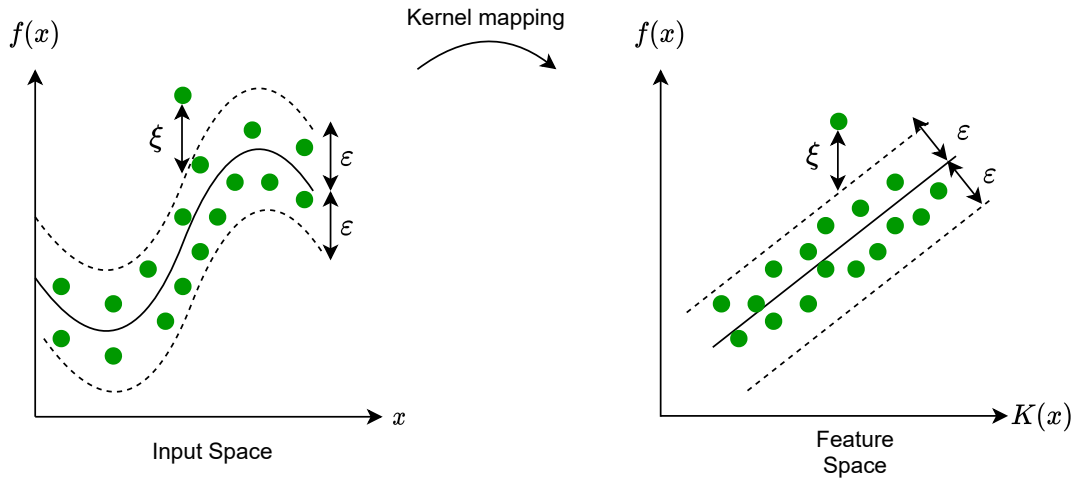


Figure 3.1 : SVR Kernel Mapping

3.2.3 LSTM

LSTM architecture with *memory cell* was introduced by Hochreiter and Schmidhuber in 1997 [83], as a variant of RNNs [53], and further improved by Gers et al. [84] with an additional forget gate. To overcome the gradient disappearance or explosion in RNNs, LSTM contains separate gate control to combine long-term memory and short-term

memory [85]. A standard regression type LSTM neural network comprises multiple layers, including a sequence input layer, an LSTM layer, and a regression output layer.

The basic unit of an LSTM layer is described as a *memory block*, and it contains a single, or multiple memory cells [54]. The architecture of a memory block is represented in Fig. 3.2. The input gate g_t works just as the simple RNN layer. Three control gates in the LSTM cells, forget gate, input gate and output gate are represented by f_t , i_t , o_t , respectively. These gates apply sigmoid activation function producing an output range between 0 and 1 [86]. The decisions of the gates are dependent on the current input x_t and the previous output h_{t-1} [53].

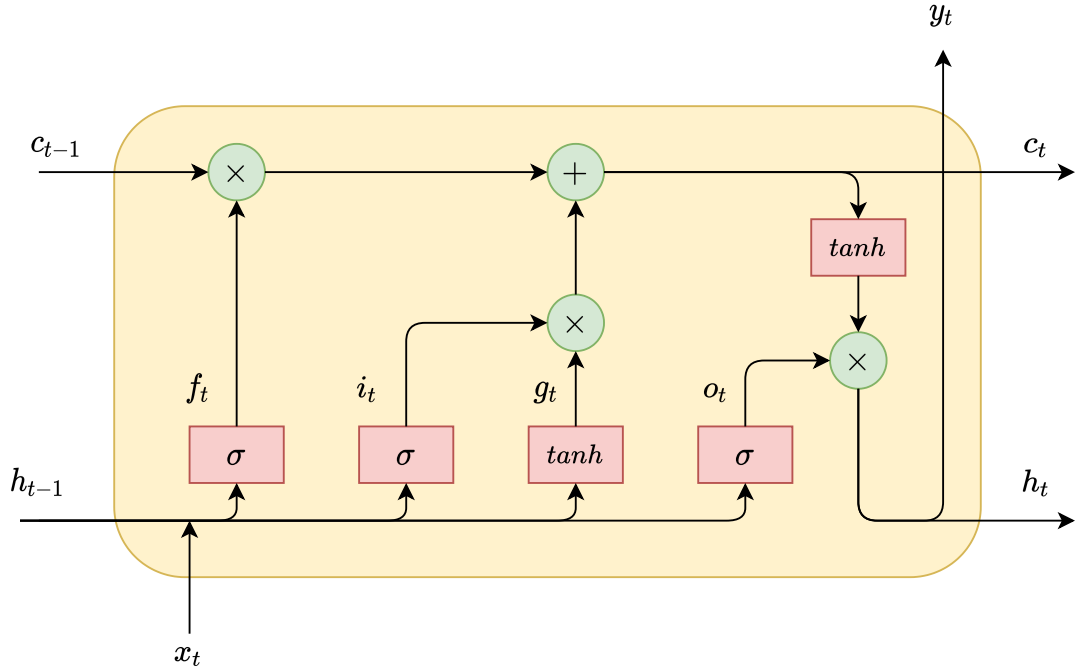


Figure 3.2 : LSTM Memory Block

The forget gate controls what to forget from the previous state, and is represented with Eq. 3.8:

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f), \quad (3.8)$$

where W_f is the weight matrix, x_t is the input at time t , h_{t-1} is the output at time $t - 1$ and b_f is bias term [85]. The input gate regulates what to conserve in the current state and is formed by two functions. The useful information is updated through the *sigmoid* function. Then, a new candidate value vector is created through the *tanh* function [85].

The formulae are as follow:

$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i), \quad (3.9)$$

$$g_t = \tanh(W_g \times [h_{t-1}, x_t] + b_g), \quad (3.10)$$

where W_i and W_g are weight matrices, i_t input gate state, g_t is the new state, b_i and b_g are bias terms, as represented in Eq. 3.9 and 3.10.

Moreover, the cell state c_t is updated through the formulation in Eq. 3.11:

$$c_t = f_t \times c_{t-1} + i_t \times g_t, \quad (3.11)$$

The output gate is also represented by two functions. Firstly, the information to be output is filtered through the *sigmoid* function in the output gate o_t , as in Eq. 3.12. Then, the hidden state h_t is calculated, as in Eq. 3.13. The equations are as followed:

$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o), \quad (3.12)$$

$$h_t = o_t \times \tanh(c_t). \quad (3.13)$$

4. PROPOSED FRAMEWORK

4.1 System Architecture

As the key contribution of this thesis, we proposed a fog-based framework for modeling the smart applications for microgrids, taking into account the requirements mentioned earlier in Section 2.2.3. As a specific application scenario, we focused on the *energy demand forecasting*. We deployed the proposed energy demand forecasting application with its hardware and software components and performed a set of experiments to show the effectiveness of the fog-based framework.

As first step, we depicted a reference architecture for the proposed framework. The 3-tier grid architecture comprises of **edge layer**, **fog layer** and **cloud layer**, as shown in Figure 4.1.

The edge layer contains the smart meters and sensors that are the source of generated data at the edge of the network. The edge layer refers to a group of NANs where each NAN is originated from multiple HANs. Electrical appliances, sensors, and smart meters are located in a HAN. Thus the household energy consumption data is produced in each HAN. The weather or social factors are also taken into consideration while modeling consumption behavior. The weather data, such as air temperature, humidity, and rainfall, are also produced through various sensors in the neighborhood level at the edge layer. The generated data in both HANs and neighborhood level is gathered and transmitted to the next step of the information flow, the fog layer, via a NAN gateway located between physical level devices and the fog layer.

The fog layer is an intermediate layer between devices in the edge and information centers in the cloud. Low-latency, distributed computing resources, and storage space are required to run intelligent grid applications for the demand management of neighborhoods where distributed microgrids supply energy. A machine that provides the computational power and data storage in the fog layer is called *fog node* in our framework. Each neighborhood has its fog-based information center, namely *fog*

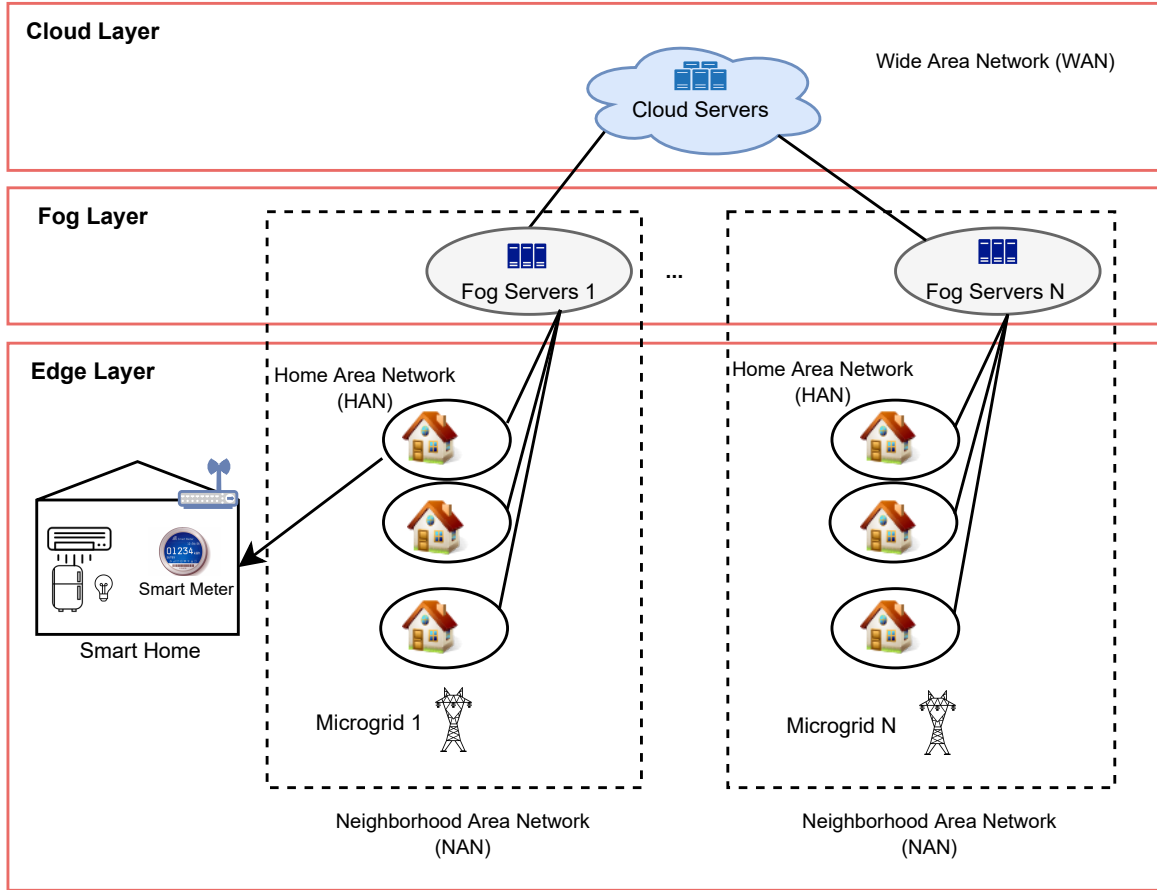


Figure 4.1 : Fog-based Reference Architecture

cluster formed by fog nodes. The fog clusters are located at NANs to address the decentralized information management of microgrids. The devices in the edge layer should be able to communicate with the nodes in the fog cluster. Moreover, the fog nodes within the cluster are supposed to talk among themselves. Consequently, a cluster topology, i.e., count of nodes, manager and worker nodes, needs to be described. The architecture design and deployment of grid applications to be running in a fog cluster should also be addressed.

The cloud layer is the topmost layer in the information flow from smart meters towards global information management systems. The fog clusters in NANs are connected to cloud layer resources in WAN. The grid application running in the fog nodes processes the input data and produces meaningful information for consumers or utility suppliers. This information is transferred to the cloud layer for decision-making. The decision-making systems here facilitate the management of energy generation and transmission grid subsystems settled in the WAN. If the fog nodes' resources are

insufficient to execute a task, the task is propagated to cloud servers. In addition, the cloud layer is employed for long-term data storage and data analysis scenarios.

Based on this reference architecture, we proposed a fog-based framework with real Raspberry Pi devices running the use case scenario in Figure 4.2. The framework also comprises **edge layer**, **fog layer** and **cloud layer**. The developers or researchers can use the proposed framework to develop, test, or evaluate a smart grid application. We call these target users *operator* to standardize the role.

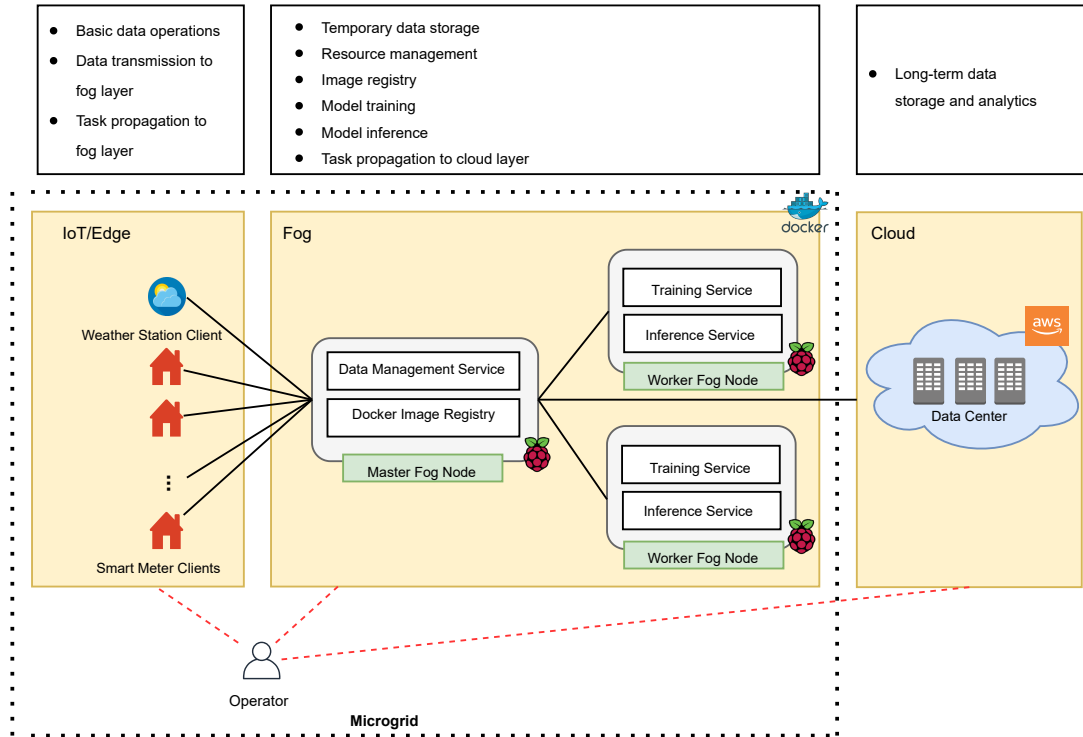


Figure 4.2 : System Architecture

4.1.1 Edge Layer

To deploy an energy demand forecasting application, two elements that need to be modeled in the edge layer:

- Devices, i.e., smart meters and weather stations
- Networking and communication between edge and fog layer

4.1.1.1 Clients

Since application testing and validation are the main focus of this study, the physical setup of smart meters and sensors on the edge layer is not included in our work.

Instead, a smart meter is simulated with the smart meter client, and a weather station is mocked up with the weather station client. Pre-collected load and weather data sets can be used for the experiments to be conducted in our framework. Several reasons have brought us to decide to simulate the sensors through clients publishing the presented data sets:

1. Physical setup of smart meters is a challenging task.
2. To reach the consumers with various load profiles requires extra work out of the scope of this study.
3. Smart grids are large systems requiring long-range communication protocols with specific networking devices and deeper domain knowledge.

Furthermore, the client implementation allows the flexibility and rapid development. The *periodicity* of transmitted data can be configured on demand. Moreover, clients can perform basic data operations such as cleaning redundant data, or aggregating multiple sets. In this thesis, we call these clients as *edge clients* in general.

4.1.1.2 Networking and Communication

The simulation of a real grid network formed by HAN, NAN, and WAN is one of the challenges of framework design. Since we are restricted to the same local area network when conducting the study, we simulated a group of HANs connected to a NAN in the referred LAN. Based on the experimental scenario, one or multiple neighborhoods can be fulfilled through the clients to be deployed in the edge layer. We employed a personal computer (PC) as an edge node to deploy smart meters clients and weather station client.

Hyper Text Transfer Protocol (HTTP) and Message Queuing Telemetry Transfer (MQTT) are the communication protocols widely used in IoT applications. HTTP is request/response based communication model, whereas MQTT is publish/subscribe based model [87]. In MQTT communication, the client-side does not need to probe the server for data transfer in each attempt and it sends the data without asking. Since the MQTT keeps the established connections alive, the TCP communication is reduced for scenarios where tiny packets are sent frequently. In our framework, we build a data transmission flow where smart meter data is sent in batches at specific periods instead

of in a large number of tiny packets. The HTTP protocol is advantageous because it keeps the connection alive for a shorter time compared to MQTT. Moreover, HTTP is simpler than MQTT since it is universal and not dependent on any broker specification. Therefore, we built a manageable HTTP system in which edge clients POST file data to a remote server using the Representational State Transfer (REST) architectural style.

4.1.2 Fog Layer

Based on the discussions in the introduction part of Section 4.1, there are several functional requirements to deal in fog layer:

- Cluster topology
- Application architecture
- Application deployment

4.1.2.1 Cluster Topology

A fog cluster is a number of fog nodes forming a mesh. We organized the fog cluster in master-worker topology. To establish a fog cluster in the proposed framework, we used low-cost and relatively high-performance Raspberry Pi devices due to its several advantages mentioned in Section 3.1.1 and especially its feasibility for machine learning. We designed 3-node of Raspberry Pi, consisting of 1 master node and 2 worker nodes. The master and worker nodes are assigned to execute specific services of the application. These services are explained in Section 4.2.1. 2 worker nodes to demonstrate the effect of increased computing capacity in the experiments. As mentioned in Section 4.1.1.2, hardware devices in the framework are placed in a LAN. Therefore, fog nodes communicate with the edge node through in the same LAN.

4.1.2.2 Application Architecture

We proposed *microservice* architecture for the applications to be deployed in a fog-based framework. In the microservice approach, an application is developed as decoupled services that run independently but can communicate with each other [35]. This architecture facilitates the operating grid applications in the proposed framework. Besides, the microservices can run distributed across fog nodes. Consequently,

the microservice architecture ensures flexibility and maintainability in the proposed framework.

4.1.2.3 Application Deployment

The architecture design with a couple of microservices poses the issue of resource utilization. The lightweight resource usage is the goal of an efficient microservice operation. Instead of a device used itself for each microservice, the virtualization technologies enable multiple applications to be running in the same device efficiently. As mentioned in Section 3.1.2, the Docker container is an outstanding technology for IoT devices with limited resources to be used with maximum efficiency. Therefore, we used Docker containers for service deployments on top of the fog cluster with Raspberry Pi devices. To orchestrate the containers deployed across cluster machines, we used Docker Swarm¹ which is a container orchestration tool. The *service* term in Swarm implies a group of replica containers that executes a program in a distributed environment. Swarm allows configuring service properties, i.e., number of replica containers, CPU and memory limits, or exposed port. The *overlay network* driver creates a distributed network among Docker daemon hosts, allowing swarm service containers connected to it and communicate with each other. This Docker-based infrastructure orchestrated by Swarm helped us satisfy our design criteria as below:

- **High-availability:** Multiple Docker containers should be spread over 3-node Raspberry Pi cluster.
- **Scalability:** The number of containers to run in the cluster should be configurable, and this number can be scaled up or down based on the resource demand.
- **Reusability:** The application images should be built and to kept them in a registry. Thus, the images can be utilized many times to run containers in any platform with compatible system architecture.

4.1.3 Cloud Layer

We also aim at benchmarking key performance metrics such as network latency and execution time when the learning models are trained either in the fog or in the cloud

¹<https://docs.docker.com/engine/swarm/>

environments. Therefore, we employed an AWS EC2 cloud server to deploy the services for benchmark scenarios. We call the cloud server as *cloud node*.

4.1.4 Operating the Framework

To recall, the target users of the proposed framework are called as *operators*. The developers or researchers can become operators of the proposed framework to develop, test, or evaluate a smart grid application. The authentication in Raspberry Pi nodes is done with username and password securely. The operators are authorized to access Docker engine API. They follow the steps below to operate the framework:

1. **Initializing fog cluster:** The operator initializes the swarm cluster with the following command to connect the nodes:

```
docker swarm init --advertise-addr <MNGR_IP_ADDR>
```

The node targeted by this command becomes a *manager* in the newly created single-node swarm. The other two machines join as worker nodes based on a token generated on the manager node. For this, the following command is run on the worker nodes:

```
docker swarm join --token <TOKEN> <MNGR_IP_ADDR>
```

The nodes and their roles are listed to validate that the cluster is initialized properly:

```
docker node ls
```

2. **Managing image registry:** The operator hosts a registry within the manager node using the open source Docker Registry.

When pushing an image to the Docker registry, image tag is represented in the following format: `HOSTNAME:<PORT>/<IMAGE_NAME>`. The first part of the image tag is a hostname and port is interpreted as the location of a registry by Docker. For example, this an image tag: `manager1:5000/ml_base_image` where the registry address is `manager1:5000`, the `manager1` is the hostname of manager node, `5000` is the registry port, and `ml_base_image` is the image name.

3. **Building service images:** A Docker image is built through *Dockerfile*. A Dockerfile is a document that contains all the commands to assemble an image. The

operator can build the images of multiple applications through a **docker-compose** stack. Docker-compose refers to a service deployment model that bundles multiple service definitions in a YAML file. The docker image for each service is built and pushed into the registry. The Source Code 4.1 shows the Dockerfile of a service. The base image with installed packages that are required for service execution is at Line 1. Line 4-6 includes the commands to copy the source files from the host machine into the container file system. At the Line 9, Python script `main.py` is called to execute the program.

Source Code 4.1: Dockerfile

```
1 FROM ml_base:v1
2
3 # Copy files from the root to the work directory
4 COPY . /src/
5 RUN ls -la /src/*
6 WORKDIR /src/
7
8 # Trigger Python script
9 CMD ["python3", "-u", "/src/app/main.py"]
```

A simple `docker-compose.yml` file comprises the list of service and volume definitions is shown in Source Code 4.2. Each service definition allows defining the attributes of the related service. The attribute of *build* is the folder location of Dockerfile, while *image* refers to the tag of the output image after the building stage.

Source Code 4.2: Docker Compose File for Building Service Images

```
1 version: '3.6'
2 services:
3   training_service:
4     build: training_service/.
5     image: manager1:5000/training_service:v1
6     ...
7   inference_service:
8     build: inference_service/.
9     image: manager1:5000/inference_service:v1
10    ...
11  data_management_service:
12    build: data_management_service/.
13    image: manager1:5000/data_management_service:v1
14    ...
```

The operator uses following command to build the images via the `docker-compose.yml` file:

```
docker-compose build
```

Then, the built images are pushed into the image registry `manager1:5000` in `manager` node:

```
docker-compose push
```

Consequently, the images are available in the registry to be pulled when creating containerized services.

4. **Deploying swarm services:** Since the framework is to be deployed in a 3-node cluster, the services are deployed across the nodes in swarm mode. The deployment of services in swarm mode requires a stack description in a compose file format. The option of *build* is ignored when deploying a stack in swarm mode. Therefore, the images should be built before deploying a stack in swarm mode, as described in the previous step. The complete version of `docker-compose.yml` for stack deploy is demonstrated in Source Code 4.3:

Source Code 4.3: Docker Compose File for Deploying Swarm Stack

```
1  services:
2      training_service:
3          image: manager1:5000/training_service:v1
4          ports:
5              - 8070:8070
6          volumes:
7              - models:/src/models
8              - data:/src/data
9          networks:
10             - web
11         deploy:
12             replicas: 3
13             resources:
14                 limits:
15                     cpus: '2'
16                     memory: 512M
17                 reservations:
18                     cpus: '1'
19                     memory: 128M
20         logging:
21             driver: "json-file"
22             options:
23                 max-size: 1m
24     inference_service:
```

```

25     image: manager1:5000/inference_service:v1
26     volumes:
27         - models:/src/models
28     ports:
29         - 8000:8000
30     networks:
31         - web
32     deploy:
33         replicas: 3
34         resources:
35             limits:
36                 cpus: '1'
37                 memory: 512M
38             reservations:
39                 cpus: '0.5'
40                 memory: 128M
41     logging:
42         driver: "json-file"
43         options:
44             max-size: 1m
45     data_management_service:
46         image: manager1:5000/data_management_service:v1
47         ports:
48             - 8090:8090
49         networks:
50             - web
51         deploy:
52             placement:
53                 constraints: [node.role == manager]
54             resources:
55                 limits:
56                     cpus: '1'
57                     memory: 512M
58                 reservations:
59                     cpus: '0.5'
60                     memory: 128M
61         logging:
62             driver: "json-file"
63             options:
64                 max-size: 1m
65
66     volumes:
67         data:
68         models:
69
70     networks:
71         web:

```

The attribute of *deploy* defines service configuration related to the deployment. There are also several sub-options of *deploy* attribute. We used *replicas* to specify the number of containers to be running at a given time. Besides, we defined *resources* to configure resource constraints, e.g., CPU and memory. It is also possible to specify the placement of constraints and preferences. In our framework, we scheduled *data management service* as so running in *manager* node. Furthermore, the *volume* option is used to mount a volume into the container, and the top-level *volumes* section creates the volumes. *port* represents the port exposed. The attribute of *networks* states the overlay network that connects swarm services when they are distributed across multiple nodes. The top-level *networks* section creates a named overlay network across multiple nodes.

To create the stack, the following command is executed:

```
docker stack deploy --compose-file docker-compose.yml
energy_demand_forecasting_framework
```

The last argument is the name of the stack. Each created network, volume, and service name starts with the stack name.

5. **Deploying edge clients:** The edge clients are the web applications used to simulate the functions of data generating entities in grid networks. Real data sets are used as data sources for these clients, so grid entities, e.g., smart meters or weather stations, flexibly complement the test scenarios. The clients are capable of: 1) splitting the data set into pieces in a level of granularity 2) transmitting data to the fog layer.

These functionalities are developed in Python in our framework, and the applications are basically files with .py extensions. These files are sourced in a Git repository, and need to be cloned into the machine used as edge node at first:

```
git clone <GIT_REPOSITORY_URL>/clients.git
```

After cloning the project, the Python client applications are executed as follow:

```
python smart_meter_client.py
python weather_station_client.py
```

4.2 Application Design

In this section, we introduced the design phases of the proposed energy demand forecasting application. We explained the services in charge of delivering specific functionalities of the application. We also presented the details of data processing and

the ensemble learning stages for the proposed energy demand forecasting application. Besides, we discussed the implemented features, and illustrated the communication between the components with sequence diagrams.

4.2.1 Services

The energy demand forecasting application consists of 3 services, namely *data management service*, *training service* and *inference service*.

4.2.1.1 Data Management Service

The power consumption and air temperature data published from the edge layer arrive at the *data management service*. The data preprocessing steps are applied here, and preprocessed data is stored in the *master* fog node. The aggregated level of household energy consumption data is also propagated to the cloud for long-term analysis. The *training* and *inference* services use the household energy consumption and temperature data as the input for forecasting. This service also provides the APIs that forward requests to *training* and *inference* services in the fog layer. Due to experimental purposes, the inference and training tasks can also be propagated to the cloud servers. Therefore, it enables communication with other services in fog or cloud layer.

Role-based access is not focused in this thesis, but in theory, the separate roles can be defined for two following user groups of grid applications. They can access the framework with a token-based authentication mechanism:

1. Utility suppliers: The energy consumption prediction data helps utility suppliers to enable demand management actions such as dynamic pricing.
2. Consumers: The predicted energy data enables the consumers to analyze their energy consumption behavior.

4.2.1.2 Training Service

This service provides functions to train a group of learning models with historical data. The motivation to implement this service has emerged from the ensemble learning approach proposed in Section 4.2.2. The ensemble learning model combines the prediction results of involved machine learning or deep learning models. Therefore, with the object-oriented programming approach, separate classes have

been implemented for each sub model characteristic. Even though the function implementation details differ based on models, the sub model classes contain abstract methods that apply the same scheme. The best model parameters of sub models are searched in the training phase. The best models are saved to the disk to be loaded during predictions later. The service runs in distributed fog nodes to leverage resource availability for complex learning tasks. The service can also be deployed to the cloud in some scenarios, such as comparing performance key metrics in cloud and fog.

4.2.1.3 Inference Service

The inference service is designed to execute the forecasting model algorithm to predict live data points. The evaluated best models are saved into the disks of fog nodes by the training service. These saved models are used in the inference phase. Firstly, the live hourly data is preprocessed in the data management service. Then, the prediction request is done to the inference service for this data point. The inference service loads the sub models and predicts the energy consumption value of the next hour using the proposed ensemble learning approach in Section 4.2.2. The inference service runs on distributed fog nodes or is deployed to the cloud for experimental scenarios, similar to the training service.

4.2.2 Energy Demand Forecasting

4.2.2.1 Data Preprocessing

The raw data transmitted into the data management service in Figure 4.2 is *power* measurements in Watt. For hourly energy demand forecasting, household power consumption data is firstly transformed into hourly energy that is described as the integral of power over time. The data management module produces hourly energy consumption values from the power measurements recorded every N seconds, where N ranges from 0 to 60 and can vary. For this purpose, the power measurements in Watt are aggregated into *energy* measurements in Watt-hour (Wh) with the weighted average approach [6]. The weights are represented by the time elapsed between the irregular measurements within an hour.

Besides, if the energy demand forecasting target is all houses in a microgrid, hourly energy consumption data incoming from each house should be aggregated. The air

temperature and temporal features (i.e. month, day of the week, and hour) can be correlated with load patterns and are expected to contribute to the model's success. These features are fused into energy consumption data consequently.

4.2.2.2 Ensemble Model for Short-term Load Forecasting

For the ensemble learning module, we proposed an ensemble STLTF model that incorporates SVR as a shallow model and LSTM as a deep learning model. Since energy consumption is dependent on many factors such as social or climatic factors, we decided to use the aforementioned learning models that capture the non-linearities and are popular in STLTF. SVR is not a probabilistic model but it splits the observations into separate classes by the longest distance. The model parameters of SVR are also easy to tune [6]. On the other hand, LSTM is an RNN based deep learning model with a recurrent feedback loop and the memory cell. Therefore, it is effective at capturing temporal dynamic behavior [12].

STLTF is concerned with daily or weekly energy consumption prediction. The data decomposition reveals the seasonal patterns (daily or weekly) underlying the energy consumption data set. The auto-correlation function (ACF) coefficients of the data set also give the lag orders of highly correlated values with the current load value. For example, if load data is presented in 1-hour granularity, there can be correlations in time lag 24 which implies daily seasonality. Similarly, in the load data set with a daily resolution, the correlations in time lag 7 address the weekly seasonality. Thus, the correlated load sequences are used to build learning models for daily or weekly energy prediction. These sub models integrate into an ensemble model to obtain better prediction results [14].

The ensemble model setup for STLTF scenario is shown in Figure 4.3. The appropriate data frames should be prepared for each sub model training. D_t refers to the data point at time t , which comprises load data, temperature data, or temporal data. The lag values 1 to T derive from auto-correlation coefficients. k value indicates how many steps a data point in the *continuous* historical data is behind the current data point, while K shows the order of a data point in the historical data recorded at the *intervals of T*.

Thus, we followed two approaches for the prediction of the next T -step window. In the first approach, a single model uses the continuous sequence of data S to predict

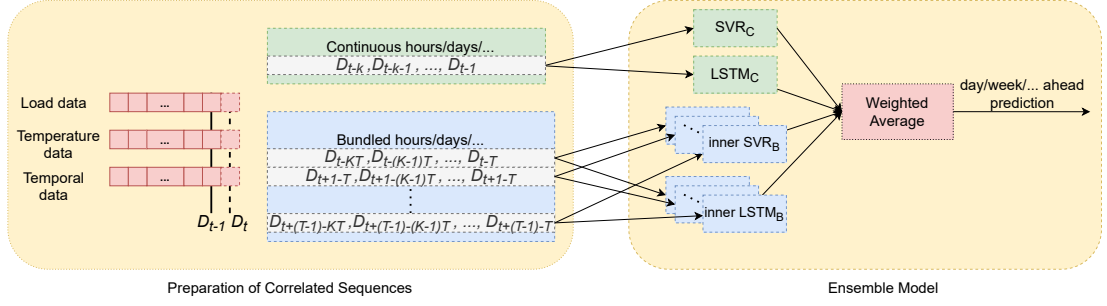


Figure 4.3 : The Proposed Model Overview

the T steps ahead. On the other hand, the separate models can be trained with data corresponding to the target sequence S_i where i ranges from 0 to $T-1$ (e.g. hour in a day, or day in a week) in the second approach. Therefore, each of the next T steps can be predicted through separate models. Both approaches differ in their learning abilities and complement each other to form a more powerful learning model. From this point of view, we constructed the *sub models* that we call $LSTM_C$, SVR_C , $LSTM_B$ and SVR_B to be ensembled. The sub models $LSTM_C$ and SVR_C consist of single models and learn from *continuous* historical data points. Nevertheless, the sub models $LSTM_B$ and SVR_B represent the model groups formed by the individual models, namely *inner models*. Each inner model learns from *bundled* historical data points recorded at the time from 0 to $T-1$.

Before training the $LSTM_C$ model and $LSTM_B$ inner models, the time-series data is transformed to an appropriate input-output format for supervised learning. As a result, the input consists of the energy consumption of the last hour, temporal data, and temperature data, while the output is the energy consumption of the next hour. We used the same input and output for all models. As the next step, the data is normalized and reshaped into the 3D tensor for the LSTM cells. Similarly, the data is standardized so that its distribution would have 0 mean and standard deviation of 1, in advance of SVR_C model and SVR_B models' training.

In the hyperparameter tuning stage, the grid search is applied to find out the optimal epoch, nodes, and batch size required for the model fitting for each LSTM network. To find the best SVR models, the kernel functions are evaluated using grid search. $LSTM_C$, SVR_C , $LSTM_B$ and SVR_B are ranked based on the root mean square error (RMSE) performance measure in the training phase. Based on their rankings, the model predictions are aggregated in a weighted average manner to discover the

possibility of an ensemble STLF model. The more accurate model has the higher weight in the final prediction results.

4.2.3 Implemented Features

In this section, we explained the implemented features for *ensemble learning pipeline* to execute in the fog-based framework. We also demonstrated how the framework components communicate with each other to execute each task in the workflow, using sequence diagrams. The ensemble learning pipeline and related features that are implemented in the energy demand forecasting application are shown in Figure 4.4.

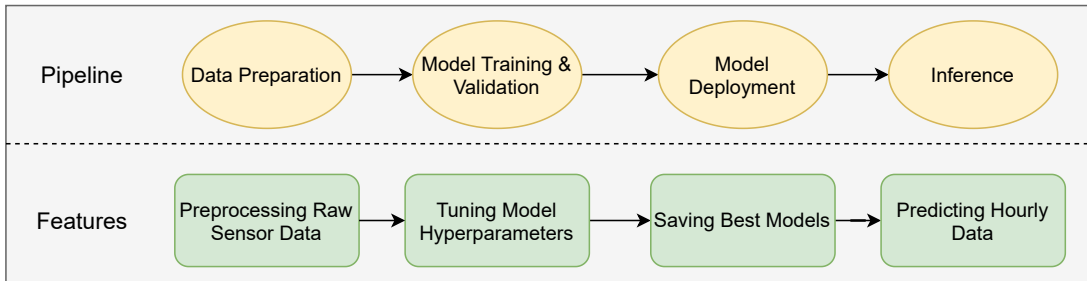


Figure 4.4 : Ensemble Learning Pipeline

4.2.3.1 Preprocessing Raw Sensor Data

To recall, the operator is the person who uses the proposed framework to develop, test, or evaluate a smart grid application. The operator controls the experiments to be conducted in the framework and should upload the experimental data set into the edge node as the first step. The edge clients are the application simulation of real sensing devices that transmit the raw data. The clients run in two modes: 1) training mode and 2) inference mode. The client mode determines how the experimental data is treated when transmitting it to the fog layer. The client transmits the data in *batch mode* when running in training mode. On the other hand, it transmits the data in *chunks* to mimic the live data points if it executes in inference mode.

Figure 4.5 is the sequence diagram shows the interactions among the operator, edge and fog layer components.

The first interaction is between the operator and clients in the edge node. The operator uploads the raw data set into the edge node (1), then runs the edge clients in *training mode* for the training stage that comes first (2). Before the data is uploaded into the

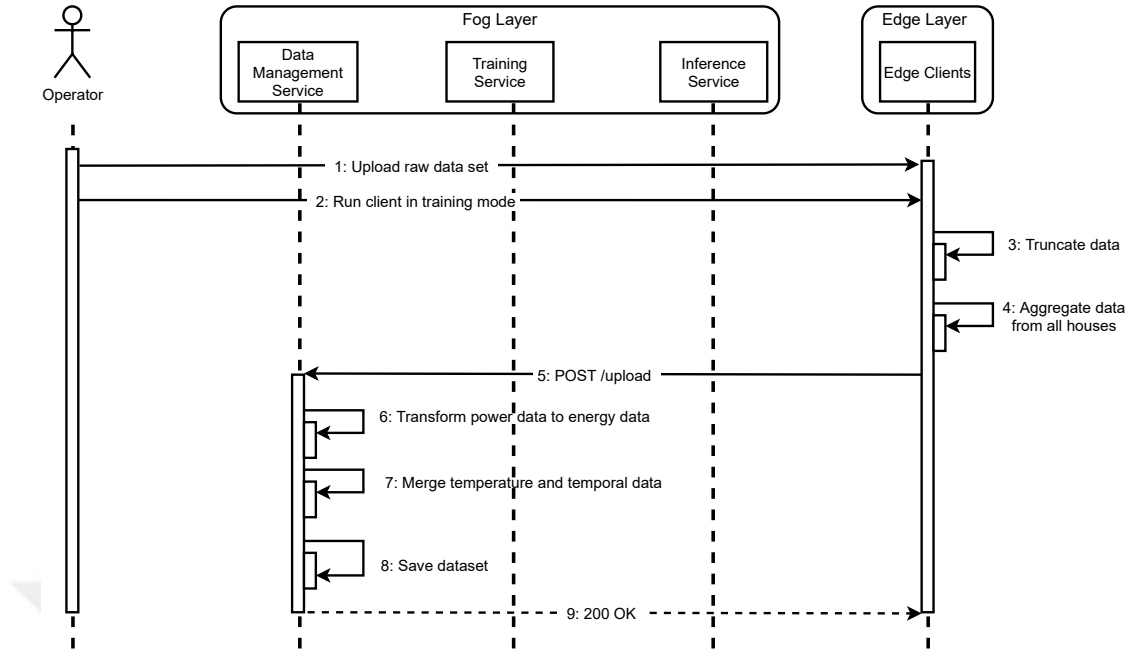


Figure 4.5 : Sequence Diagram for Preprocessing Raw Sensor Data

fog layer, the first preprocessing step is done at the edge side, the redundant attributes of data are truncated (3). For example, if the aggregated level power consumption is concerned only, the appliance level data is unnecessary. Besides, the data sets obtained from multiple houses are aggregated into one data set, since we focused the aggregated load in the microgrid level (4). The second interaction is between the edge clients and the data management service in the fog layer. The data management service has an API that allows the data upload into the master fog node. The edge clients upload the raw data in .csv format in *batch mode* making a POST request to */upload* endpoint (5). The preprocessing steps described in Section 4.2.2.1 are applied through the data management service (6,7). Then the preprocessed data is saved into the disk (8), and the API returns HTTP code 200 to the edge clients (9).

Finally, the data set appears as in Table 4.1.

4.2.3.2 Tuning Model Hyperparameters

After the preprocessing step is completed, the operator is able to execute the training task. The main goal of the operator is to find the best learning models for the proposed energy demand forecasting application and compare them against an ensemble model. The ensemble learning approach we proposed in Section 4.2.2.2 utilizes SVR and

Table 4.1 : Data set

Time	Weighted Power	Temperature	Month	Day	Hour
2015-01-07 10:00:00	353.433	6.349	1	3	10
2015-01-07 11:00:00	342.900	6.622	1	3	11
2015-01-07 12:00:00	367.433	7.220	1	3	12
2015-01-07 13:00:00	512.466	7.507	1	3	13
2015-01-07 14:00:00	420.200	7.934	1	3	14
...	...	...	...	...	...

LSTM learning methods. The operator can find model hyperparameters that are as good as possible, using the capabilities of the training service.

The sequence diagram of the training task is depicted in Figure 4.6. The first interaction occurs between the operator and the data management service in the fog layer. The data management service has an API that allows the training request. The operator sends an initial POST request to */train* endpoint (1). Firstly, the data bundling is performed to bundle correlated sequences indicated in Figure 4.3 (2). The continuous data is used to train the LSTM_C and SVR_C models. On the other hand, LSTM_B and SVR_B inner models are trained with the bundled data.

Then, the second interaction is between the data management service and training service. The training service has separate APIs for accessing the methods associated with each model, namely */train_svr* and */train_lstm*. Data management service loops the set of bundled data sequences and makes POST requests by appending the .csv file containing the data sequence to the request body (3.1, 3.2). The data within the request arrives at the training service, and the time-series data is framed for supervised learning (4.1, 4.2). Then, the data is split into the partitions of train, validation, and test (5.1, 5.2). The pre-defined model hyperparameters are tuned in a grid search, where the target data sequence is trained repeatedly with each parameter set (6.1, 6.2).

After fitting the each model on train data set for both the SVR and LSTM, the prediction was performed on the validation data set. The parameter set that gives the best RMSE score, i.e., lowest prediction error, is selected as the ideal parameter set for the target data. The model parameters are saved into the disk in each request (7.1, 7.2). A model is trained with the best set of parameters, and the validation data is predicted using this model (8.1, 8.2). All training requests to */train_svr*, and */train_lstm* return

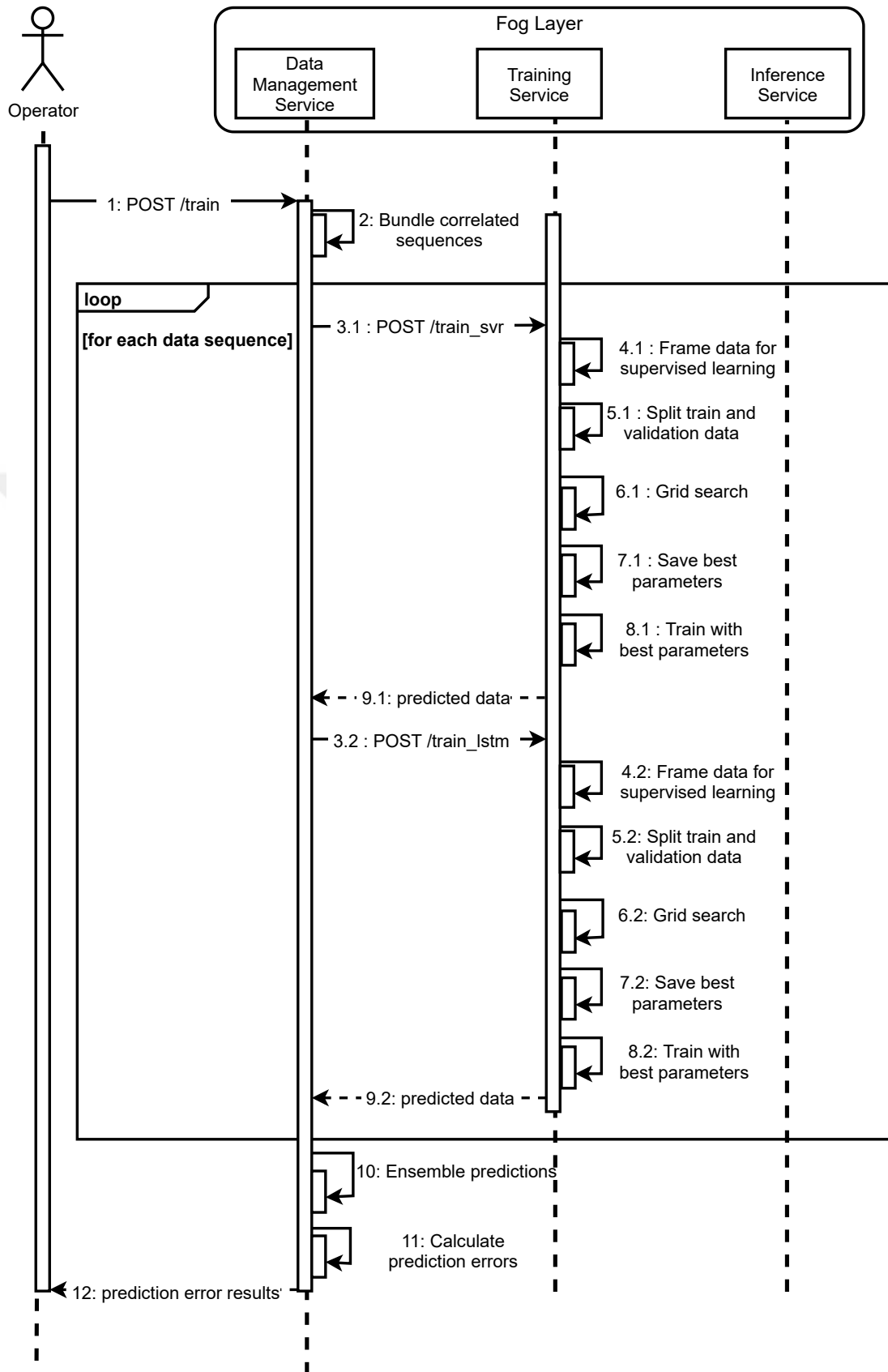


Figure 4.6 : Sequence Diagram for Tuning Model Hyperparameters

the actual and prediction values to the data management service after completing their lifecycle (9.1, 9.2).

The mechanism with multiple API calls relies on parallel tasks launched using the `concurrent.futures` library in Python. The asynchronous execution is performed with threads, using `ThreadPoolExecutor`. For readability, the sequence diagram that shows the POST requests to `/train_svr` and `/train_lstm` endpoints sequentially. In fact, these requests are processed concurrently by the training service.

The predicted values that come from $LSTM_C$ and SVR_C are already continuous. However, the prediction results yielded by the inner models of $LSTM_B$ and SVR_B are bundled data sequences. These results return to the data management service asynchronously are merged to form a continuous sequence of data. After formatting data, the prediction results of each sub models, i.e., $LSTM_C$, SVR_C , $LSTM_B$ and SVR_B are combined to form an ensemble model (10). The RMSE errors that refer to *prediction errors* are calculated for each sub model and also the ensemble model (11). Consequently, the data management service returns the prediction errors of each sub model to the operator (12). Thus, the operator can compare the model performances through these errors and see how the ensemble model improves the prediction ability.

4.2.3.3 Saving Best Models

The hyperparameter tuning provides the best parameters for the models used in energy forecasting. The train data has been used to fit the models, while validation data has been used to evaluate the model fit. If the prediction RMSE results on the validation data are acceptable, the operator is also expected to evaluate a *final* model fit. For this, the final models should be trained with the combined *train* and *validation* data. Besides, the final models should be evaluated with the unseen *test* data.

The training service provides functionality for fitting the final models and saving these models for future inference tasks. The interactions between the components are demonstrated in Figure 4.7. The flow shows the similarity with the interactions depicted in Figure 4.6, and asynchronous execution is performed with threads like the previous stage. In this feature, the training task is performed with several differences.

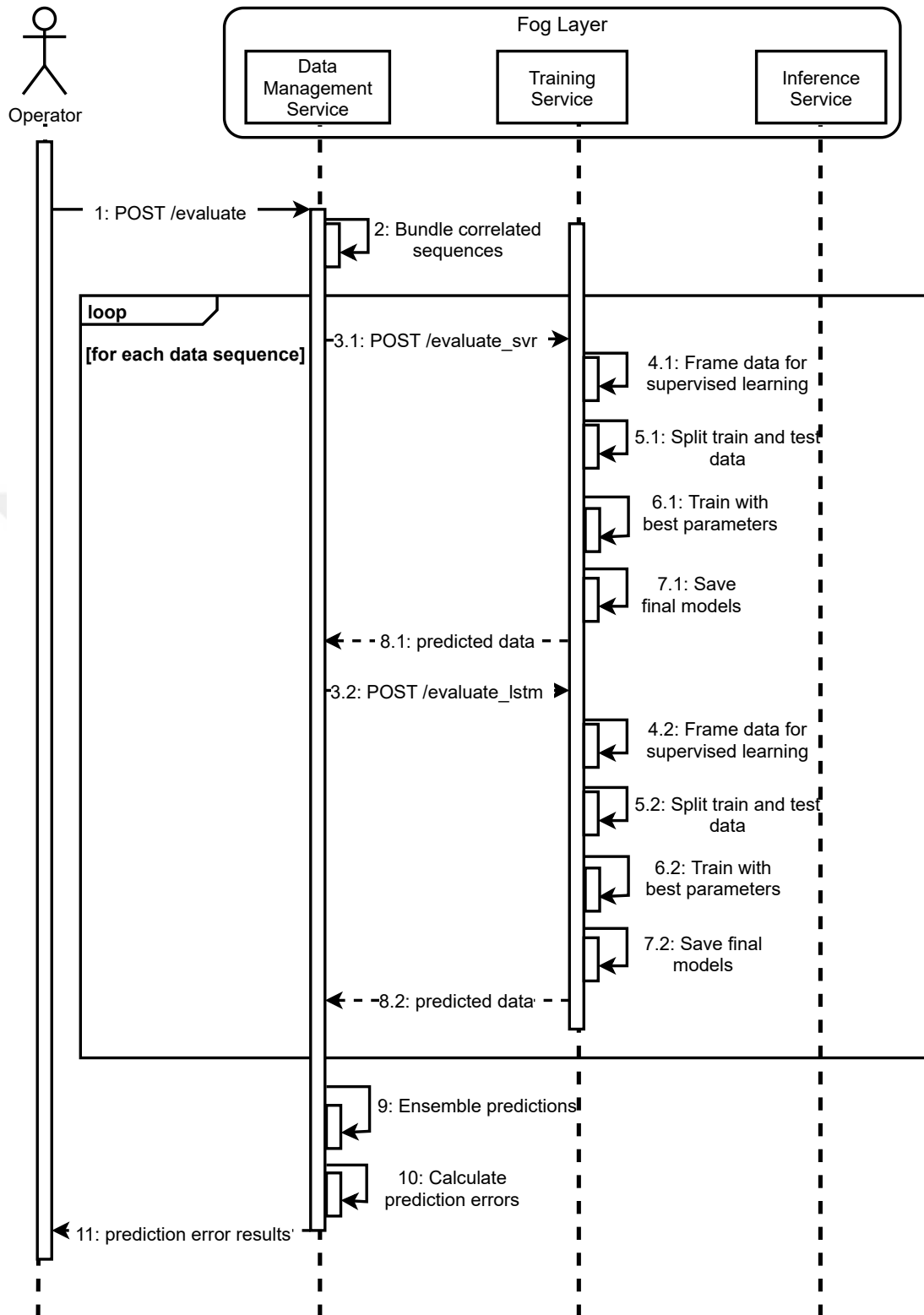


Figure 4.7 : Sequence Diagram for Saving Best Models

The operator interacts with the data management service by sending an initial POST request to */evaluate* endpoint (1). Data management service loops the set of bundled data sequences (2), and makes POST requests to the */evaluate_svr* and */evaluate_lstm*

endpoints with correlated data sequences that are included in the request body (3.1, 3.2). The training service frames the time-series data (4.1, 4.2) and splits the framed data set into train and test data (5.1, 5.2). This time, the train data set consists of the combination of train and validation data used in the previous hyperparameter tuning stage. The last partition of the data set is used as test data.

The training service trains final SVR and LSTM models on bundled and continuous data sequences using the best parameters that have already been recorded in the previous stage (6.1, 6.2). Afterward, the models are saved into the disk for future inferences (7.1, 7.2). The predictions are performed on the test data, and the training service returns the prediction results to the data management service (8.1, 8.2). Here, the ensemble model results are calculated based on the prediction results of each sub models, i.e., LSTM_C, SVR_C, LSTM_B and SVR_B (9). Lastly, the prediction performances of sub models LSTM_C, SVR_C, LSTM_B, SVR_B and ensemble model are evaluated on the test data (10), and resulting RMSE values are returned to the operator (11).

4.2.3.4 Predicting Hourly Data

The data preprocessing, model training, and evaluation have been executed before this stage. The final models are used to predict live data points, and this is called *inference*. From this aspect, inference is the last stage of the model deployment pipeline in our framework.

The sub models LSTM_C, SVR_C, LSTM_B, SVR_B are built to predict the total energy consumption of houses in the neighborhood 1-hour ahead. Therefore, the final models are used to accomplish hourly prediction.

The interactions among the components are shown in Figure 4.8.

Firstly, the operator runs the client in *inference mode* (1). The clients running in this mode are capable of dividing the data set into parts according to the specified periodicity, e.g., minutely, 15-minutely, or hourly, and converted into files in .csv format. In our scenario, the client divides data sets from each house into hourly sets to mimic the hourly live data points during the inference phase (2).

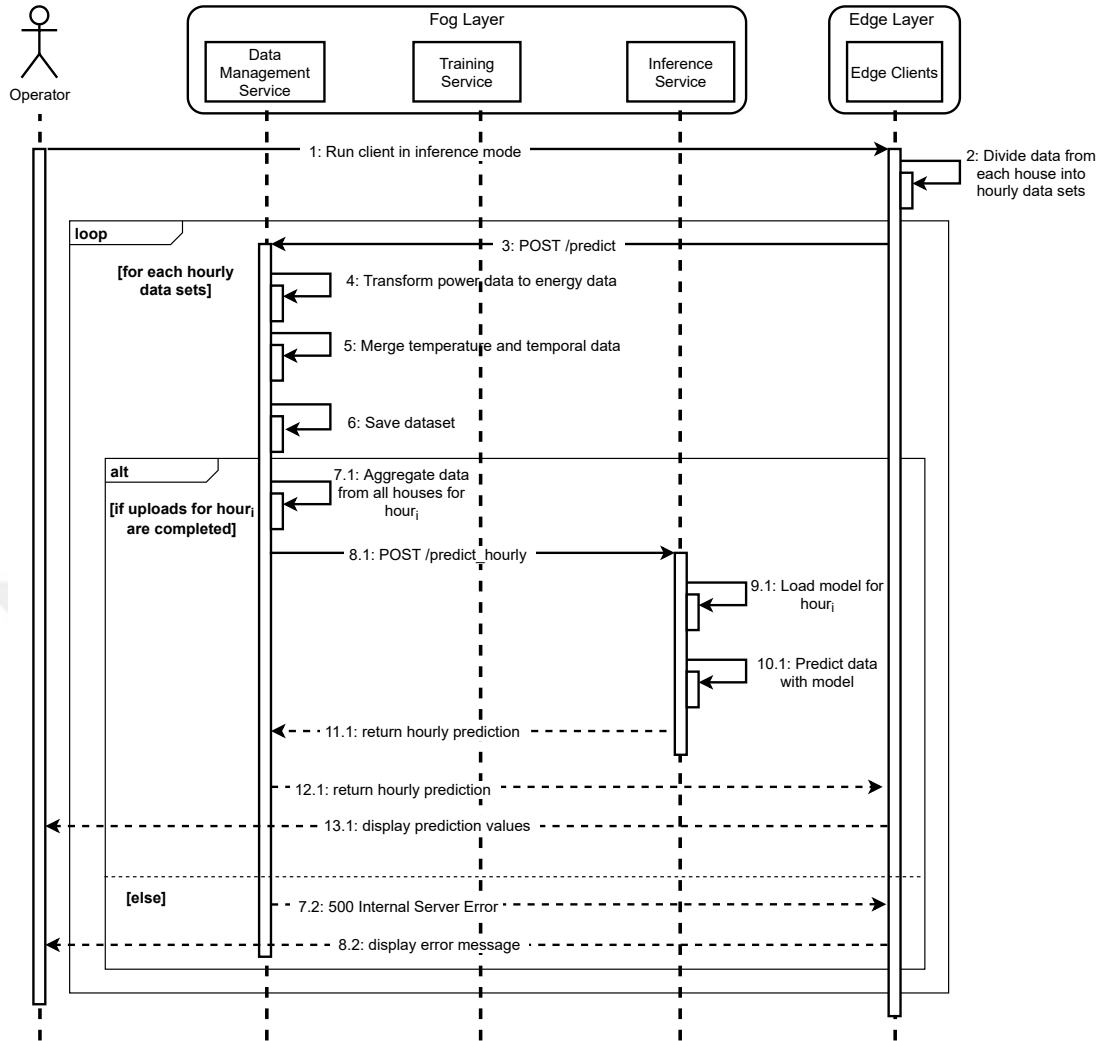


Figure 4.8 : Sequence Diagram for Predicting Hourly Data

Each file corresponds to the specific time recordings of measurements that have been produced by a smart meter. The *house number* and *datetime* of recordings are included in .csv file as follow:

House_{number}_{date}_{time}.csv

An example is shown below:

House_11_2015-03-07_13.csv

where the data is produced in House 11 and in the time between 13:00 and 13:59 on the day of 07-03-2015. Here, the date is represented in day-month-year format. The file in the example is recorded 1-hour granularity. Hence the number of 13 corresponds to the 13th hour.

These files contain raw consumption data and are transmitted to the fog layer via POST requests by the smart meter client (3). Similarly, temperature data is transmitted by the weather station client. The power measurements are transformed to energy (4), and the final data point is assembled with the energy, temperature, and temporal features by the data management service (5). Then, the preprocessed data set is saved (6).

For example, the preprocessed data belonging to `House 11` at hour 13 eventually appears as in Table 4.2.

Table 4.2 : Hourly Data set

Time	Weighted Power	Temperature	Month	Day	Hour
2015-03-07 13:00:00	562.800	15.040	3	6	13

All timestamped household data sets are aggregated for the hour i (7.1). Then, the data management service sends the yielded hourly level data for prediction within a POST request to the `/predict_hourly` endpoint of the inference service (8.1). The pre-saved models to be used in the load prediction of the hour $i + 1$:

- $LSTM_C$
- SVR_C
- $LSTM_B$ inner model hour i
- SVR_B inner model hour i

The target models are loaded from the disk (9.1), and prediction is done through these models (10.1). The inference service returns the predicted load value for the hour $i + 1$ to the data management service (11.1). Lastly, the data management service returns the value to the edge client (12.1), allowing the operator to see the output (13.1). If all timestamped household data sets are not properly uploaded to the fog layer, the process exited with 500 HTTP code (7.2) and the error message is displayed to the operator (8.2).

5. EXPERIMENTS AND RESULTS

5.1 Experimental Setup

5.1.1 Software and Hardware

The implementation of all software components described in Chapter 4 was done using Python 3. For the ensemble model explained in Section 4.2.2.2, the LSTM networks were implemented with Keras¹ library built on top of Tensorflow² library, while the SVR models were applied using the scikit-learn³ library. As the fog nodes, 3 of Raspberry Pi 4⁴ microcomputers with Broadcom BCM2711 quad-core Cortex-A72 CPU cores 1.5GHz processor and 4 GB RAM were placed in a LAN. The services of energy demand forecasting application were deployed into the Docker containers on these fog nodes. To compare the performance metrics of fog and cloud, we launched AWS EC2⁵ T2 instance with Intel Xeon 1 vCPU and 1 GiB Memory. The clients in the edge layer were established in a computer with Intel Core i5-6300U 2.50 GHz processor and 16 GB RAM.

5.1.2 Dataset

We utilized the publicly available REFIT electrical load dataset to perform hour ahead energy prediction with the proposed energy demand forecasting application. The dataset contains 1,194,958,790 readings of two-year power measurements from 20 houses located in Loughborough, United Kingdom [88]. The power consumption data in 6-10 second intervals is presented in both appliance and aggregated levels. From this aspect, the dataset is appropriate to perform the analysis on STLF models. The

¹<https://keras.io/>

²<https://www.tensorflow.org/>

³<https://scikit-learn.org/stable/>

⁴<https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>

⁵<https://aws.amazon.com/ec2/>

dataset also includes the measurements of air temperature, rainfall, humidity collected in 15-minute granularity.

5.1.3 Training Parameters

In this step, we, as operators, used the fog framework to train the model with the target data set. We identified the 10-week aggregated consumption level data for the experimental setup, and separated the hourly dataset into training, validation, and test sets in 70:20:10 ratio through the edge clients. The load dataset has demonstrated daily seasonality in the data decomposition phase. Furthermore, correlation analysis has shown the lag 1 and 24 autocorrelations are significantly high, as shown in Figure 5.1.

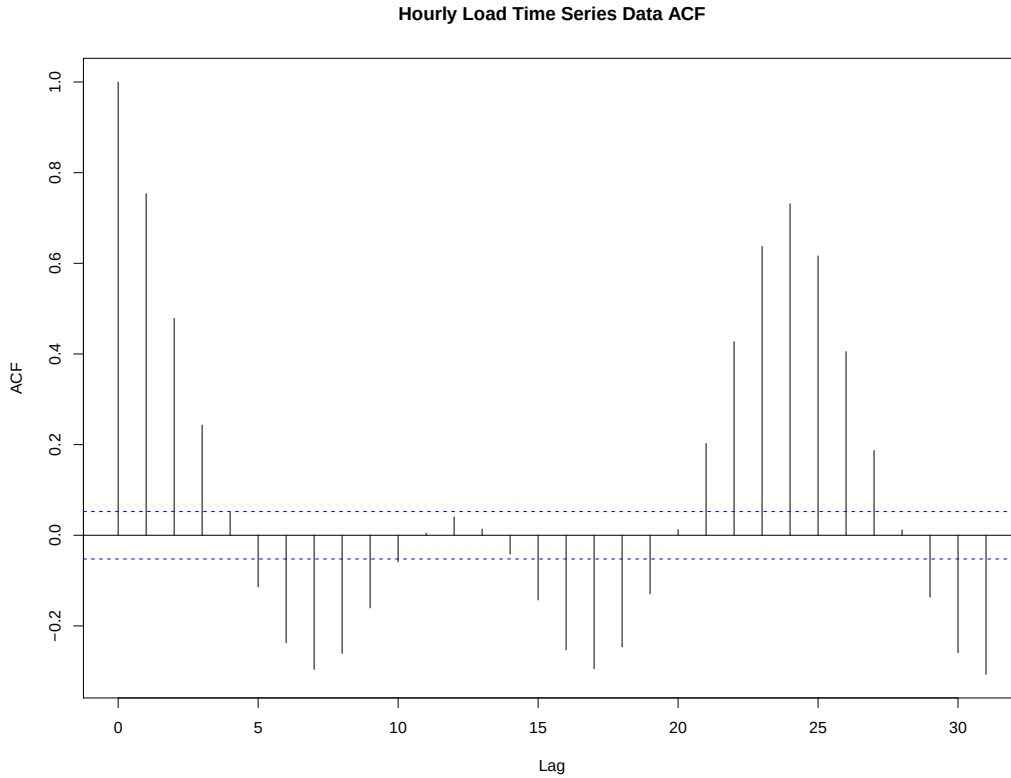


Figure 5.1 : Autocorrelation Function of Historical Load Data

Therefore, we trained a single model for $LSTM_C$ and 24 individual models for $LSTM_B$ in the *training service*. We also applied the same approach for SVR_C and SVR_B . All LSTM networks were compiled with ADAM optimizer [89] and with MSE as the loss function. Regarding tuned model hyperparameters, $LSTM_C$ model was trained with 32 nodes and 16 batch size in 200 epochs, while $LSTM_B$ inner networks were trained with node count ranges from 8 to 64 and batch sizes from 16 to 32 in epochs from 16

to 96. On the other hand, SVR_C model was trained with linear kernel, while the SVR_B models were trained with the kernels including linear, sigmoid, radial basis function, and polynomial.

5.1.4 Evaluation Methods and Metrics

5.1.4.1 Evaluation Methods of Prediction Accuracy

We compared the *prediction accuracies* of the trained models on the unbiased test set. We used mean absolute percentage error (MAPE), root mean squared error (RMSE), and mean absolute error (MAE) as the evaluation metrics.

MAPE is the average of absolute percentage errors of forecast values. The error is calculated as follow:

$$MAPE = \frac{100}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right| \quad (5.1)$$

where n is the number of data points, A_t is the actual value and F_t is the forecast value. The absolute percentage errors are summed to calculate MAPE. This measure is easy to interpret since it yields the error in terms of percentages. It also allows comparing forecasting methods across different time series.

RMSE is the standard deviation of the residuals that are the difference between forecast and actual values. Residuals show how far predictions fall from the regression line data points. RMSE aggregates the quantities of the errors in forecasts for different data points into a single measure. RMSE is calculated as follow:

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}} \quad (5.2)$$

where n is the number of data points, y_i is the actual value, and $\hat{y}_i$ is the forecast value. RMSE is suitable to compare forecasting errors of different methods for a specific data set, since it is scale-dependent. RMSE penalizes large errors more.

MAE is the average of all absolute differences between actual and forecast values. The formula of MAE is shown below:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - x_i| \quad (5.3)$$

where n is the number of data points, x_i is the actual value, and y_i is the forecast value. MAE gives all differences equal weight.

As the base model in model comparison, we used SARIMA which is a popular statistical-based model due to its simplicity and the ability to deal with temporal effect corresponding to autocorrelation, trend, and seasonality in load forecasting [79].

5.1.4.2 Evaluation Methods of Timing

To evaluate the timing performance of the fog framework, we studied the *network latency* and *execution time* of the ensemble model training.

The reduced network latency is significant for the microgrid applications that require the transmission of the big data collected from a vast number of houses. If the network latency is high, the volume of the data transmitted in a unit of time, i.e. throughput decreases. In our fog framework, we measured the latency using Transmission Control Protocol (TCP) for both scenarios, where the data is propagated within the job requests from the master fog node to worker nodes or the cloud node. We run the following steps in a loop on the master node:

1. Create a low-level socket
2. Record start time
3. Connect to the remote destination socket, namely worker fog node or cloud node
4. Record end time
5. Measure the time difference as latency

Then, we calculated the average of latency values from all iterations. We used the output value as *network latency* metric in our experiments.

The execution time of model training is also a valuable metric for the STLTF applications. Time-efficient re-trainings are required to update the model based on the new data at some intervals. For this, the operator executes the ensemble learning

pipeline in Figure 4.4. As an illustrative case, we measured the execution time for the stage of *model deployment* that corresponds to the feature of *saving best models* in Figure 4.7. We followed the steps below:

1. Make POST request to */evaluate* endpoint
2. Record start time
3. Wait for model trainings with best parameters, and final model savings
4. Record end time
5. Measure the time difference as execution time

We conducted experiments to compare *execution time* in changing environments (i.e., fog nodes or cloud) as well as varying application deployment models (i.e., number of replication containers).

5.2 Results

5.2.1 Results on Prediction Accuracy

In first scenario, we compared accuracy performances of the proposed ensemble model and base models. For this experimental scenario, we aggregated historical data from 10 houses into a single data set. Except for SARIMA, we trained all sub models with multivariate data including air temperature and temporal data. We observed heuristically that sub models trained with these additional features gave better prediction results than when trained with load values only.

Table 5.1 indicates the performance of each model. Although SARIMA is trained with the load data only, it yielded better RMSE and MAE scores from the sub model SVR_C . It emphasizes that SARIMA is effective at dealing with the seasonality using its periodic components despite its simplicity. Besides, $LSTM_B$ and SVR_B performances are relatively close to each other, while $LSTM_C$ performance is slightly better. The number of samples used to train each inner network in $LSTM_B$ is less than in $LSTM_C$. Therefore, the inner networks of $LSTM_B$ were trained with a smaller range of epochs since they tend to overfit in the high number of epochs. However, $LSTM_C$ still

outperformed at generalizing the model with a larger data set in our case. The sub model SVR_C was unable to capture the temporal pattern of the continuous data set and produced the highest RMSE and MAE results. This was within our expectations, since LSTM network often outperforms SVR model at modeling non-linear data and producing dynamic prediction results. The proposed ensemble model gave the most accurate results among the sub models and the base model SARIMA, with RMSE of 132.5 Wh, MAPE of 19.9% and MAE of 103.5 Wh.

Table 5.1 : Comparison of Prediction Errors

Model	RMSE (Wh)	MAPE (%)	MAE (Wh)
LSTM _C	143.6	21.4	111.7
LSTM _B	147.1	21.0	113.9
SVR _C	166.3	23.8	127.4
SVR _B	148.5	24.6	117.3
SARIMA	156.9	27.0	125.4
The proposed model	132.5	19.9	103.5

Fig. 5.2 shows the actual and predicted load values yielded by all models on test samples for two consecutive days. We observed that all baseline models are able to follow the trend of the actual data adequately.

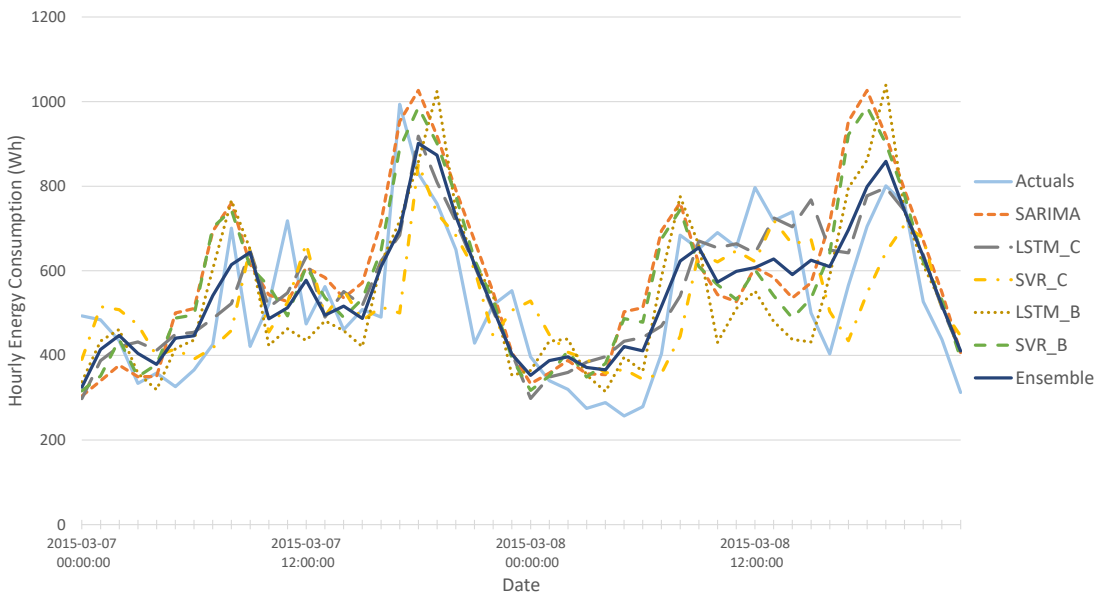


Figure 5.2 : Prediction Results of Compared Models in Two Consecutive Days

We also calculated MAPE values yielded by baseline models and ensemble model for each sample in the test data. The comparison of the lowest and highest error obtained among all models with the error obtained from the ensemble model is illustrated in Figure 5.3. The best performance line represents the lowest MAPE values produced among the baseline models, while the worst performance line corresponds to the highest MAPE values produced among the baseline models. The ensemble model was able to approximate the best or average performance for each sample in test data, which gave robustness to the forecasting system.

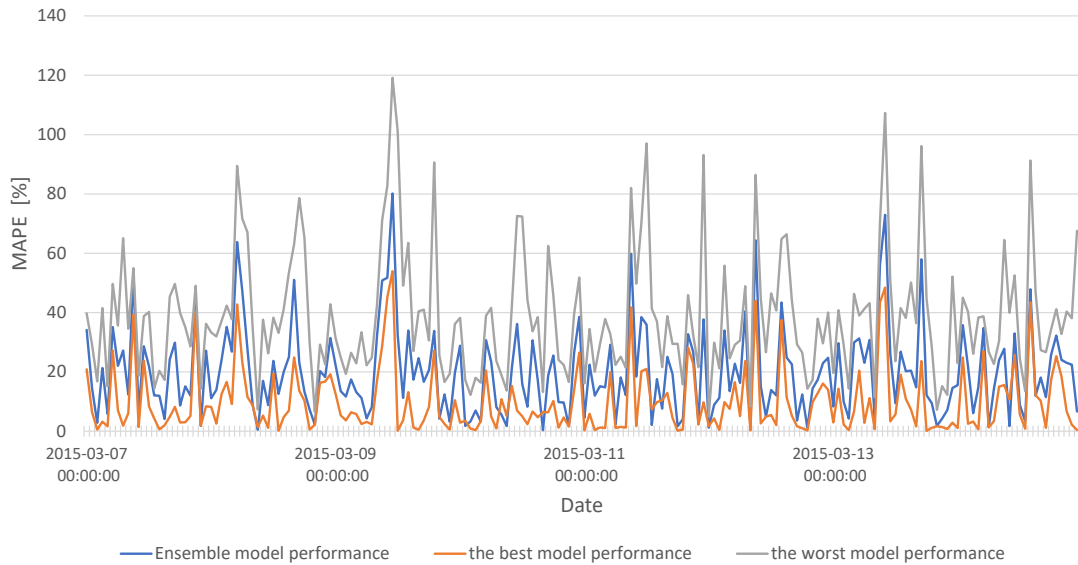


Figure 5.3 : MAPE Results of the Ensemble Model against the Best and Worst Performance

In addition, we evaluated the effect of number of houses involved in the aggregated data on predictive success of the models. In this experiment, we first prepared three separate data sets corresponding to the average power consumption of chosen 5, 10, and 20 houses, respectively. Using these data sets, we trained three groups of sub models, $LSTM_C$, SVR_C , $LSTM_B$, SVR_B , and also the ensemble model. The MAPE values of three cases with 5,10,20 houses are shown in Figure 5.4.

It can be clearly observed that the MAPE values of all models are in a decreasing trend against the increasing aggregation level. MAPE yielded from the ensemble model was calculated as 32.1% when training an aggregated dataset of 5-houses, and 19.9% when

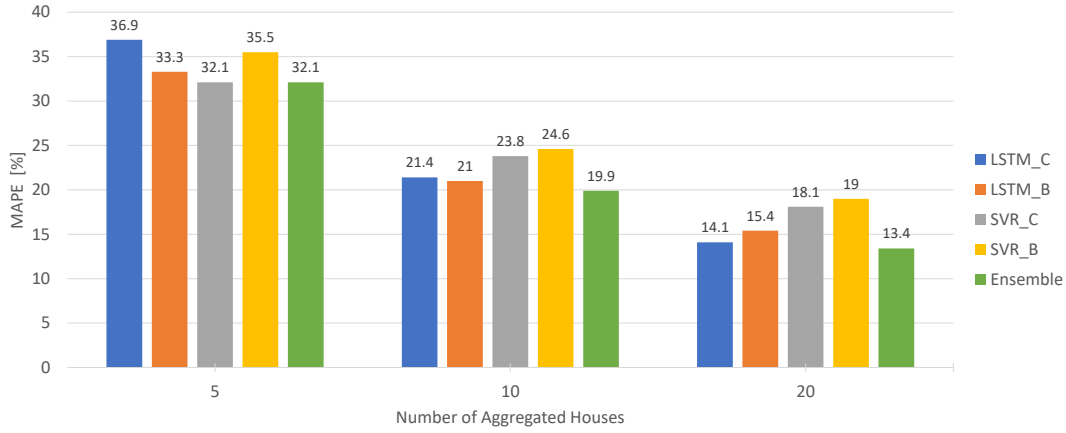


Figure 5.4 : Number of Aggregated Houses versus MAPE Results

training on an aggregated data set of 10-houses. The MAPE of 13.4% was the lowest error value in the case of training with aggregated data from 20-houses. Therefore, we deduced that as the number of houses included in the data set increased, the trained models predicted more accurately. It demonstrated that introduced variability on consumption behavior increased the prediction ability of the models. We also found that the ensemble model produced lower MAPE than the baseline models in the cases of 10-houses and 20-houses. Besides, the ensemble model MAPE result converged to the performance of one of the baseline models in the 5-houses scenario.

Furthermore, we evaluated the performance of the ensemble model in two cases: 1) *forecasting the aggregated load* of the individual houses, 2) *aggregating the load forecasts* of the individual houses. The first approach has already been used in the previous experiments. In this experiment, we aimed to observe the effect of aggregating individual load predictions on the final prediction success.

For this purpose, we trained individual sub models for each house, i.e., LSTM_C, SVR_C, LSTM_B, SVR_B and the ensemble model. We obtained predictions on the test data of each house using these models. After that, we calculated the average of the predictions of each house as the final forecasting result. The averaging was performed on the predictions of the same house groups with the previous experiment, namely 5,10, and 20 houses. As a result, we acquired forecasts that are comparable with the results of the previous approach.

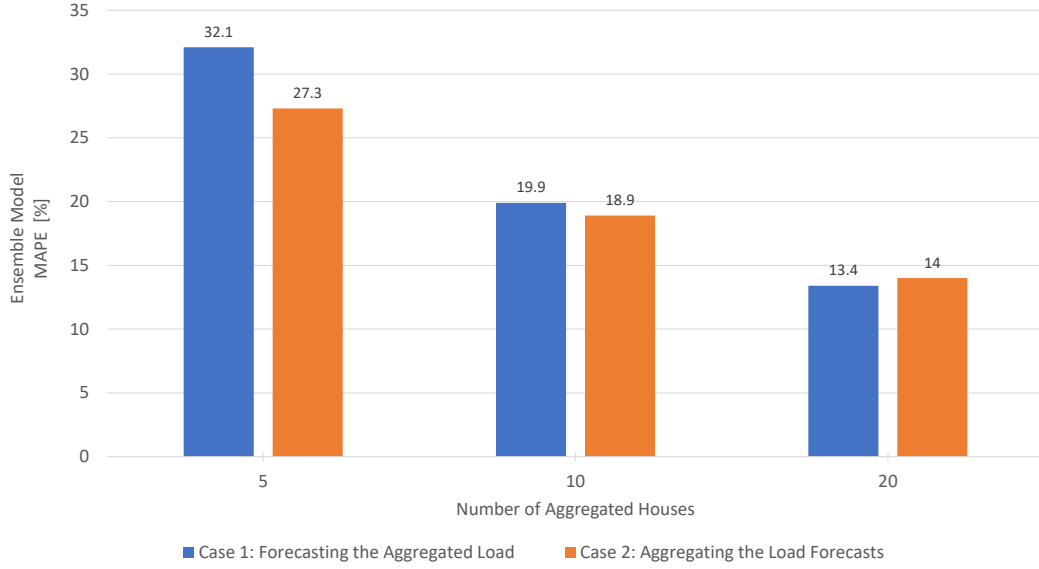


Figure 5.5 : MAPE Results: Forecasting the Aggregated Load versus Aggregating the Load Forecasts

Figure 5.5 shows the MAPE results yielded by ensemble model in both cases. In the 5-houses aggregation level, the measured MAPE of 27.3% when aggregating the forecasts of all houses was significantly lower than the MAPE of 32.1% produced when forecasting the aggregated load. The measured MAPE of 18.9% when aggregating the load forecasts was still lower than the MAPE of 19.9% in the level of 10-houses, yet the error difference between the cases was slighter. In the 20-houses level, forecasting the aggregated load yielded a lower MAPE result (13.4%) compared to aggregating the load forecasts (14%). The MAPE summary demonstrates that aggregating the forecasts performs better when there are fewer houses in the group. However, when the number of houses in the group increased to 20, forecasting the aggregated load brought about a better forecasting performance with a lower MAPE.

The time complexity of forecasting the aggregated load is $O(1)$ or constant, since the input size, i.e., number of the houses, doesn't affect the complexity (the complexity introduced by the individual model parameters, e.g., epoch, batch size, has been ignored). The data set of each house is aggregated into a single data set, regardless of the number of individual houses. On the other hand, the time complexity of aggregating the load forecasts is $O(n)$ or linear, since a particular model is trained on

the data set of each house. In that case, the complexity linearly increases as the number of houses increases involved in training. It is anticipated that the complexity will severely affect the timing when the group of houses is large. Thus, the first case, i.e., forecasting the aggregated load, is advantageous in terms of time and computational complexity.

5.2.2 Results on Timing

In the timing experiments, we first measured the TCP latency for two types of communication, between the master fog node and worker nodes, and between the master fog node and the cloud node. As a result, the average latency was measured at 5.29 ms between the master and worker fog nodes located in a LAN. The average latency was 44.54 ms between the master fog node and the cloud node, higher due to multi-hop communication.

We also investigated the effect of various deployment setups, i.e., number of running fog nodes and replica containers, on the execution time of training tasks. In the proposed application, $LSTM_C$ and SVR_C consist of one each model, while $LSTM_B$ and SVR_B contain each of the 24 models. Thus, a total of 50 models need to be trained in total per each correlated data sequence. We trained $LSTM_C$, SVR_C , $LSTM_B$ and SVR_B models in Docker containers concurrently. We aimed to observe the speedup impact of concurrency on the training tasks.

The concurrency was introduced by a Python library called `concurrent.futures`, as mentioned in Chapter 4. This library provides a class called `ThreadPoolExecutor` which uses a pool of at most maximum workers threads to execute calls asynchronously. The executor class adjusts the number of maximum workers according to the number of processors on the machine. Therefore, we assumed that the higher number of processors enables as much code as possible to execute concurrently. To take the advantage of resource-efficient Docker containers, their CPU and memory resources should be limited. Otherwise, a container can utilize all resources in the host. Therefore, we identified the CPU and memory limits of each containers executing the services. The allocated CPU and memory resources increase proportionally to the number of containers. Thus, we assumed that

the services with more replica containers enable the higher concurrency speedup as long as the machine resources suffice.

The execution times that have been calculated in each deployment setup are shown in Table 5.2. We configured the training service as a 3-container service at first. When we placed the containers in the master fog node only, training took 387.76 ms. In case the containers were placed in both the master fog node and 1 worker fog node, training was completed in 297.49 ms, better than the first measurement. When we further spread the containers into master and 2 worker fog nodes, we measured the execution time in 254.59 ms. It validated our assumption that an increasing number of processors speeds up the concurrent code execution. The last experiment was repeated by increasing the number of containers to 5. The execution time was 208.67 ms, and here we found the result was further improved. The better execution time confirmed our assumption that higher concurrency speedup can be achieved with more containers. On the other hand, the execution time was measured at 90.73 ms in the cloud. As expected, training in the cloud node yielded the best execution time because of more powerful processors.

Table 5.2 : Comparison of Execution Times

Deployment Setup	Execution Time (ms)
Master fog node (3-container service)	387.76
Master and 1 Worker fog nodes (3-container service)	297.49
Master and 2 Worker fog nodes (3-container service)	254.59
Master and 2 Worker fog nodes (5-container service)	208.67
Cloud	90.73

As a result, we observed that the fog framework is promising for low-latency applications and can help in achieving execution times converging to the cloud with its distributed and cost-effective resources.



6. CONCLUSION

The distribution grids are vulnerable to outages resulted from extreme and destructive weather events. When parts of the grid are equipped with distributed energy sources, they can continue serving other loads and meeting local energy needs with the local generation. This concept refers to islanding. Microgrids are electrical systems that can disconnect from the larger grid, handling intentional islanding. Microgrids differ from a simple distributed energy system since they keep the power flowing when the central grid fails. These localized grids can achieve better grid reliability as they can manage the generated and consumed power themselves.

A robust energy demand management can ensure the interaction between microgrids and the distribution network. Short-term load forecasting is vital for localized supply-demand balance in microgrids. An accurate load forecasting in microgrids is vital due to several reasons. Firstly, emerging actors such as RES and EVs in microgrids require careful energy planning. Besides, forecasting at such small scales is more challenging than the large-scale forecasts, due to the higher variability of the load relative to its mean. The fog computing paradigm can unlock the potential of microgrids by enabling real-time analysis, distributed intelligence, and security and privacy. For this reason, we proposed a fog architecture for computational requirements in microgrids.

The test environments enable researchers and developers to compare target learning models, decide which model is suitable, and design a deployable model for production environments. As the main contribution of this study, we designed and implemented a fog framework that gives tech-savvy people, i.e., operators, an opportunity to create their pipeline for building machine learning and deep learning models and deploy the selected models for smart grid applications. Since application testing and validation are the main focus of this study, the physical setup of smart meters and sensors on the edge layer is not included in our work. The main reasons were to avoid challenges which are the setup of real smart meters and the collection of power consumption data from

various load profiles. Instead, we simulated these devices with client applications that transmit pre-collected load and weather data to the fog layer. The clients also enabled flexible and rapid development. Our motivation was to utilize low-cost edge devices, e.g., Raspberry Pi, with relatively lower computation capabilities as fog servers in testing energy demand management tasks, specifically energy demand forecasting. In this study, we deployed a fog cluster using 3 Raspberry Pi devices for the proposed framework that acts as the fog-based microgrid information management system. We proposed a microservice architecture for the services with separate responsibilities in the energy demand forecasting application. Furthermore, we used the Docker technology to isolate the computational resources of each service efficiently. We also used Docker Swarm to distribute the services across the fog nodes. Therefore, we offered the operators a smooth service deployment model.

We also proposed an ensemble STL model that integrates SVR as a shallow model and LSTM as a deep model. The model is based on the concept of training correlated data sequences to predict hourly data. Therefore, we trained sub models $LSTM_C$ and SVR_C on a continuous series of data, $LSTM_B$ and SVR_B on multiple bundled series of data. The ensemble model takes the weighted average of prediction results produced by sub models. Each sub model was ranked based on their validation RMSE errors. We used the proposed model in the implementation of the energy demand forecasting application. We also created APIs to make the service features accessible when executing the steps of the ensemble learning pipeline. Then, we deployed the application to distributed and low-latency fog nodes to demonstrate the feasibility of complex learning models in the fog-based framework.

We used the publicly available REFIT electrical load data set consisting of 20 household power measurements for training the models. To measure the prediction accuracy, RMSE, MAE, MAPE performance metrics were used for the evaluation of the outputted models and SARIMA as a base model. The sub models are trained on aggregated data set with the measurements of 10 houses. Except for SARIMA, all submodels are trained with multivariate data, namely load, temperature, and temporal features. The temporal data including hour, day of the week, and month, improved the model success, while the effect of weekday-weekend classification on the accuracy was negligible. Therefore, the weekday-weekend classification was not used as a feature to

reduce model complexity. The prediction results on 14-day test samples demonstrated that the ensemble model has superior performance with a MAPE error of 19.9% than the other sub models and the base model. We also performed ensemble model evaluation for 1-day test samples to observe how model accuracy changed across test sets of different sizes. All base models yielded lower MAPE error in the 1-day test set and the resulted MAPE error for the ensemble model was calculated as 15.8%. Since the better performance obtained with a smaller, i.e., 1-day, test set might not be reliable to evaluate the model, we conducted the rest of the model experiments with 14-day testing data. As future work, we considered cross-validation for time series that can help increase reliability on model performances for different sizes of test sets. In our experiments, we treated the forecasting horizon as the next 1-hour for the ensemble model, however, we heuristically found that the prediction error increased in the case of the 24-hour forecasting horizon.

We also analyzed how the number of houses involved in the aggregated data affects the predictive success of the models. We prepared the data sets in 3 aggregation levels, i.e., 5, 10, and 20 houses, by choosing household data sets. We trained the models for each aggregation level and compared the MAPE results. The experiment with the group of 20 houses produced the lowest error of MAPE for all sub models and the ensemble model. The introduced variability on consumption behavior increased the prediction ability of the models.

Furthermore, we evaluated the ensemble model accuracy between two scenarios: forecasting the aggregated load and aggregating the load forecasts. In previous experiments, we have already performed forecasting on the data set aggregated from individual houses. For this experiment, we additionally trained separate ensemble models for each household. Then, we aggregated the forecasts within the same three groups, i.e., 5, 10, and 20 houses. At 20 household level, the measured MAPE when forecasting the aggregated load was lower than the resulted MAPE in aggregating the load forecasts. The approach of forecasting the aggregated load performed better than the other approach as the number of households increased. Forecasting the aggregated load is also computationally efficient due to its lower time complexity.

We also evaluated the timing performance of the proposed framework through the network latency and training execution time metrics. For this, we deployed the training

service with varying configurations, i.e., the number of fog nodes or replica containers, and even in diverse environments, i.e., fog or cloud. For latency analysis, we firstly deployed the data management service that forwards the request to the training service in the master fog node. Besides, we deployed the training service in worker nodes and the cloud, then compared the network latency between the ends. Due to the closer geographic distance, the network latency between the master and worker fog node was lower than the latency between the master fog node and the cloud node.

Moreover, we investigated the effect of various deployment configurations on the execution time of training tasks. We spread the containers into different numbers of fog nodes, i.e., master, master and worker, master and 2 workers, for each experiment set. We trained LSTM_C, SVR_C, LSTM_B and SVR_B models in Docker containers concurrently. When we spread the containers into more fog nodes, the execution time decreased. This situation validated our assumption that an increasing number of processors speed up the concurrent code execution. On the other hand, training in the cloud node took less time than in fog nodes. However, the training in distributed fog nodes considerably improved the time efficiency. We also measured execution time by increasing the number of replica containers in the training service when the rest of the conditions were identical. The decrease in execution time confirmed our assumption that higher concurrency speedup can be achieved with more containers due to the increased CPU and memory resources.

The study demonstrated that the framework with cost-effective, low-latency, and distributed Raspberry Pi fog nodes enables validating the complex learning models proposed for energy demand forecasting in microgrids. Therefore, the framework can contribute to the integration of microgrid applications with fog computing. Moreover, it can help minimize costs and deployment time, prevent catastrophic errors in production systems, and collect customer feedback.

Alongside the test frameworks, the integration of *edge computing* in smart grid information flow can increase efficiency in terms of cost and time. When the data is processed closer to where it is produced, network bandwidth usage and latency decreases. Since small processing tasks are offloaded to the edge layer, the resource utilization and installation costs decrease in the fog and cloud layer. Besides, the smart grid environments are heterogeneous, so the various devices are employed in

information flow. When designing the proposed fog framework, we utilized devices that were identical to each other for simplicity. However, a real scenario requires handling *heterogeneous devices*, which is challenging since these devices might have different operating system architectures or computational capabilities. Moreover, the *resource management* is one of the challenges in fog computing systems. In this study, we used the cloud layer for benchmark purposes, and the operator was the decision-maker on the target where the task requests are forwarded. However, the decision-making mechanisms are needed to schedule training or inference task requests to the fog or cloud computing layers based on their resource availability. In addition, *network management* was also a simulation challenge that we encountered. Since smart grids are large-scale systems consisting of multiple networks, i.e., HAN, NAN, WAN, the network simulation tools can help the operators simulate network delays, packet loss, etc.

As future work, RES and EV's effects will be included in the energy demand forecasting scenario. The microgrid load analysis scenarios to be implemented and evaluated within the fog computing framework will also be expanded. Modeling usage patterns of individual appliances or load disaggregation can be given as examples of these scenarios. Besides, the requirements of handling heterogeneity, resource management, and network management are not focused yet, but they will be considered in future work. The usability of the framework will also be improved based on the human-computer interaction principles and evaluated by interviewing smart grid operators.



REFERENCES

- [1] **Ghasempour, A.** (2016). Optimum number of aggregators based on power consumption, cost, and network lifetime in advanced metering infrastructure architecture for Smart Grid Internet of Things, *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp.295–296.
- [2] **Jalali, F., Vishwanath, A., de Hoog, J. and Suits, F.** (2016). Interconnecting Fog computing and microgrids for greening IoT, *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, pp.693–698.
- [3] **Hernandez, L., Baladron, C., Aguiar, J.M., Carro, B., Sanchez-Esguevillas, A.J., Lloret, J. and Massana, J.** (2014). A survey on electric power demand forecasting: Future trends in smart grids, microgrids and smart buildings, *IEEE Communications Surveys and Tutorials*, 16, 1460–1495.
- [4] **Bera, S., Misra, S. and Rodrigues, J.J.P.C.** (2015). Cloud Computing Applications for Smart Grid: A Survey, *IEEE Transactions on Parallel and Distributed Systems*, 26(5), 1477–1494.
- [5] **Samie, F., Bauer, L. and Henkel, J.,** (2019). Edge Computing for Smart Grid: An Overview on Architectures and Solutions: Design Challenges and Paradigms, pp.21–42.
- [6] **Petrican, T., Vesa, A.V., Antal, M., Pop, C., Cioara, T., Anghel, I. and Salomie, I.** (2018). Evaluating Forecasting Techniques for Integrating Household Energy Prosumers into Smart Grids, *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp.79–85.
- [7] **Jaiswal, R., Davidrajuh, R. and Rong, C.** (2020). Fog computing for realizing smart neighborhoods in smart grids, *Computers*, 9, 1–28.
- [8] **Okay, F.Y. and Ozdemir, S.** (2016). A fog computing based smart grid model, *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pp.1–6.
- [9] **Singh, S. and Yassine, A.** (2019). IoT Big Data Analytics with Fog Computing for Household Energy Management in Smart Grids, volume 256, Springer Verlag, pp.13–22.
- [10] **Hong, W.C.** (2013). *Intelligent Energy Demand Forecasting*, volume 10.
- [11] **Nichiforov, C., Stamatescu, I., Fagarasan, I. and Stamatescu, G.** (2017). Energy consumption forecasting using ARIMA and neural network models, pp.1–4.

- [12] **Wang, L., Mao, S. and Wilamowski, B.** (2019). Short-Term Load Forecasting with LSTM Based Ensemble Learning, *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM) and IEEE Smart Data (SmartData)*, pp.793–800.
- [13] **Tan, M., Jin, J. and Su, Y.** (2019). An Ensemble Learning Approach for Short-Term Load Forecasting of Grid-Connected Multi-energy Microgrid, *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp.497–502.
- [14] **Tang, L., Yi, Y. and Peng, Y.** (2019). An ensemble deep learning model for short-term load forecasting based on ARIMA and LSTM, *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp.1–6.
- [15] **Atzori, L., Iera, A. and Morabito, G.** (2010). The Internet of Things: A survey, *Computer Networks*, 54(15), 2787 – 2805.
- [16] **Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M.** (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, *IEEE Communications Surveys Tutorials*, 17(4), 2347–2376.
- [17] **Ghasempour, A.** (2019). Internet of Things in Smart Grid: Architecture, Applications, Services, Key Technologies, and Challenges, *Inventions*, 4, 22.
- [18] Cisco Annual Internet Report (2018–2023) White Paper, <https://bit.ly/3pCMnbx>, accessed on 2020-04-25.
- [19] **Miao Yun and Bu Yuxin** (2010). Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid, *2010 International Conference on Advances in Energy Engineering*, pp.69–72.
- [20] **Gungor, V., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C. and Hancke, G.** (2011). Smart Grid Technologies: Communication Technologies and Standards, *Industrial Informatics, IEEE Transactions on*, 7, 529 – 539.
- [21] **López, G., Moura, P., Custodio, V. and Moreno, J.** (2012). Modeling the Neighborhood Area Networks of the Smart Grid, pp.3357–3361.
- [22] **Pramudhita, A.N., Asmara, R.A., Siradjuddin, I. and Rohadi, E.** (2018). Internet of Things Integration in Smart Grid, *2018 International Conference on Applied Science and Technology (iCAST)*, pp.718–722.
- [23] **Saleem, Y., Crespi, N., Rehmani, M.H. and Copeland, R.** (2019). Internet of Things-Aided Smart Grid: Technologies, Architectures, Applications, Prototypes, and Future Research Directions, *IEEE Access*, 7, 62962–63003.

- [24] **Lloret, J., Tomas, J., Canovas, A. and Parra, L.** (2016). An Integrated IoT Architecture for Smart Metering, *IEEE Communications Magazine*, 54(12), 50–57.
- [25] **Wang, Y., Lin, W., Zhang, T. and Ma, Y.** (2012). Research on application and security protection of Internet of Things in Smart Grid, *IET International Conference on Information Science and Control Engineering 2012 (ICISCE 2012)*, pp.1–5.
- [26] **Pan, J., Jain, R., Paul, S., Vu, T., Saifullah, A. and Sha, M.** (2015). A Internet of Things Framework for Smart Energy in Buildings: Designs, Prototype, and Experiments, *IEEE Internet of Things Journal*, 2, 1–1.
- [27] **Mohanty, S., Panda, B.N. and Pattnaik, B.S.** (2014). Implementation of a Web of Things based Smart Grid to remotely monitor and control Renewable Energy Sources, *2014 IEEE Students' Conference on Electrical, Electronics and Computer Science*, pp.1–5.
- [28] **Ghasempour, A.** (2016). Optimized advanced metering infrastructure architecture of smart grid based on total cost, energy, and delay, *2016 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pp.1–6.
- [29] **McDaniel, P. and McLaughlin, S.** (2009). Security and Privacy Challenges in the Smart Grid, *IEEE Security Privacy*, 7, 75–77.
- [30] **Wang, Y., Huang, Y., Wang, Y., Li, F., Zhang, Y. and Tian, C.** (2018). Operation Optimization in a Smart Micro-Grid in the Presence of Distributed Generation and Demand Response, *Sustainability*, 10(3).
- [31] **Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L.** (2016). Edge Computing: Vision and Challenges, *IEEE Internet of Things Journal*, 3(5), 637–646.
- [32] **Schmitt, L., Kumar, J., Sun, D., Kayal, S. and Venkata, S.S.M.** (2013). Ecocity Upon a Hill: Microgrids and the Future of the European City, *IEEE Power and Energy Magazine*, 11(4), 59–70.
- [33] **Yaprakdal, F., Yilmaz, M.B., Baysal, M. and Anvari-Moghaddam, A.** (2020). A Deep Neural Network-Assisted Approach to Enhance Short-Term Optimal Operational Scheduling of a Microgrid, *Sustainability*, 12(4).
- [34] **Moradzadeh, A., Zakeri, S., Shoaran, M., Mohammadi-Ivatloo, B. and Mohammadi, F.** (2020). Short-Term Load Forecasting of Microgrid via Hybrid Support Vector Regression and Long Short-Term Memory Algorithms, *Sustainability*, 12(17).
- [35] **Pau, M., Patti, E., Barbierato, L., Estebarsari, A., Pons, E., Ponci, F. and Monti, A.** (2018). A cloud-based smart metering infrastructure for distribution grid services and automation, *Sustainable Energy, Grids and Networks*, 15, 14–25, technologies and Methodologies in Modern Distribution Grid Automation.

- [36] **Meloni, A. and Atzori, L.** (2016). A Cloud-Based and RESTful Internet of Things Platform to Foster Smart Grid Technologies Integration and Re-Usability, pp.387–392.
- [37] **Hussain, M., Alam, M. and Beg, M.** (2018). Fog Computing in IoT Aided Smart Grid Transition- Requirements, Prospects, Status Quos and Challenges.
- [38] **Akram, J., Najam, Z. and Rafi, A.** (2018). Efficient Resource Utilization in Cloud-Fog Environment Integrated with Smart Grids, *2018 International Conference on Frontiers of Information Technology (FIT)*, pp.188–193.
- [39] **Yan, Y. and Su, W.** (2016). A fog computing solution for advanced metering infrastructure, *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T D)*, pp.1–4.
- [40] **Barik, R.K., Gudey, S.K., Reddy, G.G., Pant, M., Dubey, H., Mankodiya, K. and Kumar, V.** (2017). FogGrid: Leveraging Fog Computing for Enhanced Smart Grid Network, *2017 14th IEEE India Council International Conference (INDICON)*, pp.1–6.
- [41] **Tom, R.J., Sankaranarayanan, S. and Rodrigues, J.J.P.C.** (2019). Smart Energy Management and Demand Reduction by Consumers and Utilities in an IoT-Fog-Based Power Distribution System, *IEEE Internet of Things Journal*, 6(5), 7386–7394.
- [42] **Kell, A., McGough, A.S. and Forshaw, M.** (2018). Segmenting Residential Smart Meter Data for Short-Term Load Forecasting, *Proceedings of the Ninth International Conference on Future Energy Systems, e-Energy '18*, Association for Computing Machinery, New York, NY, USA, p.91–96.
- [43] **Ahmad, M. and Suhartono, S.** (2010). *Development of Malaysia People Satisfaction Index base on Malaysian National Key Area (NKRA) View project Research Scheme "Competence based Research" 2017 View project.*
- [44] **Rana, M. and Koprinska, I.** (2016). Forecasting electricity load with advanced wavelet neural networks, *Neurocomputing*, 182, 118–132.
- [45] **Vantuch, T., Vidal, A.G., Ramallo-González, A.P., Skarmeta, A.F. and Misák, S.** (2018). Machine learning based electric load forecasting for short and long-term period, *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp.511–516.
- [46] **Daut, M.A.M., Hassan, M.Y., Abdullah, H., Rahman, H.A., Abdullah, M.P. and Hussin, F.** (2017). Building electrical energy consumption forecasting analysis using conventional and artificial intelligence methods: A review, *Renewable and Sustainable Energy Reviews*, 70, 1108 – 1118.
- [47] **Rodrigues, F., Cardeira, C. and Calado, J.** (2014). The Daily and Hourly Energy Consumption and Load Forecasting Using Artificial Neural Network Method: A Case Study Using a Set of 93 Households in Portugal, *Energy Procedia*, 62, 220–229, 6th International Conference on Sustainability in Energy and Buildings, SEB-14.

- [48] **Hernandez, L., Baladrón, C., Aguiar, J.M., Carro, B., Sanchez-Esguevillas, A.J. and Lloret, J.** (2013). Short-Term Load Forecasting for Microgrids Based on Artificial Neural Networks, *Energies*, 6(3), 1385–1408.
- [49] **Barelli, L., Bidini, G., Bonucci, F. and Ottaviano, A.** (2018). Residential micro-grid load management through artificial neural networks, *Journal of Energy Storage*, 17, 287–298.
- [50] **Zhao, D., Ge, Q., Tian, Y., Cui, J., Xie, B. and Hong, T.** (2020). Short-term load demand forecasting through rich features based on recurrent neural networks, *IET Generation, Transmission Distribution*, 15.
- [51] **Wu, H., Niu, D. and Zongyun, S.** (2017). Short-Term Electric Load Forecasting Based on Data Mining, *International Journal of Engineering and Technology*, 9, 250–253.
- [52] **Jinjin, Z., Qian, Z., Guoli, L., Yuan, M. and Can, W.** (2020). Application of HIMVO-SVM in short-term load forecasting, *2020 Chinese Control And Decision Conference (CCDC)*, pp.768–772.
- [53] **Kong, W., Dong, Z.Y., Jia, Y., Hill, D.J., Xu, Y. and Zhang, Y.** (2019). Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network, *IEEE Transactions on Smart Grid*, 10(1), 841–851.
- [54] **Hossain, M.S. and Mahmood, H.** (2020). Short-Term Load Forecasting Using an LSTM Neural Network, *2020 IEEE Power and Energy Conference at Illinois (PECI)*, pp.1–6.
- [55] **Zheng, J., Chen, X., Yu, K., Gan, L., Wang, Y. and Wang, K.** (2018). Short-term Power Load Forecasting of Residential Community Based on GRU Neural Network, *2018 International Conference on Power System Technology (POWERCON)*, pp.4862–4868.
- [56] **Svorobej, S., Endo, P.T., Bendeche, M., Filelis-Papadopoulos, C., Giannoutakis, K.M., Gravvanis, G.A., Tzovaras, D., Byrne, J. and Lynn, T.** (2019), Simulating fog and edge computing scenarios: An overview and research challenges.
- [57] **Xu, Q. and Zhang, J.** (2019). piFogBed: A Fog Computing Testbed Based on Raspberry Pi, *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pp.1–8.
- [58] **Sonmez, C., Oztgovde, A. and Ersoy, C.** (2017). EdgeCloudSim: An environment for performance evaluation of Edge Computing systems, *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp.39–44.
- [59] **Mayer, R., Graser, L., Gupta, H., Saurez, E. and Ramachandran, U.** (2017). EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures, *2017 IEEE Fog World Congress (FWC)*, pp.1–6.

- [60] **Naas, M.I., Boukhobza, J., Raipin Parvedy, P. and Lemarchand, L.** (2018). An Extension to iFogSim to Enable the Design of Data Placement Strategies, *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pp.1–8.
- [61] **Iorga, M., Feldman, L., Barton, R., Martin, M.J., Goren, N. and Mahmoudi, C.,** (2018), Fog computing conceptual model.
- [62] EdgeCompMarket, <https://bit.ly/3vd987e>, accessed on 2020-04-25.
- [63] Gartner, <https://gtnr.it/352dD9R>, accessed on 2020-04-02.
- [64] **Loghin, D., Ramapantulu, L. and Teo, Y.M.** (2017). On Understanding Time, Energy and Cost Performance of Wimpy Heterogeneous Systems for Edge Computing, Institute of Electrical and Electronics Engineers Inc., pp.1–8.
- [65] **Vujović, V. and Maksimović, M.** (2015). Raspberry Pi as a Sensor Web node for home automation, *Computers and Electrical Engineering*, 44, 153–171.
- [66] **McManus, S. and Cook, M.** (2017). *Raspberry Pi For Dummies*, For Dummies, 3rd edition.
- [67] Benchmarking TensorFlow Lite on the New Raspberry Pi 4, Model B, <https://bit.ly/3g6xXx3>, accessed on 2020-04-26.
- [68] **Al-Rakhami, M., Alsahli, M., Hassan, M.M., Alamri, A., Guerrieri, A. and Fortino, G.** (2018). Cost Efficient Edge Intelligence Framework Using Docker Containers, *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp.800–807.
- [69] **Ismail, B.I., Mostajeran Goortani, E., Ab Karim, M.B., Ming Tat, W., Setapa, S., Luke, J.Y. and Hong Hoe, O.** (2015). Evaluation of Docker as Edge computing platform, *2015 IEEE Conference on Open Systems (ICOS)*, pp.130–135.
- [70] **Morabito, R. and Beijar, N.** (2016). Enabling Data Processing at the Network Edge through Lightweight Virtualization Technologies, *2016 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*, pp.1–6.
- [71] **Mendki, P.** (2018). Docker container based analytics at IoT edge Video analytics usecase, *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pp.1–4.
- [72] **Pan, J. and McElhannon, J.** (2018). Future Edge Cloud and Edge Computing for Internet of Things Applications, *IEEE Internet of Things Journal*, 5(1), 439–449.

- [73] **Rahman, S., Rabiul Alam, M.G. and Mahbubur Rahman, M.** (2019). Deep Learning based Ensemble Method for Household Energy Demand Forecasting of Smart Home, *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, pp.1–6.
- [74] **Sarajcev, P., Jakus, D. and Vasilj, J.** (2020). Ensemble learning with time-series clustering for aggregated short-term load forecasting, *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, pp.553–558.
- [75] **Hyndman, R.J.**, (2001), ARIMA processes.
- [76] **Hyndman, R.J.**, (2001), Box-Jenkins modelling.
- [77] **Lai, X., Yang, T., Wang, Z. and Chen, P.** (2019). IoT implementation of Kalman Filter to improve accuracy of air quality monitoring and prediction, *Applied Sciences (Switzerland)*, 9.
- [78] **Mohamed, N., Ahmad, M., Ismail, Z. and Suhartono, S.** (2010). Short Term Load Forecasting Using Double Seasonal ARIMA Model, *Statistics Faculty of Computer and Mathematical Sciences*, 15, 57–73.
- [79] **Hadri, S., Naitmalek, Y., Najib, M., Bakhouya, M., Fakhri, Y. and Elaroussi, M.** (2019). A comparative study of predictive approaches for load forecasting in smart buildings, volume160, Elsevier B.V., pp.173–180.
- [80] **Cortes, C. and Vapnik, V.** (2009). Support-vector networks, *Chem. Biol. Drug Des.*, 297, 273–297.
- [81] **Makridakis, S., Spiliotis, E. and Assimakopoulos, V.** (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward, *PLoS ONE*, 13.
- [82] **Ahmed, N., Atiya, A., Gayar, N. and El-Shishiny, H.** (2010). An Empirical Comparison of Machine Learning Models for Time Series Forecasting, *Econometric Reviews*, 29, 594–621.
- [83] **Hochreiter, S. and Schmidhuber, J.** (1997). Long Short-Term Memory, *Neural Comput.*, 9(8), 1735–1780.
- [84] **Gers, F., Jürgen Schmidhuber and Cummins, F.** (1999). Learning to Forget: Continual Prediction with Lstm Learning to Forget: Continual Prediction with Lstm.
- [85] **Zhichao, R., Chao, C., Yingying, D., Wentao, Z., Jun, W. and Ruixiao, Z.** (2020). Short-term load forecasting of multi-layer LSTM neural network considering temperature fuzzification, *2020 IEEE Sustainable Power and Energy Conference (iSPEC)*, pp.2398–2404.
- [86] **Pramono, S.H., Rohmatillah, M., Maulana, E., Hasanah, R.N. and Hario, F.** (2019). Deep Learning-Based Short-Term Load Forecasting for Supporting Demand Response Program in Hybrid Energy System, *Energies*, 12(17).

- [87] **Yokotani, T. and Sasaki, Y.** (2016). Comparison with HTTP and MQTT on required network resources for IoT, *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pp.1–6.
- [88] **Murray, D., Stankovic, L. and Stankovic, V.** (2017). An electrical load measurements dataset of United Kingdom households from a two-year longitudinal study, *Scientific data*, 4, 160122.
- [89] **ADAM Optimizer**, <https://keras.io/api/optimizers/adam/>, accessed on 2021-05-30.



CURRICULUM VITAE

Name Surname : Tuğçe Keskin

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Keskin T., İnce G.** 2021: An Ensemble Learning Approach for Energy Demand Forecasting in Microgrids using Fog Computing. *3rd International Conference on Intelligent and Fuzzy Systems*, August 24–26, 2021 Istanbul, Turkey.

