

Master's Programme in Computer, Communication and Information Sciences

# Hybrid Cloud Solution for Mission Critical Applications: A Case Study on Electricity Market Forecasting

---

Leo Suojärvi

© 2025. This work is licensed under a [CC BY-NC-SA 4.0](#) license.

---

**Author** Leo Suojärvi

---

**Title** Hybrid Cloud Solution for Mission Critical Applications: A Case Study on Electricity Market Forecasting

---

**Degree programme** Computer, Communication and Information Sciences

---

**Major** Computer Science

---

**Supervisor** Prof. Valeriy Vyatkin

---

**Advisors** Dr. Seppo Sierla, Dr. Rakshith Subramanya

---

**Collaborative partner** Automous Oy

---

**Date** 10 October 2025

**Number of pages** 68

**Language** English

---

**Abstract**

To ensure the stability of the power grid, it is essential to keep the grid frequency close to a nominal value by balancing electricity production and consumption at every moment. Different electricity markets incentivize generator, storage, and load operators to actively participate in the balancing effort. One key category of markets is frequency reserve markets in which the transition service operator procures reserve capacity from all reserve providers that have generators, storages, or loads prequalified for the market. Providers submit bids before a gate closure deadline, and an auction mechanism determines bid acceptance and the clearing price. For all parties, it is beneficial that the bids are made for the market intervals when the market prices are highest and therefore most needed. To be able to make the bids at the times of the highest prices, accurate and highly available forecasts of the markets are needed. One suitable forecasting model for this is seasonal autoregressive integrated moving average model with exogenous variables (SARIMAX). Due to the gate closure deadline, there is a limited time between the availability of required input data for the model and the deadline by which forecast users require the forecasts to be ready. This creates real-time requirements for forecasting. While prior research has addressed electricity price forecasting and creating highly available real-time systems separately, the integration of these two in the context of electricity market forecasting has not been studied. This thesis aims to bridge this gap by studying how the SARIMAX model can be used to forecast the prices of one of the Finnish electricity reserve markets with high availability using a hybrid cloud approach with diverse redundancy. The thesis begins by introducing the concepts of hybrid cloud, workflow orchestration, redundancy and the forecasting model SARIMAX. After this, a case study of a hybrid cloud system with SARIMAX forecasting for frequency containment reserve for normal operations (FCR-N) market and the results of the system are presented. Based on the results, the SARIMAX model is a viable solution for predicting the FCR-N results and the used system architecture is suitable for this kind of a real-time system.

---

**Keywords** Diverse Redundancy, Electricity Markets, Hybrid Cloud, Mission Critical Application, Orchestration, SARIMAX, Time Series Forecasting

---

---

**Tekijä** Leo Suojärvi

---

**Työn nimi** Hybridipilviratkaisu liiketoimintakriittisille sovelluksille: tapaustutkimus sähkömarkkinoiden ennustamisesta

---

**Koulutusohjelma** Master's Programme in Computer, Communication and Information Sciences

---

**Pääaine** Computer Science

---

**Työn valvoja** Prof. Valeriy Vyatkin

---

**Työn ohjaajat** Dr. Seppo Sierla, Dr. Rakshith Subramanya

---

**Yhteistyötaho** Automous Oy

---

**Päivämäärä** 10. Lokakuuta 2025

**Sivumäärä** 68

**Kieli** englanti

---

### **Tiivistelmä**

Sähköverkon vakaan toiminnan kannalta on välttämätöntä pitää sähköverkon taajuus lähellä nimellistaajuutta tasapainottamalla sähkön tuotanto ja kulutus jokaisella hetkellä. Erilaiset sähkömarkkinat mahdollistavat tasapainottamiseen osallistumisen generaattoreiden, sähkövarastojen ja -kuormien operaattoreille. Olennaisen sähkömarkkinoiden kategorian muodostavat taajuusreservimarkkinat, joilla kantaverkkoyhtiö varaa kapasiteettia reservitoimittajilta, joilla on tarkoitukseen sopivia generaattoreita, sähkövarastoja tai -kuormia. Reservitoimittajat antavat tarjoukset ennen markkinakoh- taista määräaika, ja huutokauppamekanismi määrittää tarjousten hyväksymisen sekä reservin hinnan. Kaikille osapuolille on hyödyllistä, että tarjoukset kohdennetaan hetkiin, jolloin reserveistä on eniten puutetta ja hinnat siten korkeimmat. Jotta tarjoukset voidaan kohdentaa näihin hetkiin, ovat tarkat ja korkeasti saatavilla olevat ennusteet tarpeellisia. Yksi sopiva ennustemalli tähän tarkoitukseen on autoregressiivinen in- tegroiva liikkuvan keskiarvon malli kausivaihtelulla ja ulkoisilla muuttujilla (eng. SARIMAX). Tarjousten määräaika ja toisaalta mallin tarvitseman datan saatavuus ai- heuttavat ennusteille reaaliaikavaatimuksia. Aikaisempi tutkimus on käsitellyt sähkön hinnan ennustamista ja korkeasti saatavilla olevien reaaliaikajärjestelmien toteuttamista erikseen, mutta näiden yhdistelmää reservimarkkinoiden kontekstissa ei ole tutkittu. Tämä diplomityö pyrkii täyttämään tämän aukon tutkimalla, miten SARIMAX-mallia voidaan käyttää reservihintojen ennustamiseen yhdellä Suomen reservimarkkinoista saavuttaen samalla korkean saatavuuden käyttämällä monimuotoista redundanssia hybridipilviarkkitehtuurissa. Diplomityö alkaa hybridipilven, prosessien orkestroinnin, redundanssin sekä SARIMAX-mallin esittelyllä. Tämän jälkeen esitellään käytännön työ, jossa SARIMAX-mallia käytetään hybridipilviarkkitehtuurissa ennustamaan taa- juusohjatun käyttöreservin hintaa sekä työn tulokset. Tulosten perusteella SARIMAX on toimiva malli taajuusohjatun käyttöreservin hintojen ennustamiseen ja käytetty järjestelmäarkkitehtuuri soveltuu tällaiseen reaaliaikaiseen järjestelmään.

---

**Avainsanat** Aikasarjaennustus, Hybridipilvi, Liiketoimintakriittinen sovellus, Monimuotoinen redundanssi, Orkestrointi, SARIMAX, Sähkömarkkinat

---

## **Preface**

I would like to express my gratitude to everyone at Automous for giving me the opportunity to carry out this thesis within the organization. I am particularly thankful to my advisors, Seppo Sierla and Rakshith Subramanya, whose guidance and insightful feedback were invaluable throughout this work. I would also like to thank Valeriy Vyatkin for supervising this thesis. Finally, I want to thank my partner for their encouragement and support throughout this journey.

Espoo, 10 October 2025

Leo Suojärvi

# Contents

<b>Abstract</b>	<b>3</b>
<b>Abstract (in Finnish)</b>	<b>4</b>
<b>Preface</b>	<b>5</b>
<b>Contents</b>	<b>6</b>
<b>Abbreviations</b>	<b>8</b>
<b>1 Introduction</b>	<b>9</b>
<b>2 Hybrid Cloud Architecture</b>	<b>12</b>
2.1 Cloud Computing Overview . . . . .	12
2.2 Cloud Computing for Time Series Forecasting Tasks . . . . .	14
2.3 Hybrid Cloud Patterns . . . . .	15
2.4 Hybrid Cloud Combining Private and Public Clouds: Trade-offs and Design . . . . .	17
<b>3 Workflow Orchestration</b>	<b>20</b>
3.1 Orchestration in Software Systems . . . . .	20
3.2 Orchestration Tools . . . . .	20
<b>4 Redundancy</b>	<b>23</b>
4.1 Redundancy Overview . . . . .	23
4.2 Diverse Redundancy in a Hybrid Cloud Architecture . . . . .	23
4.3 Orchestration Design in a Redundant Hybrid Cloud Architecture . . . . .	24
<b>5 Time Series Forecasting with Autoregressive Integrated Moving Average Models</b>	<b>27</b>
5.1 Introduction to Autoregressive Integrated Moving Average Models . . . . .	27
5.1.1 Autoregressive Moving Average Model . . . . .	27
5.1.2 Autoregressive Integrated Moving Average Model . . . . .	27
5.1.3 Seasonal Autoregressive Integrated Moving Average Model . . . . .	28
5.1.4 Seasonal Autoregressive Integrated Moving Average Model with Exogenous Variables . . . . .	29
5.2 Use Cases of Autoregressive Integrated Moving Average Models . . . . .	29
<b>6 System Implementation and Use Case</b>	<b>32</b>
6.1 Use Case Overview . . . . .	32
6.2 Electricity Market Background . . . . .	32
6.3 General Considerations for Tuning Autoregressive Model . . . . .	34
6.4 Implementing an Autoregressive Model for Frequency Containment Reserve Price Forecasting Application . . . . .	35

6.5	System Architecture and Implementation . . . . .	38
6.5.1	Apache Airflow . . . . .	41
6.5.2	Forecasters . . . . .	44
6.5.3	Database . . . . .	44
6.5.4	Output Aggregation . . . . .	46
<b>7</b>	<b>Forecasting Model Results and Further Tuning</b>	<b>47</b>
7.1	Evaluating the Forecasting Model Performance with Different Parameter Selections . . . . .	47
7.2	Evaluating the Forecasting Model Performance with Different Exogenous Variable Selections . . . . .	53
<b>8</b>	<b>Conclusions and Further Work</b>	<b>61</b>
	<b>References</b>	<b>62</b>






## Abbreviations

ADF	augmented Dickey–Fuller test
aFRR	automatic frequency restoration reserve
AIC	Akaike information criterion
AMD	Advanced Micro Devices (company name)
API	application programming interface
AR	autoregressive
ARIMA	autoregressive integrated moving average
AWS	Amazon Web Services
CD	continuous delivery
CI	continuous integration
CPI	consumer price index
DAG	directed acyclic graph and a pipeline in Apache Airflow
ETL	extract, transform, load
FaaS	function as a service
FCR-N	frequency containment reserve for normal operation
FCR-D down	frequency containment reserve for disturbances downwards
FCR-D up	frequency containment reserve for disturbances upwards
FFR	fast frequency reserve
GARCH	autoregressive conditional heteroskedasticity
GCP	Google Cloud Platform
GPU	graphics processing unit
HMAC	hash message authentication code
IaaS	infrastructure as a service
MA	moving average
MAE	mean absolute error
mFRR	manual frequency restoration reserve
ML	machine learning
MWAA	Amazon Managed Workflows for Apache Airflow
NoOps	no operations
OCSB	Osburn, Chui, Smith, Birchenhall test
OS	operating system
PaaS	platform as a service
RMSE	root mean squared error
SARIMA	seasonal autoregressive integrated moving average
SARIMAX	seasonal autoregressive integrated moving average with exogenous regressor
SaaS	software as a service
TSO	transmission system operator
UI	user interface
VPN	virtual private network

# 1 Introduction

A stable grid frequency near the nominal frequency is essential for the operation of an electricity grid. In many countries, including Finland, this is achieved by utilizing a complex electricity market that forms a price for electricity and reserve products that can adjust the grid frequency when needed. In practice, this means that the grid operator buys capacity from electricity reserve markets according to the estimated need. In Finland, the market prices are based on the bids that the reserve producers make. The Finnish reserve products are summarized in the figure 1..

## Reserve market places in Finland

	<b>FFR</b>	<b>FCD</b>	<b>FCRN</b>	<b>AFRR</b>	<b>mFRR</b>
	Fast Frequency reserve, Finland 20 %, In Nordics, total 0-300 MW (estimate)	Frequency Containment Reserve for Disturbances, Finland 290 MW, Nordics total 1450 MW	Frequency Containment Reserve for Normal Operation, Finland 120 MW, Nordics total 600 MW	Automatic Frequency Restoration Reserve, Finland 60-80 MW, Nordics total 300-400 MW	Manual Frequency Restoration Reserve Reference incident + imbalances of balance responsible parties
<b>Activated</b>	In big frequency deviations, In low inertia situations	In big frequency deviations	Used all the time	Used in certain hours	Activated if necessary
<b>Activation speed</b>	In a second	In seconds	In a couple of minutes	In five minutes	In fifteen minutes
					

**FINGRID**

**Figure 1:** Reserve products and reserve market places in Finland [1]

To optimize the use of the reserve capacity, it would be best if the capacity was available when the market price is highest and therefore the need for the capacity is strongest. This is especially important if the capacity cannot be used constantly. An example of these limited capacity reserves is a battery energy storage system. With this kind of capacity it is important to schedule its use in the most useful and profitable way. This requires accurate forecasting of reserve market prices to optimize the use of the battery or other capacity. In recent years the use of time series forecasting methods for electricity market predictions has increased [2, 3].

The possibilities of running time series forecasting models have changed with cloud computing. Cloud computing makes it possible to run time series forecasting models on expensive equipment without large initial investment, with relatively high availability and easy deployment by providing infrastructure and platform as a service (PaaS). The ease of deployment stands out particularly when using SaaS (software as a service) [4] or FaaS (function as a service) [5] solutions that provide ways to rapidly and with a relatively low effort deploy applications like machine learning models.

Despite the high availability and reliability of cloud resources, there is always some downtime, even when deploying to multiple physical locations within the same

cloud. On top of this, one special characteristic of the electricity reserve market forecasting is that the forecasts might be required in real-time. This becomes necessary for best performance especially in intraday markets and if using exogenous data such as weather forecasts in addition to historical market data to make predictions. The weather forecasts improve when they are made closer to the time being forecasted, and therefore creating the forecasts as late as possible is important. Even if not using exogenous data, it is beneficial to run the forecasts as fast as possible so that there is more time to make or calculate decisions based on the results. This causes a situation where it is not possible to wait for the cloud to be up again and run the model then because the waiting time will cause suboptimal use of the reserves and losses in profits. For this problem, hybrid cloud is a possible solution [6].

Hybrid cloud means combining multiple clouds into a single system. The clouds can either be private or public. Private cloud typically means that the cloud is running on on-premises machines but it is possible that the private cloud is hosted in a third party server but separated logically from other customers. Typical for private cloud is that the cost does not depend on the amount of use due to the capacity being reserved for the single customer. By combining multiple clouds, there is potential for increased availability and possibility to run a backup forecasting model in another cloud. The drawback of hybrid cloud architecture is increased complexity and depending on the application increased cost. However, if the hybrid cloud use can be optimized, it is possible to make cost savings with it [7].

When it comes to the forecasting models used for predicting time series data, one of the common choices is ARIMA (autoregressive integrated moving average) [8]. It has gained popularity for its ability to detect complex patterns in time series data. If the dataset contains seasonality, it is beneficial to use a SARIMA (seasonal ARIMA) model [9] that can detect the seasonality. If also exogenous variables that could be useful in capturing the market complexity are available, it may be beneficial to use SARIMAX (SARIMA with exogenous variables) [10]. However, this requires careful selection of exogenous variables to achieve high model accuracy.

Because of the complex nature and high resource demands, it is possible that time series forecasting tasks fail during the execution. This can be caused, for example, by the system resources such as memory running out or a bug in the code that occurs only in specific conditions such as with specific input parameters. For a mission critical time series forecasting application these are problematic risks that should be taken into account. One possible solution for this is diverse redundancy. In this time series forecasting context it means that the same model is run, for example, with different input parameters in different systems to increase availability by lowering the probability that both systems fail due to the same reason.

This thesis aims to answer these **research questions**:

1. How can a hybrid cloud architecture be used to improve the availability of the electricity market forecasting system?
2. How can SARIMAX be configured for reserve market forecasting, specifically in the frequency containment reserve market for normal operation (FCR-N)?

3. How can a hybrid cloud architecture support a mission-critical time series forecasting application with diverse redundancy?

The **hypothesis** is that the hybrid cloud architecture is applicable to real-time mission critical systems in electricity market participation.

The rest of the thesis is structured as follows. The core concepts of cloud computing and hybrid architectures are introduced in chapter 2. Chapter 3 discusses orchestration and tools that can be used for orchestration focusing on the time series forecasting applications. In chapter 4 the concept of diverse redundancy is explained and linked to hybrid cloud architecture. Chapter 5 discusses the principles of ARIMA, SARIMA and SARIMAX and presents some use cases for them. Chapter 6 presents an experiment where a hybrid cloud system is used to implement an ARIMA-model with diverse redundancy and use cases for the system. In chapter 7 the results from the experiment are presented and discussed. Finally, chapter 8 concludes the thesis and discusses ways in which the experiment can be further improved.

## 2 Hybrid Cloud Architecture

Hybrid cloud computing means combining multiple clouds into a single system to gain some benefit. The system may include a combination of public and on-premise clouds or multiple public clouds or multiple on-premise clouds depending on the requirement [11]. The benefits of a hybrid cloud setup may involve, for example, load distribution, security and cost-effectiveness [11] [12].

### 2.1 Cloud Computing Overview

Cloud computing is the delivery and use of computing services like servers, storage, and software over the internet in a way that the client does not need to have their own infrastructure. In contrast to traditional on-premises systems, cloud systems provide scalability and managed infrastructure. This allows developers to focus on modeling and improving the products instead of focusing on maintaining and possibly scaling the systems.

The existing top cloud service providers are Amazon Web Services, Microsoft Azure and Google Cloud Platform [13]. There are some differences between the services but for the most part the features are similar. The differences come from, for example, available areas and level of integration to the provider's own systems [13]. On top of these regular cloud providers there are also some specialized cloud providers such as Finnish LUMI [14] that offers supercomputer capacity to the customers.

Cloud models come in four types:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)
- Function as a Service (FaaS)

IaaS means providing virtualized computing resources such as storage, PaaS means providing a runtime environment such as virtual computer and SaaS means providing the software and cloud infrastructure for the clients to use [15]. Function as a service means that the cloud provider offers everything but the function logic and the user only needs to write the function logic, deploy it, and set resource limits before the function is running [16].

Cloud computing is based on shared resources on large servers, which makes it possible to use the resources in a different way than what has traditionally been possible. One of these concepts is serverless hosting. Serverless is a cloud-based code execution model where the cloud provider does not lock resources for one customer but dynamically provisions resources to each customer depending on the need. The serverless model can be beneficial for both the user and the platform provider due to more efficient use of resources. The benefits of serverless include functionality with no operations (NoOps) such as automatic execution of code, auto-scaling, invoicing based on service usage and separating computation and storage [17]. As a drawback,

**Table 1:** Overview of Cloud Service Models

Model	Description	Example Services
IaaS (Infrastructure as a Service)	Provides virtualized computing resources such as virtual machines and storage. Users manage the OS and software themselves. [15]	Google Compute Engine, Azure Virtual Machines, Amazon EC2
PaaS (Platform as a Service)	Provides a runtime environment to deploy and manage user's applications without managing the underlying infrastructure. [15]	Vertex AI, Azure App Service, AWS Elastic Beanstalk
SaaS (Software as a Service)	Provides complete applications running on cloud infrastructure, managed entirely by the provider. Typically accessed via a web interface or API. [15]	Colab, Github, Amazon WorkSpaces
FaaS (Function as a Service)	Provides possibility to use and implement functions natively in a cloud environment. User provides only the function and resource limits. The resources are automatically scaled to the function needs [16].	Google Cloud Functions, Azure Functions, Amazon Lambda

serverless models typically have some constraints, such as machine types or limited resources. One typical issue with serverless model is a delay before executing due to limited resources on the server and the provider trying to optimize the resource use from their perspective [18].

The cost of serverless compared to server-based cloud computing from a user perspective depends on the scenario. One scenario where cloud computing is clearly more cost-efficient than traditional on-premises infrastructure is only rarely needed services or services which have large fluctuations in the resource needs. If the infrastructure is used only for a fraction of time or the infrastructure needs to be scaled to take into account rare spikes in resource needs, the serverless model is likely a more cost-effective solution [19].

Despite the benefits of cloud computing, there are also some drawbacks compared to on-premises hosted servers. The main drawbacks include:

- **Vendor lock-in:** It can be difficult to change cloud vendors, especially if using vendor-specific tools.
- **Availability:** Although cloud services typically have very good availability, the user cannot affect this during outages.
- **Data security:** The user needs to trust the cloud provider if using a public cloud in regards to data security.
- **Infrastructure options:** There is usually a good selection of available infrastructure such as GPUs, but the user must select from what is available.

Limited availability of services and locking to the vendor are drawbacks of cloud services. If services are not natively available in the cloud, the user needs to either change to use other services or host the services themselves, for example, in a virtual machine. This counters the benefit of easy management. Using services that are unique to the cloud provider can result in a vendor lock-in problem. With cloud storage, this could cause a situation where the cloud provider decides to increase the cost of a service and for the user, it is difficult to change to other service [20]. For the user this is a difficult situation where they need to consider if it is worth the labor cost to implement the system in another environment. Another possible issue might be availability with a particular cloud provider [20]. If the cloud is not available due to issues that the user cannot affect, the user has no other choice than to wait for the cloud provider to fix those issues. For some businesses this can be a very difficult and expensive situation. For these issues [20] proposes a multi-cloud hybrid approach that can be used to access the data either in the cheapest way or from another source when the original source is unavailable. A similar approach is used in this thesis in the experiment in chapter 6.

## 2.2 Cloud Computing for Time Series Forecasting Tasks

Time series forecasting tasks typically require significant computing resources, storage and if provided as a part of a SaaS solution, considerable management and scaling. This makes cloud computing an appealing solution because cloud PaaS solutions provide all of these. There are also some SaaS solutions for time series forecasting that make the deployment even easier but are a bit more limited in what is possible.

Time series forecasting often involves adjusting thousands to millions of parameters, especially if using machine learning methods. This is why these tasks typically require more computing power compared to regular computing tasks. They might require a specialized GPU and often a lot of storage and orchestration to handle the repeating runs of the models. Cloud computing can usually answer these needs well by providing scalable on-demand infrastructure that can even be specialized for machine learning tasks. This chapter discusses the specific benefits, drawbacks, challenges, and patterns of using cloud services for time series forecasting.

A characteristic of time series forecasting tasks is that while the resource needs are high, they are not constant. This might cause low use for expensive equipment which decreases the cost-effectiveness. For this problem, a serverless cloud solution is a good fit, since the user gains access to expensive equipment much more cost-effectively than they would by buying the hardware themselves. However, there is a slight drawback with serverless possibly taking some time to start. This is called the cold-start problem. Cold-start time can be minimized by the provider by, for example, predicting the load from users [21] but with cold-start there is always some delay that can easily last for several minutes. The typical solution for this is to either use a hot instance that is constantly running and expecting requests, but that removes the point of using a serverless architecture and, therefore, increases costs significantly. A hybrid solution between these is pre-warming the resources, which could mean, for example, predicting the needed load and starting instances beforehand or if the schedule is known, starting

the instances before the scheduled time [22]. While time series forecasting tasks typically are not time-critical, there are some cases that require real-time forecasting like in [23] where solar irradiance is forecasted for a virtual power plant based on images of the sky. Another example of this kind of real-time forecasting is predicting the prices of aFRR Energy market in Finland that has 15 minutes intraday market [24]. This kind of forecasting will be addressed in the experimentation part of this thesis in chapter 6.

One important category of forecasting techniques are machine learning models. Cloud providers also offer SaaS solutions that are specifically meant for machine learning tasks. These include, for example, Google's Vertex AI platform [25] and Microsoft's Azure Machine Learning [26]. If machine learning is used to make the time series forecasting, these can be useful. By leveraging the cloud computing solutions instead of developing locally with on-premises systems, it is possible to easily achieve scalability, good availability, easy deployment, easy management, and speed, especially if using services that are available in the cloud either from the cloud provider or as an integration.

A benefit of cloud services in time series forecasting tasks is also the cheap availability of expensive resources for limited use. This allows small companies with a low amount of equity to experiment with time series forecasting models and evaluating if it is sensible to either invest in their own equipment or continue using the cloud.

A drawback with cloud services for time series forecasting tasks is the limited amount of available equipment such as GPUs. The two largest server GPU manufacturers are Nvidia and AMD, but there are differences between the GPUs they produce [27]. These differences can cause strict demands for the brand or even type of GPU that is used to ensure proper functionality and, for example, compatibility with different time series forecasting libraries. The issue is that cloud providers often have GPUs from only one brand. For example, GCP has only Nvidia GPUs [28] and a Finnish supercomputer resource provider LUMI has only AMD GPUs for time series forecasting use [29]. Azure does have GPUs from both Nvidia and AMD but only a few models [30]. This means that if the user develops the system for a certain combination of a GPU model and a cloud provider, the vendor lock-in becomes quite strong. To avoid vendor lock-in the user can develop code that is compatible with multiple types of GPUs. This, however, might limit library options, for example, or cause additional work. Also, in a situation where the cloud provider decides to change the available GPUs the user might need to do some additional work.

In conclusion, the benefits of running time series forecasting in the cloud include scalability, easy management, ease of deployment and easy integration if using services that are natively available in the cloud. The drawback is however a strong possibility of vendor lock-in which can cause additional work and costs.

## 2.3 Hybrid Cloud Patterns

Hybrid cloud means combining either multiple public clouds or a combination of public and private clouds to form a single system [11]. In practice, this could mean, for example, using two public cloud providers like GCP and Azure together or combining

a public cloud with an on-premise system. The on-premise system can be a private cloud itself or even just a single machine. The system can even be more complex and combine for example multiple public and private clouds.

The benefits of a hybrid cloud system can include cost saving, reliability, higher availability, prevention of total vendor lock-in, increased security by keeping the most critical data in a private cloud, and the ability to combine legacy systems with cloud systems when the legacy systems cannot be run in cloud but can be integrated into a hybrid system. If combining two public clouds, a benefit over using a single cloud is also a wider variety of native SaaS solutions [31]. If using a combination of public and private cloud, an additional benefit is the possibility to secure most critical information to the private cloud and information or computing power that needs high availability to the public cloud [31].

There are some existing hybrid cloud patterns that can be used when designing hybrid cloud systems. According to [32], these patterns between the private and public cloud can be divided into four patterns:

**Table 2:** Emerging Hybrid Cloud Architecture Patterns

Pattern	Description
Static Placement	Workloads are statically bound to either private or public cloud environments. They communicate through services or APIs but workloads are not shared. This is common when integrating legacy systems to cloud [32].
Assisted Replication	Workloads can be replicated from private to public clouds or from public to private cloud but not both ways. Then the component or workload can be run in either or both private and public cloud. The replication usually requires human intervention [32].
Automigration	Code or workloads can be moved between clouds via manual or automated processes both ways. Well-defined interfaces are required for this [32].
Dynamic Migration	Workloads can be moved completely dynamically between clouds, as if both environments shared the same virtual operating system [32].

For legacy systems that cannot be moved to public cloud for either practical, security or some other reason, the static placement pattern is the most common. It is also the most typical hybrid cloud approach [32]. Assisted replication is used in the experiment in chapter 6 because the workload needs to be run in both private and public cloud.

For building the hybrid cloud, it is very useful to have some kind of middleware to ease creating and managing the system. In [33] a framework for handling this is proposed. If the system requirements are quite simple and the shared workload is quite small, it is possible to use an orchestration tool to handle the resource sharing

between clouds. However, it is then important to ensure that the orchestration tool deployment does not prevent the system from benefiting from the hybrid cloud. For example, when using two public clouds and deploying the orchestration tool to only one, the system is dependent on the availability of the cloud with the orchestration tool even though the other cloud would be fine. However, it is possible to deploy the orchestration tool in a way that it is not dependent on the availability of one cloud. In some instances, it is also possible that the high availability is not a concern and the simple way of, for example, deploying the orchestration tool to a private cloud and utilizing public cloud resources with it is sufficient.

## 2.4 Hybrid Cloud Combining Private and Public Clouds: Trade-offs and Design

One common way to implement a hybrid cloud is to combine a private on-premises cloud and a public cloud. This kind of combination can take advantage of both. Some of these benefits are unique to this combination due to the different characteristics of public and private clouds. Private clouds, for example, can be trusted better than public clouds, but public clouds are very easy to scale compared to private on-premises clouds. Table 3 presents some of the main benefits that this kind of hybrid system has compared to other hybrid clouds.

**Table 3:** Benefits of a hybrid cloud combining private and public clouds compared to other hybrid architectures

Benefit	Example
Cost optimization while maintaining scalability, assuming the private cloud cost is independent of load	Running base workloads of data processing on the private cloud to avoid variable costs, while using public cloud resources for peak demand [7].
Secure storage of sensitive data in the private cloud and scalable highly available storage in public cloud for privacy or regulatory compliance	Storing patient records in the private cloud for compliance, while executing scalable data analytics in the public cloud [34].
Resilience while keeping normal operation in private cloud	Using public cloud as an encrypted backup for private cloud storage for sensitive data [35].
Improved fault tolerance to environment related issues	Using private and public cloud for scalability in normal situation but changing to only private cloud if malicious behavior is detected in the public cloud [36].

The main trade-offs of this kind of hybrid compared to other architectures are that this requires both expensive equipment for the on-premises systems and a more complex setup than just using the local or public cloud. Due to having the local infrastructure, it also requires maintenance which counters one of the benefits of

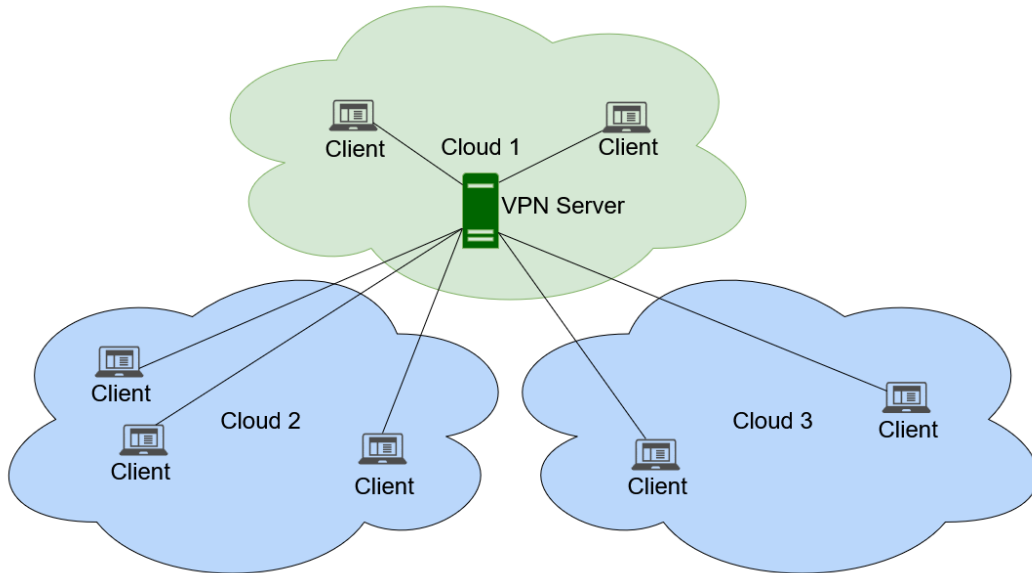
using public cloud which is the use of managed services. So, this kind of hybrid architecture is beneficial to use only in certain scenarios which usually require either high availability with low cost in the long run or a secure way of handling sensitive data while being easily scalable.

When designing the hybrid cloud that combines both private and public cloud, there are some additional things to consider that are not present in a single cloud system. The first of them is designing the location of resources and workloads. The simplest of these is that if the resource uses sensitive data, it must be placed in the private cloud. Resources that need high scaling abilities are probably better to deploy to the public cloud or to both the on-premises private cloud and the public cloud. One strategy for this is cloud bursting, where on-premises workloads are offloaded to the public cloud during peak demand. Under normal conditions, the system can run on cost-efficient on-premises infrastructure without needing investments in high-capacity hardware and during peak periods the capacity of the system can be increased cost-effectively with the public cloud [37]. If the resources need high availability, it is best to deploy them either to the public cloud and distribute them to multiple physical locations or use a combination of public cloud and private cloud where they back each other up. If massive amounts of data are handled, it is also beneficial to store the data in the same cloud where it is processed to minimize transfer latency and costs.

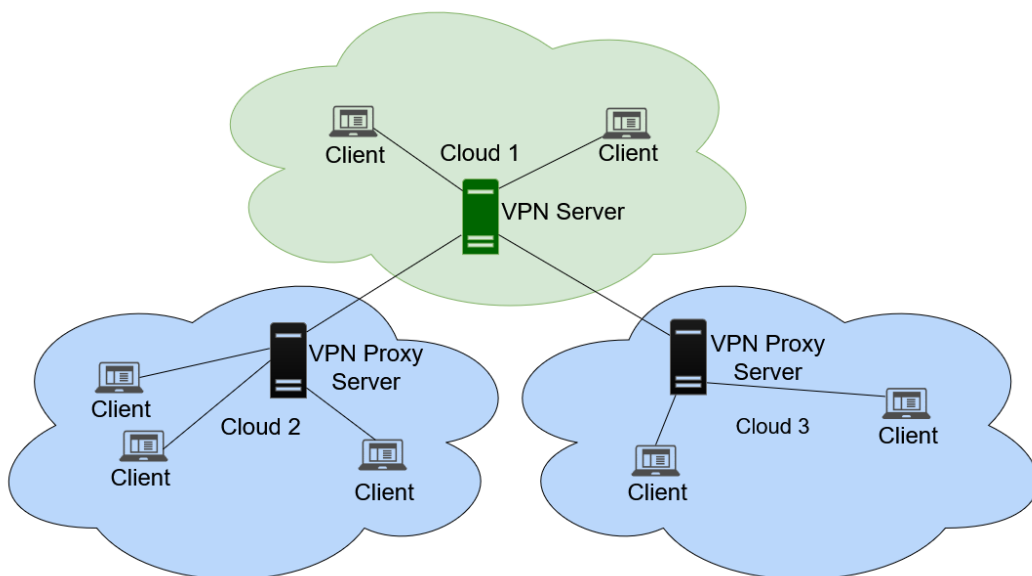
Another aspect to consider on top of resource and workload placement is the communication between the clouds. If for example using an orchestration tool to manage the resources in all of the clouds, it is important that the tool has access to all of the resources. A common way to implement this is using a virtual private network (VPN) between the clouds. This topology is presented in figure 2. However, with this approach there are some performance issues in latency and reliability. The main issue in this comes from the VPN server location because it cannot be in both on-premises and public cloud. As a solution for this [38] proposes an architecture where a main VPN server is deployed to one of the clouds and a VPN proxy server is placed in all the clouds. This topology is presented in figure 3. In this way, communication between the clouds goes through the main VPN server, but the internal communication within each cloud does not require resources from other clouds. In [38] this increased the network performance drastically in communication that went only through a proxy server.

There are also some smaller things to consider with a hybrid cloud system, such as identity and access management across the clouds, unified monitoring, version alignment for applications, and dependencies between the clouds.

When evaluating possible technologies and architectures, it is important to balance the complexity of the system and the required maintenance with the achieved results, such as cost savings or reliability. With this kind of custom solution, it is easy to make the system too complex, resulting in a non-optimal solution. It is also beneficial to try to avoid vendor lock-in to keep the system flexible and independent of cloud providers.



**Figure 2:** Directly connected VPN topology adapted from [38].



**Figure 3:** VPN topology based on proxy servers adapted from [38].

## 3 Workflow Orchestration

Workflow orchestration is used to manage software systems. Compared to simple automation, orchestration typically manages more complex systems. Automation is meant to automate the execution of single tasks, while orchestration is used to coordinate and manage complex workflows. For complex systems that have multiple tasks that depend on each other, it is important to make the orchestration centralized so that the management does not become a burden. This chapter discusses workflow orchestration, trade-offs, and how the orchestration can be used with hybrid cloud systems.

### 3.1 Orchestration in Software Systems

In software systems, orchestration tools are used to automate, schedule, and coordinate tasks based on different conditions, monitor tasks, and handle retries. Nothing that the orchestration tools do would be impossible for the specific application, but orchestration tools streamline the development process and allow developers to concentrate on the essential parts of the product.

Orchestration tools are a fundamental part of DevOps. DevOps is a trend of using software engineering tactics that bring software development and IT operations closer by reducing the time and effort between them [39]. In practice, DevOps aims to shorten the time between writing code and releasing it and to decrease the number of manual steps in this process.

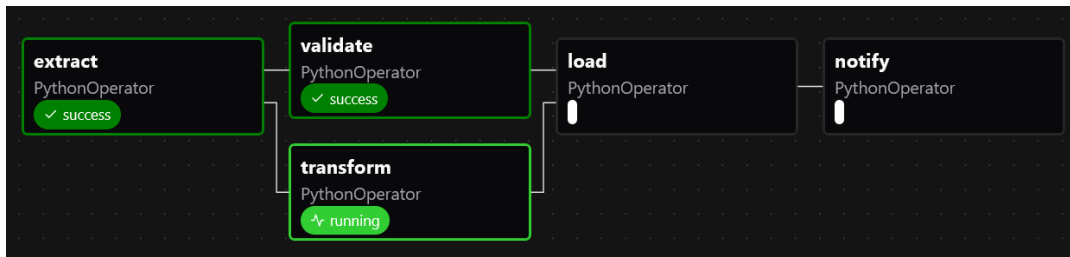
To achieve these goals, DevOps uses continuous integration (CI) and continuous delivery (CD). One main practical meaning of this is to develop small but frequent changes that are automatically prepared for deployment. This automatic process is often implemented with tools that use CI/CD pipelines [40] that automatically build, test, and deploy changes.

CI/CD pipelines can be seen as orchestration tools themselves, but other orchestration tools can be used to separate the changing software from the core functionality of the system. This way, the triggering of a task and workflow dependencies do not need to be manually adjusted when deploying updated software. This is essential to reduce the time spent on orchestrating tasks in the system with each change.

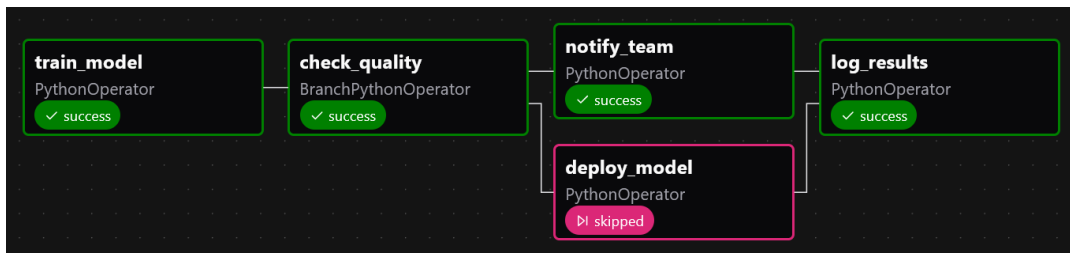
For orchestration tools, it is typical to model the workflows as directed acyclic graphs (DAGs) or similar structures. With DAGs it is easy to ensure that each task runs only when the prerequisites for the task are completed. Examples of DAGs from one orchestration tool called Apache Airflow can be seen in figures 4 and 5. There are also tools that do not require the tasks to follow a DAG format, but this is less common.

### 3.2 Orchestration Tools

There is a wide variety of different orchestration tools for different purposes. Popular open source industry standard tools include Apache Airflow and Kubernetes. Apache Airflow is a workflow orchestration tool used for managing task and data workflows. Kubernetes orchestrates infrastructure and service deployment where the applications



**Figure 4:** A screenshot from Apache Airflow user interface showing an example DAG that handles a data processing workflow. The workflow is still running here.



**Figure 5:** Another screenshot from Apache Airflow user interface showing an example DAG that trains a machine learning model and either deploys or sends an error message to the team depending on the model performance. The workflow has finished and the data quality has not been good enough so a notification to the team has been sent.

that it manages are containerized. Containerization means light virtualization often combined with standardized packaging [41]. One popular example is the open source Docker Engine that can be used to containerize and run applications in a standardized way for easy deployment in any environment where the engine is running.

Orchestration tools can be either self-managed or managed. In the self-managed model, the developer is responsible for maintaining the software and environment. In the managed model, a third party maintains the deployment, but the developer still uses and configures the tool. The drawbacks of managed tools are their cost and potential privacy or security issues if the tool has access to sensitive code or data.

If using a managed orchestration tool is selected, it is possible to use tools that are native to the cloud that is being used. These tools include for example GCP Cloud Composer, Azure Data Factory and AWS Step Functions. It is also possible that some third party or open source tools such as Apache Airflow are offered as a managed service. Examples of this are Amazon Managed Workflows for Apache Airflow (MWAA) and Apache Airflow on Astro. Cloud Composer is a hybrid between these because it is a managed Airflow that is native to GCP.

The benefit of using cloud native orchestration tools is that they often have easy integrations to other services in the cloud and are probably more cost-effective than other solutions such as hosting an open source solution in the cloud. However, there is a risk of vendor lock-in since these solutions often become a central part of the system. The risk is especially high if the system uses some features that are unique to the native solution.

While orchestration tools are an excellent help for workflow management, it is often useful to combine orchestration with other tools. [42] compares three architectures in an AWS-based cloud computing solution: a pure managed workflow for Apache Airflow (MWAA), a MWAA - AWS Lambda hybrid and a MWAA - AWS Glue integration for an ETL (extract, transform, load) workflow in a supply chain management system. AWS Lambda is a FaaS that can be utilized here as an easily manageable way to run tasks. AWS Glue is a PaaS solution that can similarly be used for running tasks with low maintenance, especially here by using ETL integrations. [42] concludes that for the supply chain system that was studied, the integration of MWAA with AWS Glue was the most beneficial architecture despite its complex initial setup due to ETL integrations that Glue has. Generally, the best architecture depends on the application, but it is clear that combining multiple tools according to the task is beneficial if the additional setup complexity is acceptable. If the alternatives are either using other tools such as AWS Lambda or creating dedicated software for the purpose, it is often lower maintenance to use other tools despite the added complexity to the system.

One common tool to combine with orchestration is a monitoring tool. Having the logs and status separated from the orchestration tool allows us to detect issues in the orchestration tool as well. Some common monitoring tools are, for example, Datadog and New Relic. If wanted, it is possible to even use the monitoring results to adjust the functionality of the system. [43] proposes an architecture, where the monitoring tool orchestrates and scales native cloud applications using machine learning. The ML model predicts the load on the system, and the resources are scaled accordingly. This is a very advanced system, but useful in scenarios where the scaling of the resources noticeably affects the costs.

There are many ways to build hybrid cloud system architectures, each with its own strengths and weaknesses. This is why it is important to select the architecture and components according to the application to achieve good performance without too much setup and maintenance overhead.

## 4 Redundancy

One of the common reasons for creating a hybrid cloud architecture is the need to reduce downtime and ensure critical workload functionality. One way to achieve this is system redundancy. This chapter discusses system redundancy, focusing on diverse redundancy in a hybrid cloud system.

### 4.1 Redundancy Overview

In the context of cloud systems redundancy means that the services are duplicated so that if one fails, another instance can take over. In this way, the system availability can be increased and the possibility of system failures decreased. Redundancy can be used in single cloud systems by using replication of system components such as VMs and deploying them to different servers [44]. The servers can even be selected to be in different physical locations. Similarly, data can be duplicated to different locations to avoid data loss. If implemented in a smart way, this kind of system has the additional benefit of having lower latency times by utilizing the resources that are closest to the system using them. The network can also be made redundant by having multiple routes for traffic and having multiple devices to handle this. This is especially important when using a VPN or similar solution instead of the public internet for the connections between the cloud components.

There are typically already existing easy ways to do the replication for services, data, and network when using native public cloud solutions. This means that typically increased maintenance is not a large issue with cloud redundancy. However, the issues are vendor lock-in, dependency on that vendor if something goes wrong, and increased cost [45]. Deploying a system that utilizes native solutions and is replicated within a single cloud can be difficult to move to another cloud. Also, when there are issues with the cloud provider, it is possible that those issues affect all or most of the servers they have. Replication also increases costs roughly in proportion to the number of replicas. An issue with identical replications is also that they all have the same weaknesses or bugs, so if one of them fails due to software issues or cyberattacks, it is likely that the other instances will fail as well. This is where diverse redundancy with hybrid cloud architecture can help.

### 4.2 Diverse Redundancy in a Hybrid Cloud Architecture

Diverse redundancy means that the redundant components are not duplicated but slightly modified so that duplications do not fail for the same reason. Modifications can be made to the environment or code, but the main functionality should remain the same so that the different instances can be used interchangeably.

An example of diverse redundancy is to have redundancy systems such as VMs run on different operating systems (OS) to increase the probability that the system will survive a cyberattack [46]. Cyberattacks are often targeted for a specific OS, so having multiple operating systems reduces the risk that the same attack works on all

virtual machines. At the same time, this also eliminates the risk of OS-specific bugs in the system.

Another way to increase diverse redundancy is to modify the software for different instances. It is possible to for example use a different algorithm to get the same or similar result, use slightly different parameters in a prediction model or write the program with a different library or even different language. However, some of these methods, such as changing the programming language, are often too heavy for the application.

A very powerful way to increase the availability of the system or data with redundancy is a hybrid cloud architecture that also avoids the vendor lock-in problem [47].

Hybrid cloud architecture with redundancy can also be used for data protection. [48] proposes an architecture in which sensitive data is first encrypted in a trusted secure secret sharing proxy that is in a private server or cloud and split to shares by applying the Krawczyk secret sharing scheme. This scheme works in a way that by having enough but not necessarily all shares allows reconstructing the original data but with too few shares, this is not possible. After splitting to shares, two hash message authentication codes (HMACs) are bundled with each share. The shares are distributed to different clouds so that in a single place there are fewer shares than is required to reconstruct the original data. This way the cloud providers or malicious actors have no way to access the data by accessing just the share or shares in a single cloud. The HMACs ensure that data modifications by the cloud provider or malicious actor are detected. This architecture allows the user to utilize public clouds to safely store sensitive data without having a single point that would expose the data when compromised.

With a hybrid cloud architecture, it is important to carefully design the system in a way that there are no critical single points of failure that would invalidate the efforts to create a high availability system with redundancy. The deployment of the orchestration tool is one potential single point of failure.

### **4.3 Orchestration Design in a Redundant Hybrid Cloud Architecture**

In a redundant hybrid cloud system, the orchestration plays a crucial role because the components need to work together. To achieve high availability, the orchestration is responsible for either running the backup when the primary system fails or selecting the system to use when there are redundant systems available.

A common approach that ensures high availability in distributed deployments is the use of redundancy for fault tolerance and replication for load balancing [49]. When redundancy is used in a hybrid cloud system to achieve high availability, it is also important to design the service orchestration in a reliable way. Otherwise, the orchestration may be a single point of failure which would nullify the redundancy.

The orchestration tools often have multiple crucial components that are needed to operate for the whole system to function at all. One of them is Apache Airflow. It

was released in 2015 and has since become an open source industry standard solution for orchestration. It is used here as an example of an orchestration tool. For Apache Airflow to function, these are the components required for a minimal installation [50]:

- Scheduler(s) for scheduling the directed acyclic graph workflows (DAGs)
- Metadata database(s) to store the state of workflows and tasks
- DAG processor(s) to parse and serialize the DAG files to the metadata database
- A directory for DAGs to be read from by the DAG processor
- API server(s) to present the user interface

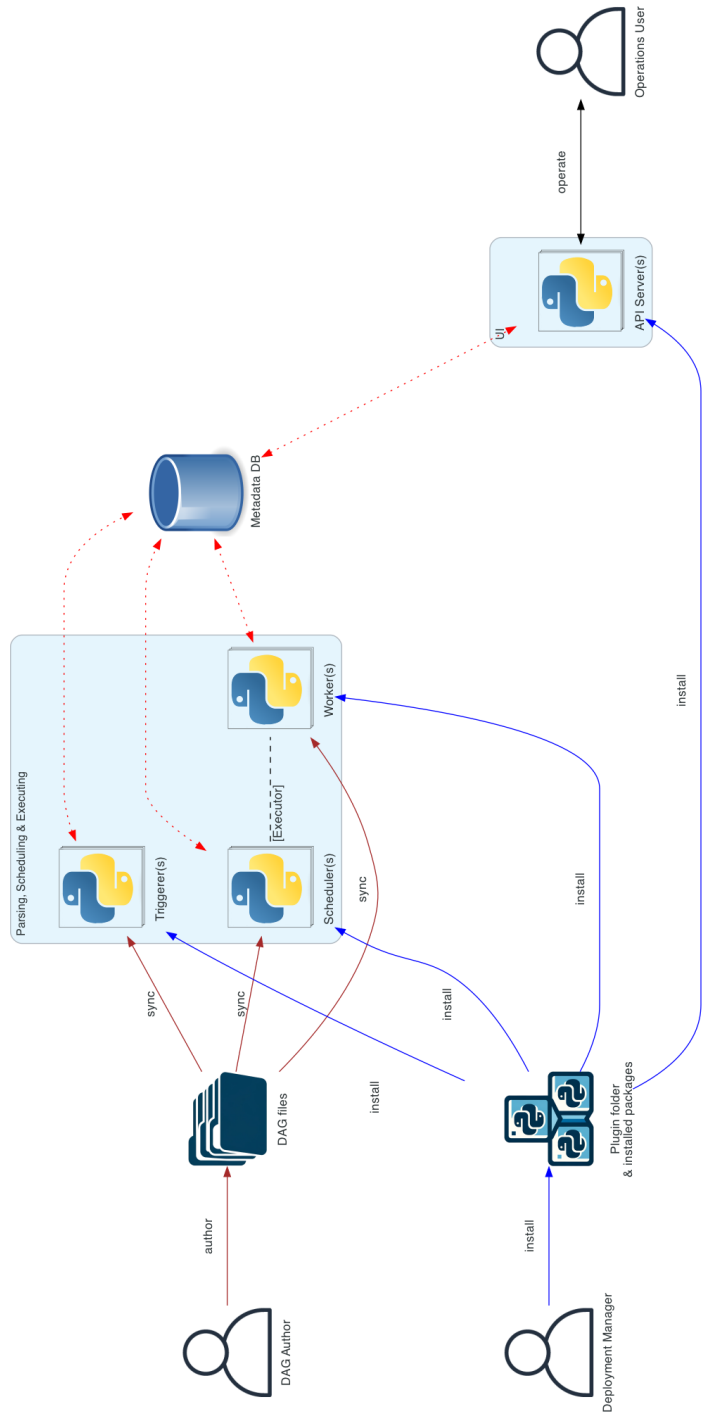
There are also some optional additional components [50]:

- Worker(s) to execute the task separately from the scheduler
- Plugin(s) to extend the functionality of the orchestrator
- Triggerer(s) to execute deferred tasks

For deploying Airflow in a redundant highly available way, the components must be deployed in a distributed way. Figure 6 shows how the components interact in a distributed deployment.

In Apache Airflow most components except for the API server support replication, meaning that multiple instances of the component can work together in parallel. The API server requires a redundant standby server to achieve high availability in case the primary server fails or undergoes maintenance. For other components the redundancy is achieved through replication. For workers, schedulers and other processing components this means running multiple concurrent instances balancing the load of each other. For the databases and DAG folder the replicated deployment is achieved by syncing the data between multiple instances. By placing the replicas on different servers in different locations, the availability and resiliency of the system is increased even more.

By deploying the orchestrator in a distributed and resilient way using redundancy and replication, the orchestration installation is no longer a single point of failure for the whole system.



**Figure 6:** A distributed Apache Airflow deployment [50].

## 5 Time Series Forecasting with Autoregressive Integrated Moving Average Models

Autoregressive integrated moving average models are a common choice for time series forecasting because they combine three key elements for forecasting: taking into account past values with autoregression, accounting for trends with integration and correcting for past prediction errors with moving average. This chapter discusses these ARIMA-models and two ARIMA-based models SARIMA and SARIMAX.

### 5.1 Introduction to Autoregressive Integrated Moving Average Models

There are multiple variations of autoregressive models. This chapter introduces the most common ones starting with the simple autoregressive moving average model and building up to more complex models, ending with seasonal autoregressive integrated moving average model with exogenous variables.

#### 5.1.1 Autoregressive Moving Average Model

Autoregressive (AR) models are a common choice for time series forecasting for their ability to take into account past data of the series that is being predicted. In practice, the autoregression takes into account past observations of the series and tries to predict based on that information.

Using only past values is rarely sufficient for accurate prediction. Adding moving average (MA) to the model takes into account past prediction errors resulting in an ARMA model. In practice, the model takes a fixed window of past prediction errors, multiplies them with suitable coefficients calculated from the past data and averages the obtained values to a single value. This value is taken into account in the prediction. The resulting ARMA model is presented in 1 adapted from [51]:

$$X_t = c + \varepsilon_t + \sum_{i=1}^p \varphi_i B^i X_t + \sum_{j=1}^q \theta_j B^j \varepsilon_t \quad (1)$$

$$B^k X_t = X_{t-k} \quad (2)$$

where  $c$  is a constant,  $\varepsilon_t$  is the residual at time  $t$ ,  $p$  is the order of the autoregressive part,  $\varphi_i$  are the autoregressive parameters obtained based on the past data,  $B^i$  is the backshift operator representing previous elements of  $X$ ,  $q$  is the order of moving average part,  $\theta_i$  are the moving average parameters obtained from the past data and  $\varepsilon_t$  are normal random variables. The formula works for all  $t > \max(p, q)$  because  $p$  and  $q$  determine how many past elements are used to make the prediction.

#### 5.1.2 Autoregressive Integrated Moving Average Model

The combination of autoregression and moving average is powerful for data with constant mean and variance and with mainly short-term dependence. However, many

time series have a trend that drifts the data to some direction over time and the ARMA-model does not work well with data that has trends. To take this into account, an integration term can be added. With the integration term the time series can be made stationary by removing the previous value or weighted average of multiple previous values from each data point. This way, the trends are removed from the series. To get the predictions to the correct magnitude, the last known value is added to the prediction. This ARIMA model is presented in formula 3 adapted from [51]:

$$\nabla^d X_t = c + \varepsilon_t + \left( \sum_{i=1}^p \varphi_i B^i \nabla^d X_t \right) + \left( \sum_{j=1}^q \theta_j B^j \varepsilon_t \right) \quad (3)$$

$$\nabla^d X_t = (1 - B)^d X_t \quad (4)$$

where  $\nabla^d X_t$  is the differenced value of  $X$ ,  $d$  is the order of differencing,  $c$  is a constant,  $\varepsilon_t$  is the residual at time  $t$ ,  $p$  is the order of the autoregressive part,  $\varphi_i$  are the autoregressive parameters obtained based on the past data,  $B^i$  is the backshift operator representing previous elements of  $X$ ,  $q$  is the order of moving average part,  $\theta_i$  are the moving average parameters obtained from the past data and  $\varepsilon_t$  are normal random variables.  $X_t$  can be calculated with formula 5 adapted from [51]:

$$X_t = \nabla^d X_t + \sum_{i=0}^{d-1} \nabla^i B X_t \quad (5)$$

### 5.1.3 Seasonal Autoregressive Integrated Moving Average Model

With integration term the achieved ARIMA-model can already be used for many scenarios, but it is still missing ability to predict seasonality in a time series. Seasonality is present in many real-world scenarios like in weather or traffic data. To account for seasonality in the data, a seasonal component can be added to the model. The seasonal component works in a similar way as the integration component, but instead of removing the consecutive values from the data, the season length is fed to the model and the corresponding value from previous season or a weighted average of values from multiple previous seasons is subtracted from each data point. To get the final predictions, the last known value is added to the predicted value. For example, daily traffic predictions can account for lower traffic on Sundays compared to weekdays. The resulting SARIMA model is presented in formula 6 adapted from [52]:

$$\varphi_p(B) \Phi_P(B^s) \nabla^d \nabla_s^D X_t = \theta_q(B) \Theta_Q(B^s) \varepsilon_t \quad (6)$$

where:

- $X_t$  is the variable to be forecasted
- $\varphi_p(B) = 1 - \sum_{i=1}^p \varphi_i B^i$  is the regular AR
- $\theta_q(B) = 1 - \sum_{i=1}^q \theta_i B^i$  is the regular MA
- $\Phi_P(B^s) = 1 - \sum_{i=1}^P \Phi_i B^{s,i}$  is the seasonal AR

- $\Theta_Q(B^s) = 1 - \sum_{i=1}^Q \Theta_i B^{s,i}$  is the seasonal MA
- $\nabla^d = (1 - B)^d$  is the regular differencing
- $\nabla_s^D = (1 - B^s)^D$  is the seasonal differencing
- $\varepsilon_t$  is the residual at time  $t$

A drawback in SARIMA is that it can only take into account one seasonality in the data. [53] proposes a solution for this where multiple SARIMA-models are used together to take into account multiple seasonal variations. This way for example the daily total traffic prediction can take into account the time of the year on top of the weekday.

#### 5.1.4 Seasonal Autoregressive Integrated Moving Average Model with Exogenous Variables

SARIMA-model is strong in predicting time series data based on the series itself. However, real-world data is often dependent on other data too. To take other datasets into account, it is possible to add exogenous variables to the model. The idea is that the model can utilize the information from other datasets to forecast the original time series. The resulting SARIMAX model is presented in formula 7 adapted from [52]:

$$\varphi_p(B)\Phi_P(B^s)\nabla^d\nabla_s^D X_t = \beta_k x_{k,t}' + \theta_q(B)\Theta_Q(B^s)\varepsilon_t \quad (7)$$

where other elements are the same as in formula 6 but  $\beta_k$  is the coefficient of the  $k$ :th exogenous input variable and  $x_{k,t}$  is the vector including the  $k$ :th exogenous variable at time  $t$ .

When adding the exogenous variables, it is important to use domain knowledge to add only variables that actually influence the target series. Adding variables with a too low influence to the target series has a negative impact on the prediction accuracy.

## 5.2 Use Cases of Autoregressive Integrated Moving Average Models

The selection of the autoregressive model depends on the use case. Despite the SARIMAX model being more capable than other autoregressive models, it is not a good idea to always use the SARIMAX model due to possible overfitting. Overfitting means that the model fits to the training data in a way that the model learns not only the trends in the data but also random noise that will only make the results worse when predicting. An example of this is using the SARIMAX model with an exogenous variable that is not related to the time series. Including the exogenous variable will add noise to the model and the model will not be able to predict the original series in the same quality as without the additional noise. The opposite problem of underfitting means that the model is too simple and is not able to learn the characteristics of the data. An example of this is using the ARIMA model to a dataset that is clearly seasonal

such as the traffic prediction mentioned above. Matching the model and the data correctly will achieve the best results without unnecessary over or underfitting. This chapter will present some use cases for ARMA, ARIMA, SARIMA and SARIMAX models.

The ARMA model is used in [54] for solar forecasting to help power system operators make informed decisions on the electricity market. However, the ARMA is often combined with other models due to the simplicity of the ARMA-model to capture different characteristics of the data. For example, [55] combines the ARMA model with another autoregressive model called autoregressive conditional heteroskedasticity (GARCH) to forecast electricity future prices. Formula 8 presents the combination of ARMA and GARCH models in [55]:

$$r_t = \mu_t + e_t \quad (\mu_t \sim ARMA(m, s)) \quad (8)$$

$$e_t = \sigma_t \varepsilon_t \quad (9)$$

$$\sigma_t^2 = \omega + \sum_{i=1}^p \alpha_i e_{t-i}^2 + \sum_{j=1}^q \beta_j \sigma_{t-j}^2 \quad (10)$$

where  $r_t$  denotes the mean equation that is the predicted value at time  $t$ ,  $\mu_t$  is the mean value from the  $ARMA(m, s)$  process, and  $e_t$  is the residual variance. The residual variance is the result of conditional variance  $\sigma_t$  of the series multiplied by  $\varepsilon_t$  that are random variables with mean 0 and variance 1.

The bottom row is the GARCH model where  $\sigma_t^2$  is the predicted variance,  $\omega$  is the baseline variance from the series,  $\sum_{i=1}^p \alpha_i e_{t-i}^2$  is the effect of previous residual errors in the time series and  $\sum_{j=1}^q \beta_j \sigma_{t-j}^2$  is the effect of previous variances.

The GARCH model is good at predicting variance in the data so combining it with ARMA that is good at predicting the mean of the data results in a good model especially for financial data that has changing variability.

While ARMA is a strong model for static predictions, having some kind of trend in a time series is very common and ARMA cannot handle that well. When the data includes a trend, it is better to use ARIMA. The trends might not be self-evident but are often present in real-world time series. For example [56] uses ARIMA for predicting metro station traffic. The traffic has an upwards trend and having the integration part in the model helps to achieve relatively good results. The ARIMA model can be combined with other methods to increase the performance. In [57] ARIMA is combined with GARCH to make short term wind power predictions.

Seasonality is often present in time series even when the series does not obviously follow natural or human-driven patterns. [58] forecasts cloud resource demand on monthly basis using SARIMA model with 12 month seasonality and achieves relatively good results. This helps cloud providers to utilize the resources more effectively and prepare for demand peaks. [59] compared SARIMA with GARCH to predict Indian electricity market prices and concluded that SARIMA clearly outperformed GARCH.

Combining these methods was not considered, but would make interesting further work.

SARIMA is a powerful method for many real-world datasets, but the systems behind time series behavior are often more complex and cannot be inferred from a single time series alone. SARIMAX enhances the SARIMA model by adding exogenous variables to it which makes it possible to predict complex systems with better quality than with SARIMA. In [10] SARIMAX with some economy related exogenous variables is used to predict consumer price index (CPI). CPI is a result of a complex system so exogenous variables have potential to help the model. [10] concludes that the SARIMAX outperformed other traditional forecasting models including ARIMA. Another use for SARIMAX is in [52] where solar power generation forecasting with SARIMAX uses solar radiation forecast as an exogenous variable. The result is compared to a SARIMA model and the conclusion is that in day-ahead predictions the SARIMAX in most cases outperforms the SARIMA model, but in intraday predictions and when the weather conditions stay similar for a long time, the SARIMA is better. This shows the importance of choosing the exogenous variables carefully to avoid introducing noise.

As can be seen, there are many autoregressive models with different strengths. There is no single model that would work in every scenario, but the model needs to be selected according to the data to achieve the best performance. It is also possible that the best performance is achieved by combining two different models with different strengths.

## 6 System Implementation and Use Case

This chapter discusses an experiment that forecasts electricity reserve market prices with redundancy. The goal is to demonstrate how this kind of architecture could be used to achieve high availability in electricity market forecasting that has real-time requirements. It also demonstrates how an autoregressive model can be used for the forecasting.

### 6.1 Use Case Overview

In the experiment, the goal is to forecast frequency containment reserve for normal operation (FCR-N) prices in the Finnish electricity reserve day-ahead market with high availability. The forecasts are done using an autoregressive model and the high availability is ensured using diverse redundancy.

In practice, the system is built using Airflow orchestrator tool. Airflow is an industry standard tool for orchestration and is therefore chosen for this experiment. In Airflow, there is a DAG that runs the forecasting in both local and cloud environment at the same time. If one of the tasks fails, the results from the other environment can be used.

The forecasts are done using an autoregressive forecasting model. The components and tuning of that model is presented in chapters 6.3 and 6.4. Overall system implementation is presented in chapter 6.5. Before these, chapter 6.2 provides some background of the electricity market and why this kind of redundant system could be useful.

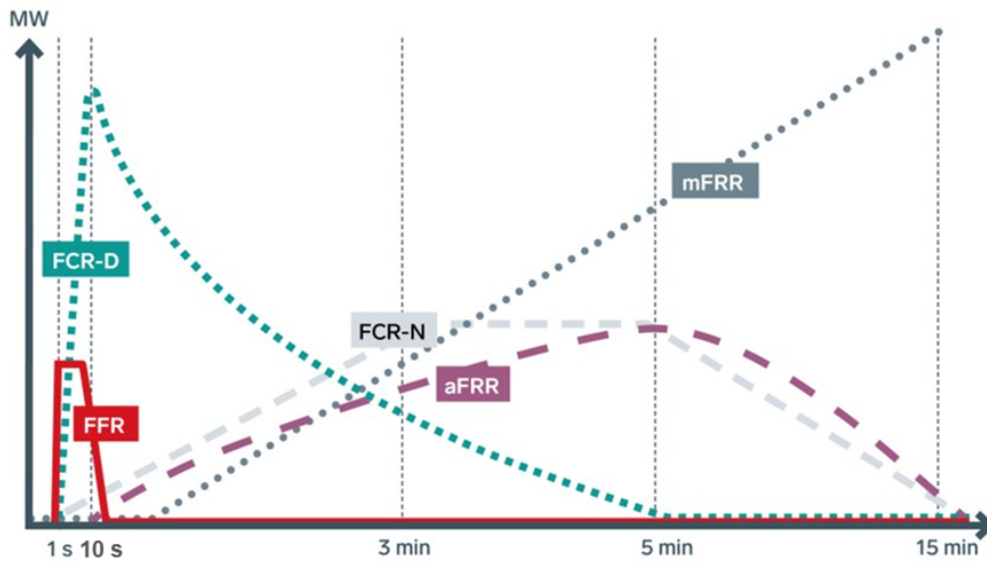
### 6.2 Electricity Market Background

For the operation of electricity system it is necessary to keep the system frequency stable at a certain frequency. In Finland, this frequency is 50 Hz, but the system is considered to be within the normal operating range as long as it stays between 49,9 and 50,1 Hz. Significant deviations from the nominal value may lead to damages in the components of the power system, equipment and facilities that are connected to the power grid and at worst to widespread power outages [60]. The frequency depends on the relationship between electricity production and consumption. When there is more production than consumption, the frequency is too high, and when there is more consumption than production, the frequency is too low.

The system is kept between these values using different electricity markets to balance the production and consumption. During the previous day, the wholesale price of electricity is formed according to available production and required consumption. In Nordic countries, this happens through an auction in the Nord Pool Day-ahead market. Both of these flex according to the electricity price so the price helps to balance supply and demand to some extent. However, due to system failures, forecasting errors and variation in for example weather and consumer behavior, it is not possible to balance the production and consumption just with the price. Different reserve markets are used for fine frequency adjustments. This need is strongly increasing, because wind

and solar power increase the need for balancing for two reasons. The first reason is the unpredictability of wind and cloud coverage, which makes it difficult to keep electricity production steady with these production methods. The second reason is the lack of inertia in these production methods. In traditional energy production forms there has been a large spinning mass such as a generator that has continued to spin for a while even if the electricity production has unexpectedly stopped. There is no such inertia in wind and solar power, making the electricity grid more prone to failure in the event of an issue in wind or solar power facilities. These risks are being tackled by increasing the amount of electricity reserves.

There are two types of reserve markets both including multiple different variations. The first type is energy market where energy is sold and bought according to the need. The cost is based on the amount of energy transferred. The second type is capacity market where different electricity capacities are reserved to be used according to the need. The capacities react to the grid frequency depending on the agreed terms. In Finland, energy market variations include manual frequency restoration reserve (mFRR) [61] and automatic frequency restoration reserve (aFRR) energy markets [62]. The capacity markets include manual frequency restoration reserve (mFRR), automatic frequency restoration reserve (aFRR), frequency containment reserve for disturbances upwards (FRC-D up), and disturbances downwards (FCR-D down), frequency containment reserve for normal operation (FCR-N), and fast frequency reserve (FFR) markets [63]. Each of these has their own characteristics and is meant for different purposes. Figure 1 briefly presents the reserve markets and their purposes. Figure 7 demonstrates how the reserve markets can together react to a sudden failure in the grid by utilizing the different characteristics of each reserve market.



**Figure 7:** Demonstration of reserve markets working together after an accident where a large power plant has suddenly failed at 0 seconds [63].

The reserve market prices are based on need and supply. The transmission system operator (TSO) determines the need for different reserves and the reserve providers

offer their capacities with a certain price. The offers are ordered by price and the offers are selected starting from the cheapest until the required capacity is achieved. The price is then determined to be the highest required offer for all offers that were equal or cheaper than this limit price for that time period.

If the capacity can be used as a reserve constantly, it is easy to just determine the price for using the capacity in the electricity market and then always offer that. However, many capacities cannot be used constantly which makes it important to concentrate the offers for hours with most demand. This is good for the TSO because the required capacity is more easily available and also good for the capacity provider because they will get higher profit from the capacity. One example of this kind of capacity where this optimization is important for are large batteries that are used purely for balancing purposes in the electricity grid. The batteries are very cost-efficient to use, but they have a limited capacity. Due to this the maximum battery input or output power can only be offered for a limited time in a day in the reserve market. To know when, with what power and to which market it is best to provide a capacity like a battery, it is necessary to forecast the reserve market prices to optimize the use of the capacity. By forecasting also the electricity price, it is possible to optimize, for example, when to load or discharge a battery to be able to use it again in the reserve markets.

The electricity market forecasts often benefit from using external data such as weather forecasts to improve the results. This external data typically improves when time passes so it is beneficial to do the electricity market forecasts as late as possible. Some of the markets even have an intraday market where the offers are made only a short time before the capacity activation time. For example, in the aFRR market in Finland, this time is only 25 minutes. The forecast methods on the other hand can take a varying computing time that can typically extend from some tens of seconds to even tens of minutes. The trader also requires some time after the forecasts are done to optimize the capacity use based on the forecast information. This means that if the forecast fails, there often is no time to run the forecast again. There could be some kind of a backup such as earlier run forecast for these cases, but the performance would be suboptimal because it would miss some of the data that the later forecast has. This causes a real-time requirement for the electricity market forecasts.

### **6.3 General Considerations for Tuning Autoregressive Model**

When making time series forecasts with autoregressive models, it is important to set the model parameters as optimally as possible. There are two ways to tune the SARIMAX parameters: manually or automatically. It is also possible to combine these by selecting the method separately for each parameter. This chapter compares manual and automatic tuning and discusses the use of autoregressive models for forecasting and the tuning process in general.

With manual tuning it is possible to use domain knowledge and manual data exploration and analysis to select the parameters and achieve good performance with less required computation power than with automatic tuning. However, with automatic tuning, the required domain knowledge and need for experimenting with different

models are lower. One additional benefit of automatic tuning is the ability of the model to automatically adjust to new data that might change the characteristics of the time series that is being predicted.

The available parameters to tune depend on the model, but the parameters available in a SARIMAX-model are  $p$ ,  $d$ ,  $q$ ,  $P$ ,  $D$ ,  $Q$  and  $m$ . The parameter  $p$  determines the number of past values used in the autoregressive part of the model.  $d$  is the order of differencing which is used to take into account possible trends in the data.  $q$  is the count of previous values that are used in the moving average part of the model.  $P$ ,  $D$  and  $Q$  are the corresponding parameters but for the seasonality, meaning that, for example, with  $P = 2$  the value from the previous season and the season before that are used.  $D$  determines seasonal differencing, meaning that possible seasonal trends can be removed. The parameter  $m$  determines the seasonality length in terms of datapoints. To transform a full SARIMAX model into a simpler model, one can set one or more parameters to be 0. The only required parameter is  $m$ , and only if any of  $P$ ,  $D$ ,  $Q$  is greater than 0.

With automatic tuning it is usually possible to adjust all other parameters except for  $m$  which typically requires domain knowledge. Other parameters usually require specifying a range of allowed values. This is why it is useful to analyze the time series data even if using the automatic tuning to get an understanding of sensible values. Autocorrelation and partial autocorrelation are commonly used to determine the values for  $p$ ,  $q$ ,  $P$  and  $Q$ . For determining  $d$  and  $D$  it is useful to detect trends in the data. To set the  $m$  correctly, it is possible to either use domain knowledge or analyze the data manually or combine these. Common values for  $m$  include 24 for hourly data (daily seasonality) or 12 for monthly data (yearly seasonality).

Selecting appropriate exogenous variables is also essential for optimal model performance. A careful selection is important because while good correlating exogenous variables may have a strong positive effect on the data, bad exogenous variables can make the model worse than what it would be without the variable. The methods for determining exact values are not explicit but correlation calculations are often useful. However, obtaining and testing relevant data requires domain knowledge.

## 6.4 Implementing an Autoregressive Model for Frequency Containment Reserve Price Forecasting Application

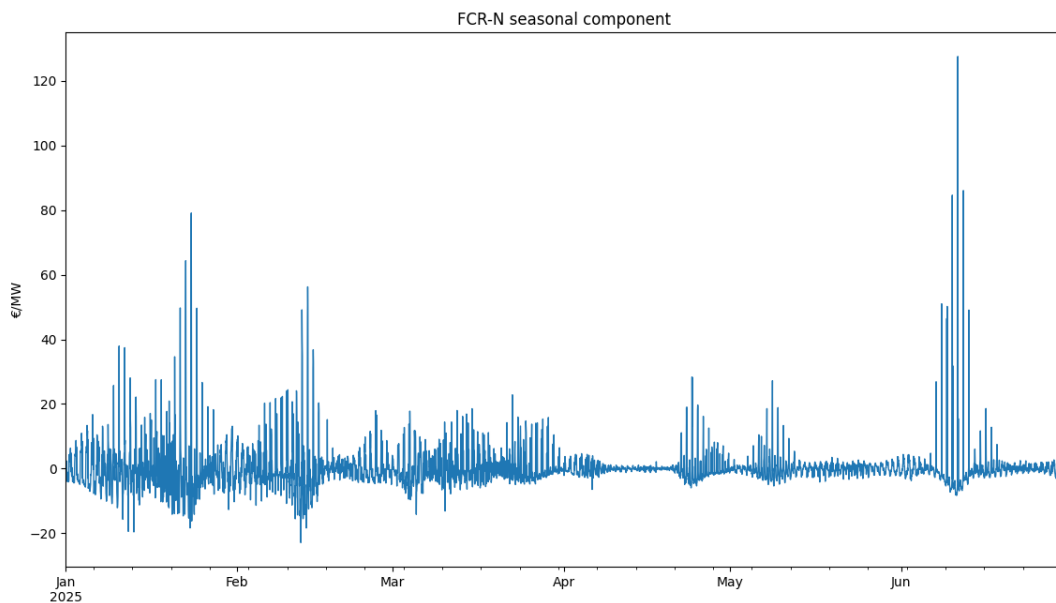
One of the research questions in this thesis is about using the SARIMAX model to forecast prices for the frequency containment reserve market for normal operation (FCR-N) in the Finnish market. This chapter discusses this forecasting process and the implementation of a SARIMAX model for that. In the implementation, the first half of data from 2025 is used for analysis, and the goal is to forecast prices for July after that.

In the implementation, electricity prices need to be forecast daily for the best results. At the same time the characteristics of the market are rapidly evolving. For this reason the automatic parameter tuning is selected for this experiment due to the ability to dynamically adapt to the changing market for every forecast. The implementation is done using a Python library called `pmdarima` [64] that includes a function called

"auto-ARIMA" for automatic parameter selection. Despite the name, it can be used also for creating a SARIMAX model. This is a common approach and, for example, [65] uses auto-ARIMA with a SARIMAX model to predict stock prices.

Although automatic tuning can adjust many of the parameters on behalf of the developer, there are some parameters that the developer needs to set. Each parameter also requires specifying selection criteria and maximum values.

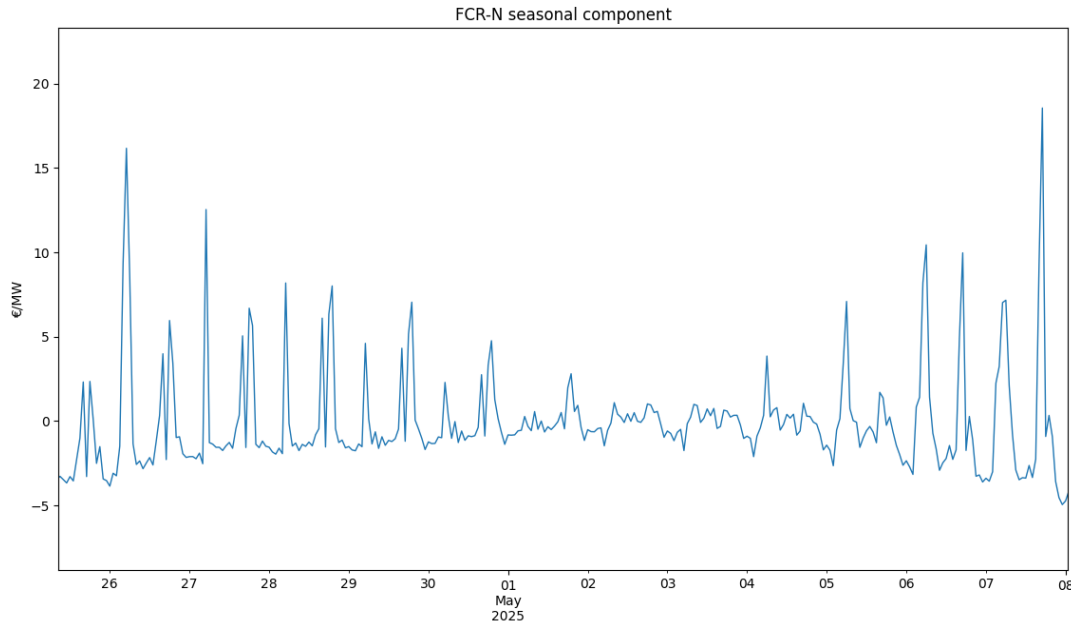
The first parameter that needs to be set is  $m$ . It is very difficult to set automatically and in the chosen library `pmdarima` it is not even possible. With FCR-N the selection of  $m$  can be based on both analysis and domain knowledge. Figure 8 presents the seasonality of the FCR-N price in the first half of 2025. From the figure, it is clearly seen that there is seasonality even though the seasonality is changing over time.



**Figure 8:** FCR-N market seasonal component for the first half of 2025.

Figure 9 presents a zoomed version of this seasonality. From this figure, we can see that there is clearly daily seasonality even though it is changing. This can be seen with the daily repeating pattern. Daily seasonality also makes sense from the domain knowledge point of view. The reserve prices are affected by electricity use and, for example, solar power, which both have daily seasonality. There could also be other seasonalities, such as weekly seasonality, in the data, but those are not taken into account here because for this model only one seasonality can be selected. So, the selection of  $m$  will be 24 as the data is hourly.

The next values to select are  $d$  and  $D$ . The auto-ARIMA model allows for automatic selection of these, but the selection criteria needs to be chosen.  $d$  and  $D$  are used to make the time series stationary.  $d$  affects the regular stationarity and  $D$  affects seasonal stationarity of the time series. For any autoregressive moving average model to work properly, the data should be stationary [66]. For a time series to be stationary, it is required that the statistical properties of it are stationary. These properties include, for example:



**Figure 9:** FCR-N market seasonal component around the start of May 2025.

- Constant mean
- Constant variance
- Constant autocorrelation

To understand which value of  $d$  is required, the stationarity of the data needs to be tested with different values for  $d$  until it is high enough to make the series stationary. The stationarity is easy to test with, for example, Augmented Dickey-Fuller test (ADF) [67]. The ADF gives a p-value signifying the probability that the time series is non-stationary. If it is less than a chosen significance level, the time series can be considered stationary. Setting the  $d$  manually would reduce required computation time and also allow setting higher values if it seems that it would be beneficial. However, setting them too high has the risk of losing important data and therefore reducing model performance. Automatic selection also allows for dynamic adjustments according to the data. For these reasons, the automatic selection is chosen for  $d$  at this point. The automatic selection is set to use the ADF test internally to determine suitable  $d$  in the experiment.

With  $D$  the idea is the same as with  $d$ , but the goal is to make the series seasonally stationary. One test that can be used for this is Osburn, Chui, Smith, Birchenhall test (OCSB) [68]. Similarly and for the same reasons as with  $d$  the value of  $D$  is determined automatically in the experiment based on the OCSB test.

The rest of the parameters  $p, q, P, Q$  have a common selection criterion. In this experiment, the chosen criterion is the Akaike information criterion (AIC) [69]. The idea of this criterion is to estimate the amount of information lost by a given model and give better scores to a model that has captured more information from the data. In

addition, it penalizes models for their complexity. In this way, the criterion takes into account both the underfitting and overfitting of the model. Underfitting means that the model is too simple to capture the data, and overfitting means that the model has fitted to noise in the training data and has become inaccurate in prediction of unknown values. It should be noted that the AIC is not able to estimate the absolute performance of the model but can estimate the performance compared to another model with the same data.

To find the best model, the auto-ARIMA is selected to use a stepwise approach that is based on the method presented in [70]. The stepwise method starts from a given parameter set and calculates the AIC for that model. After that it tries combinations that are allowed to change parameter values only by 1 to both directions from the so far found best model. It is unlikely that if a change in a parameter reduces the performance, that changing it even more would increase the performance.

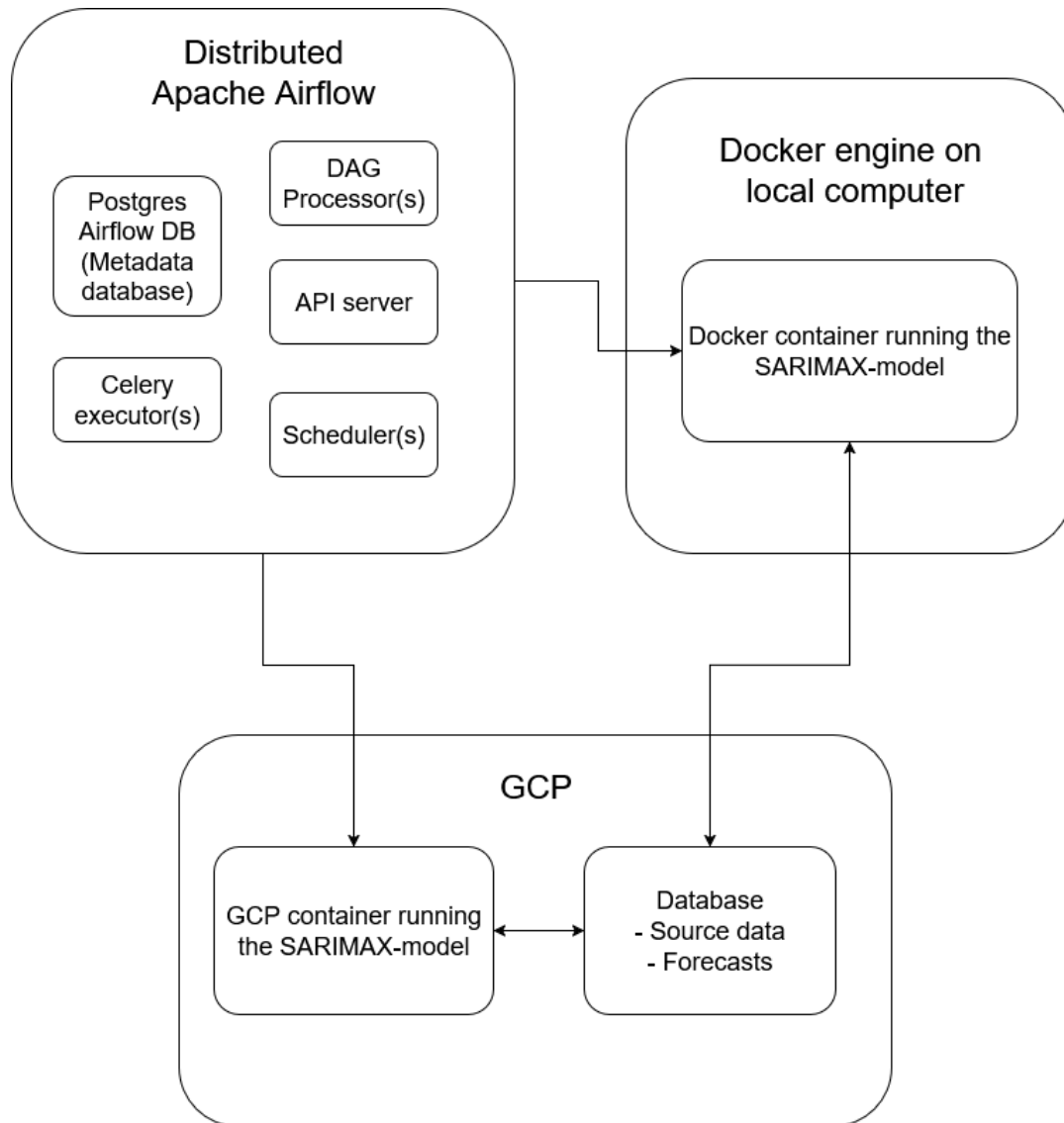
The alternative to the stepwise method would be a grid search that would try every combination of parameters and thus require a large amount of computing resources, because the grid search also tries some combinations of parameters that are clearly not suitable. In the application where the results are beneficial to be ready as early as possible, this is not a good option.

When using auto-ARIMA, the concrete values that the developer needs to select for  $p, q, P, Q$  are the maximum limit values. The minimum values are always 0. The maximum values for  $p, q$  are set to 4 and 2 for  $P, Q$  which are at the higher end of values that are usually needed but do not make the model so complex that the processing time would get too long. The maximum values are needed also for  $d$  and  $D$ . They are set to 2 because that is already very rarely needed.

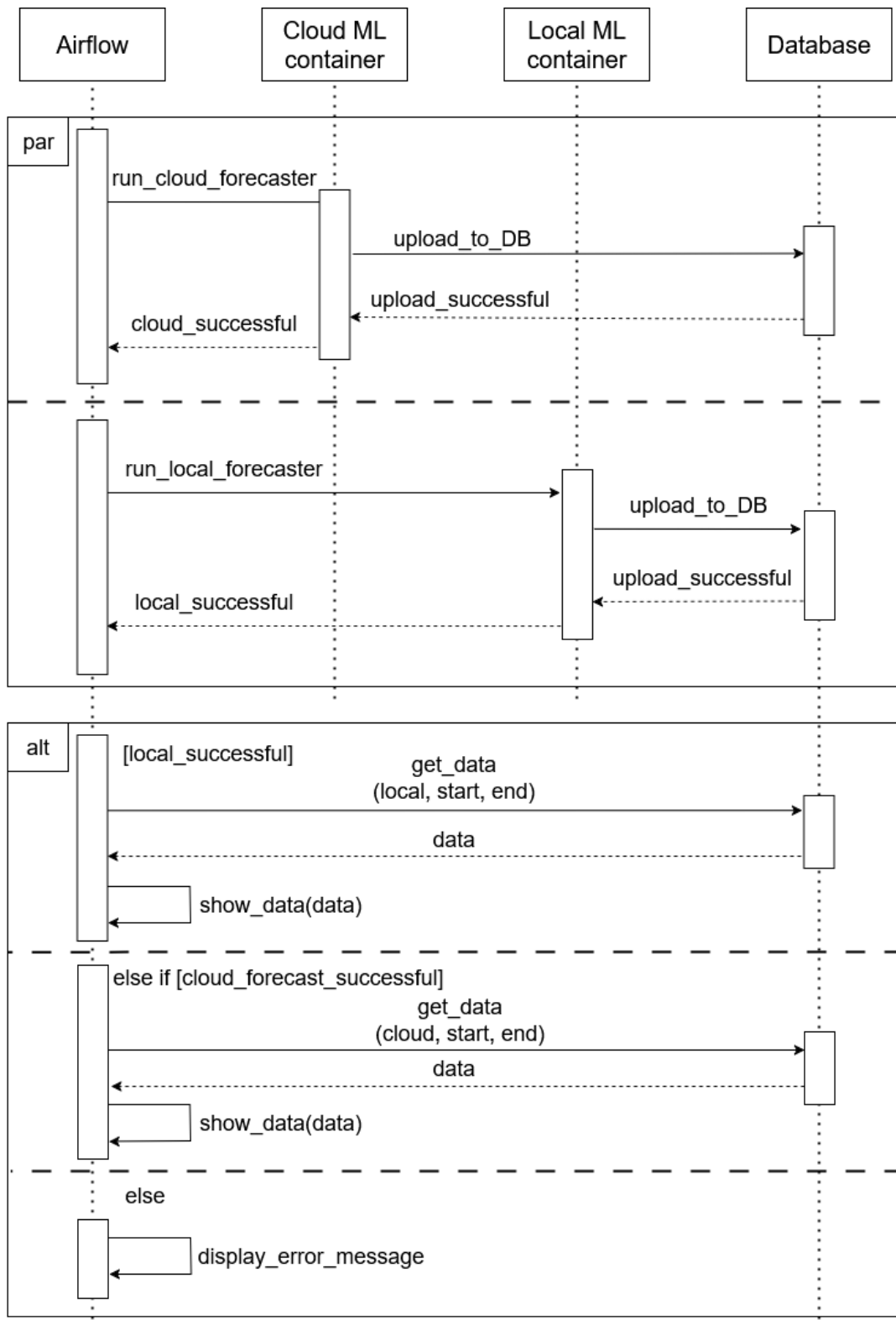
After these selections, the required parameters have an initial value set, and the model is ready for testing and further tuning. Without testing, it would be difficult to further improve the parameter selections and add exogenous variables because these changes can have unpredictable effects on the results. While the automatic selection tries different values for  $p, q, P$ , and  $Q$ , the  $d$  and  $D$  are determined by tests that aim to minimize them while keeping the data stationary. In some cases, this can result in a too simple model and higher values could improve the results. For exogenous variables, wind power generation and whether the day for which the prices are being predicted is a holiday or on a weekend seem like two good choices with domain knowledge. However, their effect and combination cannot be known without testing.

## 6.5 System Architecture and Implementation

The core component of the system is the orchestration tool Apache Airflow. Other components of the system include a database for storing raw data and results, and the forecasting model. The redundancy is implemented using a combination of local environment and Google Cloud Platform (GCP). Figure 10 shows a block diagram of the system's components and environments, and Figure 11 illustrates how these components interact. The following sections explain these components and how they work in more detail.



**Figure 10:** Block diagram of the system. The deployment location of Apache Airflow is not specified because the practical implementation of the highly available deployment is out of scope for this thesis. In the experiment it is deployed to the local Docker Engine.



**Figure 11:** Sequence diagram of the system.

### 6.5.1 Apache Airflow

Apache Airflow is selected as the orchestration tool in the experiment for being an open-source industry standard with a wide variety of integrations.

The parts of the Airflow system are the scheduler, metadata database, DAG processor, DAGs directory, and API server. In this experiment, the Airflow implementation is assumed to be highly available with redundancy. This would require a distributed deployment, as illustrated in figure 6, combined with replication. In practice, there would be multiple schedulers, DAG processors and workers in different physical locations working together to maximize the probability that at least one of each type of them is available at any given time. The metadata database and the directory for DAGs would be deployed in a distributed way where there would be multiple synchronized replications in physically different servers to ensure high availability and also help in case of a data loss in one of the servers. The practical implementation of the high availability version is out of scope for this thesis and is left for further work. Instead, a local deployment of Airflow is used with the assumption that it is highly available.

The practical Airflow deployment is done using Docker and Docker Compose. Airflow provides a Docker Image for deploying with Docker [71]. An image is a standardized package that includes all the necessary parts such as files, libraries and configurations to run a container. The configurations in the image are modified to better suit this experiment using Docker Compose. Docker Compose is a tool for defining and running applications that use Docker in multiple containers like in the case of Airflow. With Docker Compose it is easy to modify the configurations of each container and, for example, organize the startup order so that the application always starts the same way.

The API server in the Airflow deployment provides a user interface (UI) for monitoring and controlling the execution of DAGs. Figure 12 shows the main user interface (UI) of Airflow when a DAG is running. As can be seen, it also shows the health status of the main components of the system.

Figure 13 shows the tasks of the distributed forecaster in the Airflow UI. The functionality of the distributed DAG is implemented using a combination of DockerOperator and a Python task. DockerOperator is an integration of Airflow that allows for running tasks in a Docker container easily.

The triggering of the Cloud Run job is implemented directly in Python within the DAG. The same goes for showing the data. However, the local forecaster triggering is implemented using the DockerOperator. In practice, this means that the triggering creates a new container to the local Docker Engine and the Airflow is connected to it for collecting logs and following the status of that container. Otherwise the execution is not linked to Airflow. In this experiment, the execution occurs in the same Docker Engine for practical reasons, but this would not be necessary. Separate container allows for better scaling of the computing resources and separating the forecaster execution from Airflow completely. The other tasks are run inside the Airflow environment with the Celery executor(s). However, those tasks are very light so the load on the same resources that Airflow is using is very small. The tasks are linked in a way that showing the data is triggered only after both of the previous tasks are either succeeded

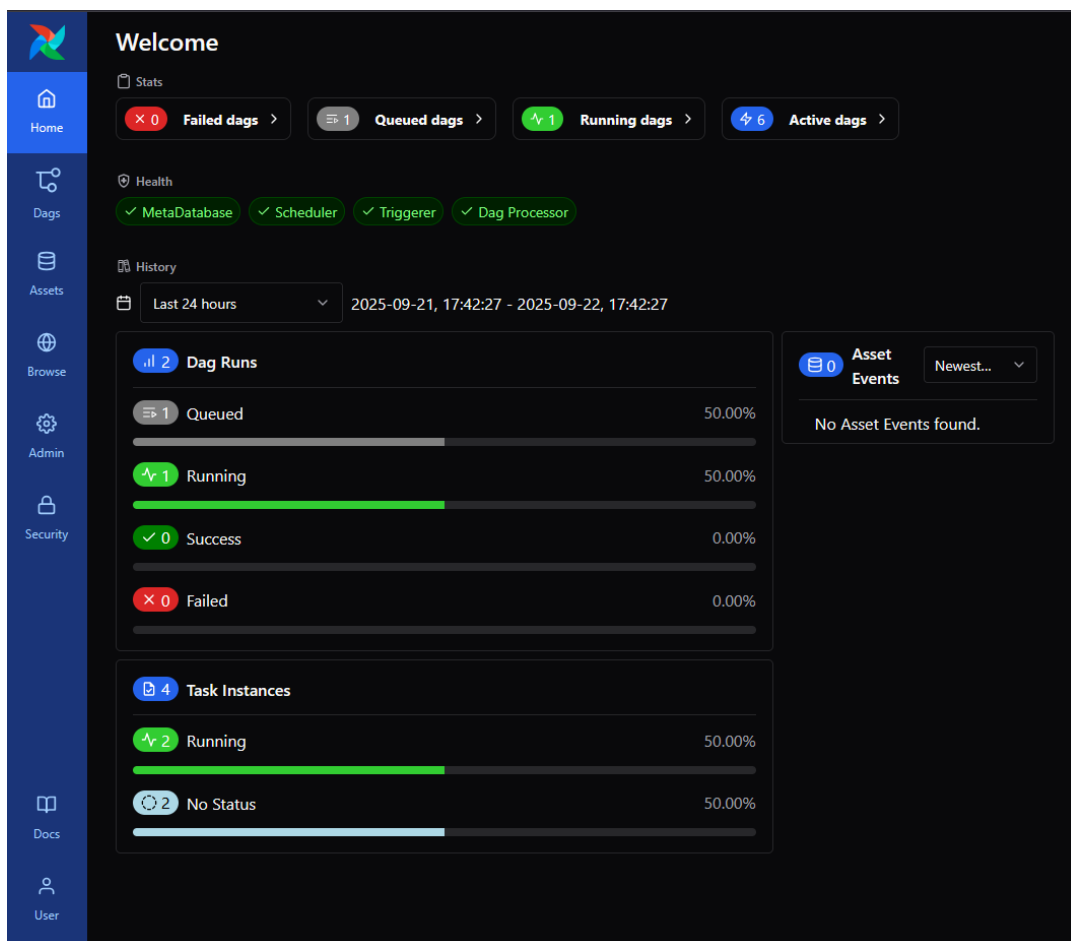
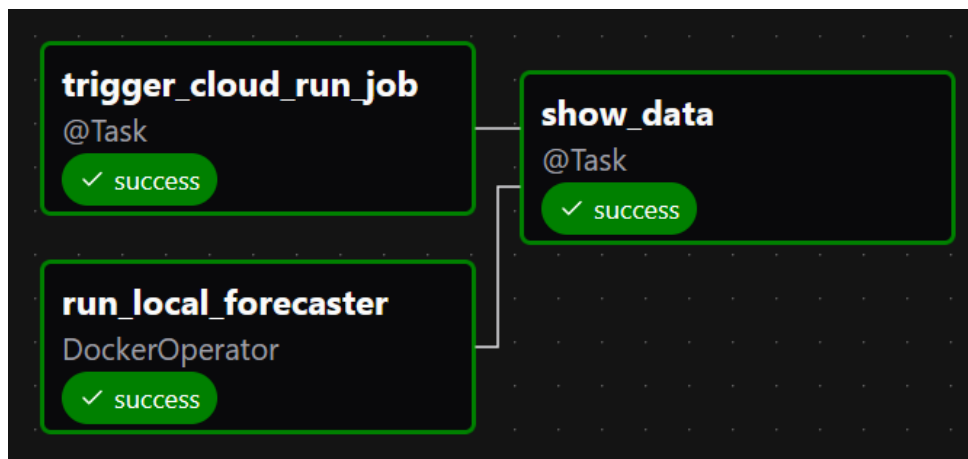


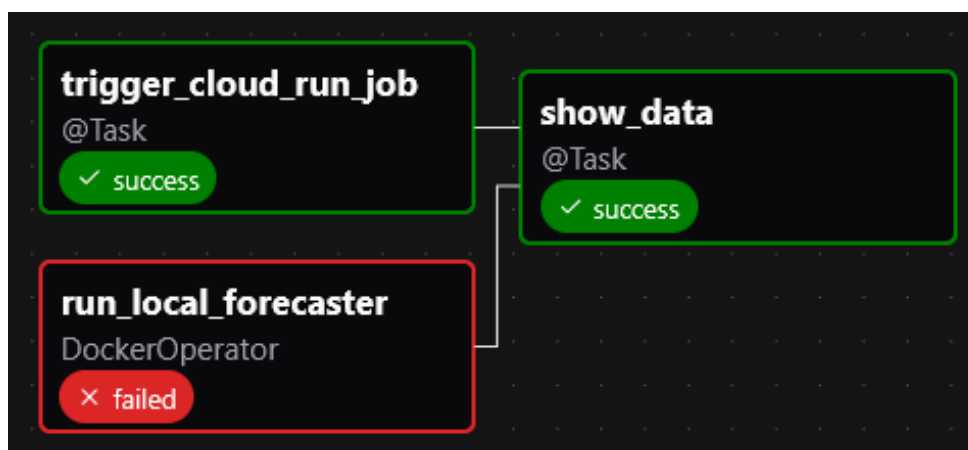
Figure 12: Airflow UI main view with a DAG running.

or failed. Showing the data would in a live scenario be, for example, sending the data forward for optimizing the bids in the electricity market.

Because the forecasting is time-critical, there is a maximum run time set for the forecasting tasks and after that they will stop even if the results are not ready. This ensures that the process proceeds at the latest when the results are available. This is needed because the forecasting time cannot be known beforehand since it is dependent on the source data. Figure 14 shows a simulated scenario where the local forecasting has failed but the cloud forecasting has been successful. The failure was caused by the complex model running out of maximum time in the local environment. Due to the diverse redundancy, this did not happen in the cloud environment with the simpler model and the results were successfully available. This shows that at least in some scenarios this architecture increases the availability of the forecasting results.



**Figure 13:** Distributed forecaster DAG in the Airflow UI.



**Figure 14:** A simulated scenario where the local forecaster has failed but the cloud forecaster has been successful due to diverse redundancy.

### 6.5.2 Forecasters

The forecasters are implemented with Python using the SARIMAX model from the pmdarima library [64]. The Python scripts are containerized to Docker images and the images are uploaded to both local Docker Engine and the GCP Artifact Registry with assisted replication. In practice, there are commands to run in the development environment that handle this uploading.

To run the forecasting in the local environment, Airflow starts a new container in the local Docker Engine using the forecaster image and the container runs until the forecasting is done or it has failed. The logs are sent to Airflow so that Airflow can keep track of the forecaster status. The code inside the container handles connection to the database for getting source data and uploading the ready forecasts.

Running the forecasting in the GCP environment is slightly more complicated. The implementation is done using GCP Cloud Run which is a PaaS meant for running code on the GCP infrastructure. In Cloud Run there is a job that creates a Docker container from the forecaster image and runs it using the GCP infrastructure. The job can be triggered externally using an API. The API is used in the Airflow DAG for both triggering the job in GCP and tracking the status so that Airflow DAG can proceed when the job has run or failed. The GCP job is connected to the database for getting the source data and uploading forecasts so that the only information going from Airflow to GCP is the triggering command and the status of the run. GCP provides a UI for configuring and running the jobs. Figure 15 shows how the UI can be used to inspect runs.

To increase diverse redundancy, the Docker image has a condition that checks the environment where it is run and based on that uses the auto-ARIMA or manually selected parameters. The local version uses auto-ARIMA and GCP uses the manual version. This way the more cost-effective local environment can be used to run the computationally more demanding automatic tuning and GCP version can run the more reliable manual version. The risks in the automatic tuning are bugs or, for example, running out of memory in the complex process and also running out of time due to the real-time requirement of the forecasts. By selecting the different approaches this way, the costs are optimized while the performance and reliability are high.

### 6.5.3 Database

The database in this project is a NoSQL database that has three replicas. It is hosted on GCP. A NoSQL database is selected to allow easy changes to the data format if there are changes in the source data or some new data sources. There are three replicas of the data to ensure high availability. Two collections in the database are important from the forecaster point of view: the source data collection and the collection for ready forecasts.

The source data collection is constantly updated by a data collector, but the implementation of that is out of scope for this thesis. The ready forecasts are uploaded from both the local and GCP Docker Containers. The data contains information about the source so that in further processes the source can be identified. If both the local and

← Job details    ▶ Execute    ▾ View & edit job configuration    🗑 Delete

Job: arima-forecaster    Region: europe-west1    Last updated: Sep 19, 2025, 6:07:07 PM

History    Observability    Triggers    YAML

Filter    Filter history    ?    ☰

	Execution ID	Creation time ↓	Tasks	End time	Actions
<input checked="" type="radio"/> ✓	arima-forecaster-zs5zt	Sep 22, 2025, 5:42:28 PM	1/1 completed	Sep 22, 2025, 5:45:33 PM	⋮
<input type="radio"/> ✓	arima-forecaster-n4x7s	Sep 19, 2025, 6:34:57 PM	1/1 completed	Sep 19, 2025, 6:39:09 PM	⋮
<input type="radio"/> ✓	arima-forecaster-xvrrp	Sep 19, 2025, 6:23:58 PM	1/1 completed	Sep 19, 2025, 6:27:41 PM	⋮
<input type="radio"/> ✓	arima-forecaster-7j4m9	Sep 19, 2025, 6:09:01 PM	1/1 completed	Sep 19, 2025, 6:11:31 PM	⋮
<input type="radio"/> ✓	arima-forecaster-dld49	Sep 19, 2025, 4:57:13 PM	1/1 completed	Sep 19, 2025, 5:04:40 PM	⋮
<input type="radio"/> ✓	arima-forecaster-qhxx5	Sep 19, 2025, 4:05:55 PM	1/1 completed	Sep 19, 2025, 4:13:27 PM	⋮
<input type="radio"/> ❗	arima-forecaster-bpxb6	Sep 19, 2025, 3:43:28 PM	Failed with errors	Sep 19, 2025, 3:45:09 PM	⋮
<input type="radio"/> ✓	arima-forecaster-gldn2	Jul 28, 2025, 7:14:32 PM	1/1 completed	Jul 28, 2025, 7:18:11 PM	⋮

**Figure 15:** GCP job UI showing the forecaster job runs.

GCP versions have succeeded, the better-performing local version using auto-ARIMA can be selected instead of the GCP version using manually selected parameters. Figure 16 shows an example of the forecast data in the database.

```
_id: ObjectId('68cd56f39b45ccc5377af3e5')
source: "cloud"
time: 2025-08-01T01:00:00.000+00:00
created_at: 2025-09-19T13:13:23.257+00:00
updated_at: 2025-09-22T14:45:28.748+00:00
value: 16.752904002508238
```

---

```
_id: ObjectId('68cd63d69b45ccc5377af4bc')
source: "local"
time: 2025-08-01T01:00:00.000+00:00
created_at: 2025-09-19T14:08:22.039+00:00
updated_at: 2025-09-19T15:16:54.264+00:00
value: 17.991978458930255
```

**Figure 16:** Examples of ready forecasts in the database.

#### 6.5.4 Output Aggregation

The forecast output aggregation is implemented in a simple way that displays the data. In a live system the aggregated results would be sent forward for analyzing and used for making the bids in the electricity markets, but this is out of scope for this thesis. The data shown is prioritized to be from the local forecaster if that is available because the local forecaster uses the more complex and likely better-performing auto-ARIMA model. If those forecasts are not available, the GCP forecasts will be used. If neither is available, no data is shown. There could be some kind of backup for these scenarios such as copying the forecasts from previous day as a backup, but that is out of scope for this part of the system.

## 7 Forecasting Model Results and Further Tuning

This chapter reviews the performance and suitability of the forecasting model for the application. The performance of the forecasting model is evaluated by forecasting values for July 2025 and calculating the difference to the real values. The forecasts are compared against heuristics and effects of some parameter changes and addition of exogenous variables are tested. After this, the whole system behavior is analyzed on a general level.

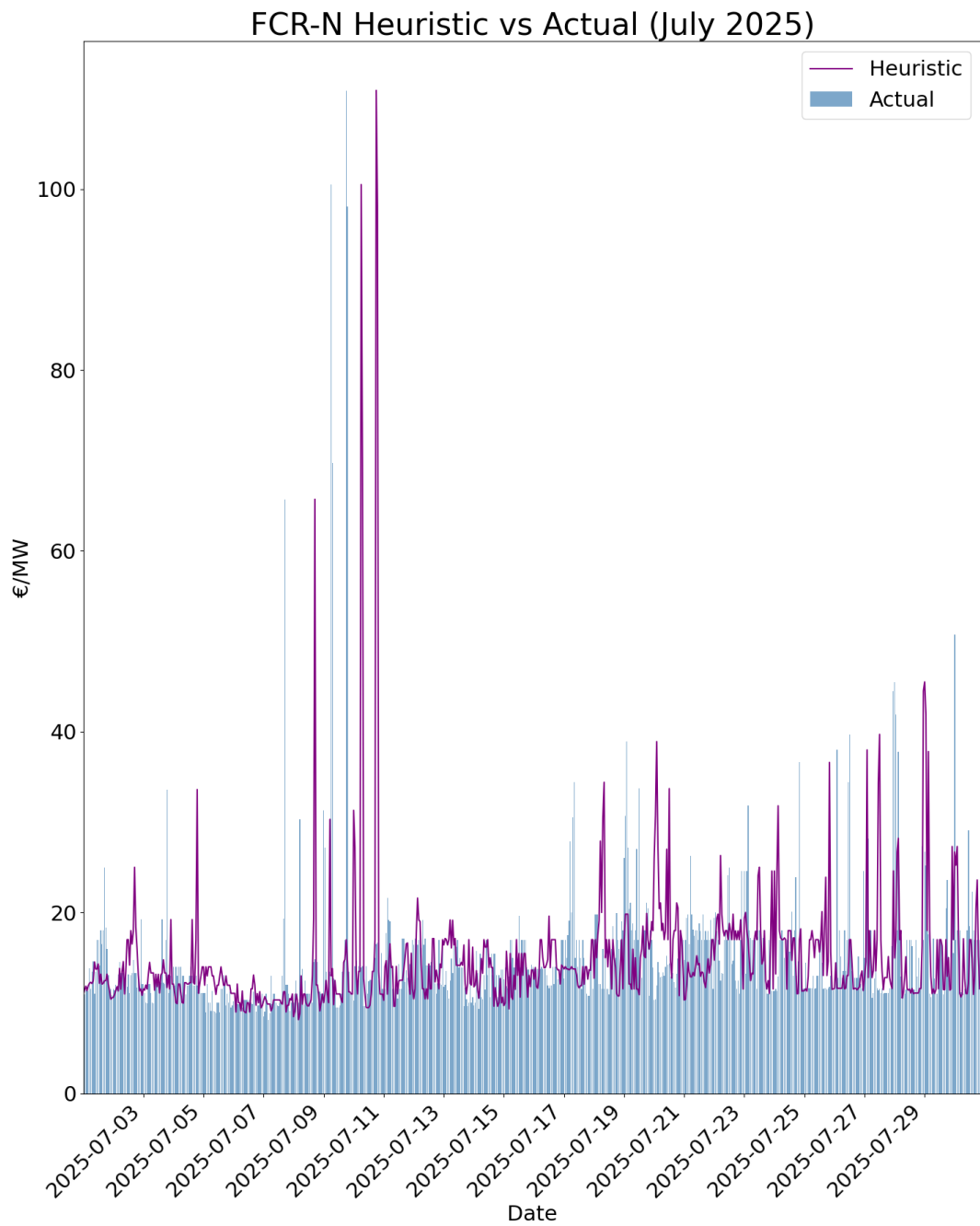
### 7.1 Evaluating the Forecasting Model Performance with Different Parameter Selections

To get an understanding of the model performance, it can be compared against a simple heuristic forecast where the values from previous day are used as the forecast. This heuristic is good when the market is stable, but when there are large peaks, they will probably cause large errors. Figure 17 shows the heuristic forecast for July 2025. In this thesis, the metrics used for evaluating the model performance are root mean squared error (RMSE) and mean absolute error (MAE). RMSE weights more heavily large errors and MAE shows how large the errors are on average. Due to the large peaks occurring at the wrong times the heuristic has quite high root mean squared error (RMSE) 10,92 €/MW compared to the mean absolute error (MAE) 4,83 €/MW.

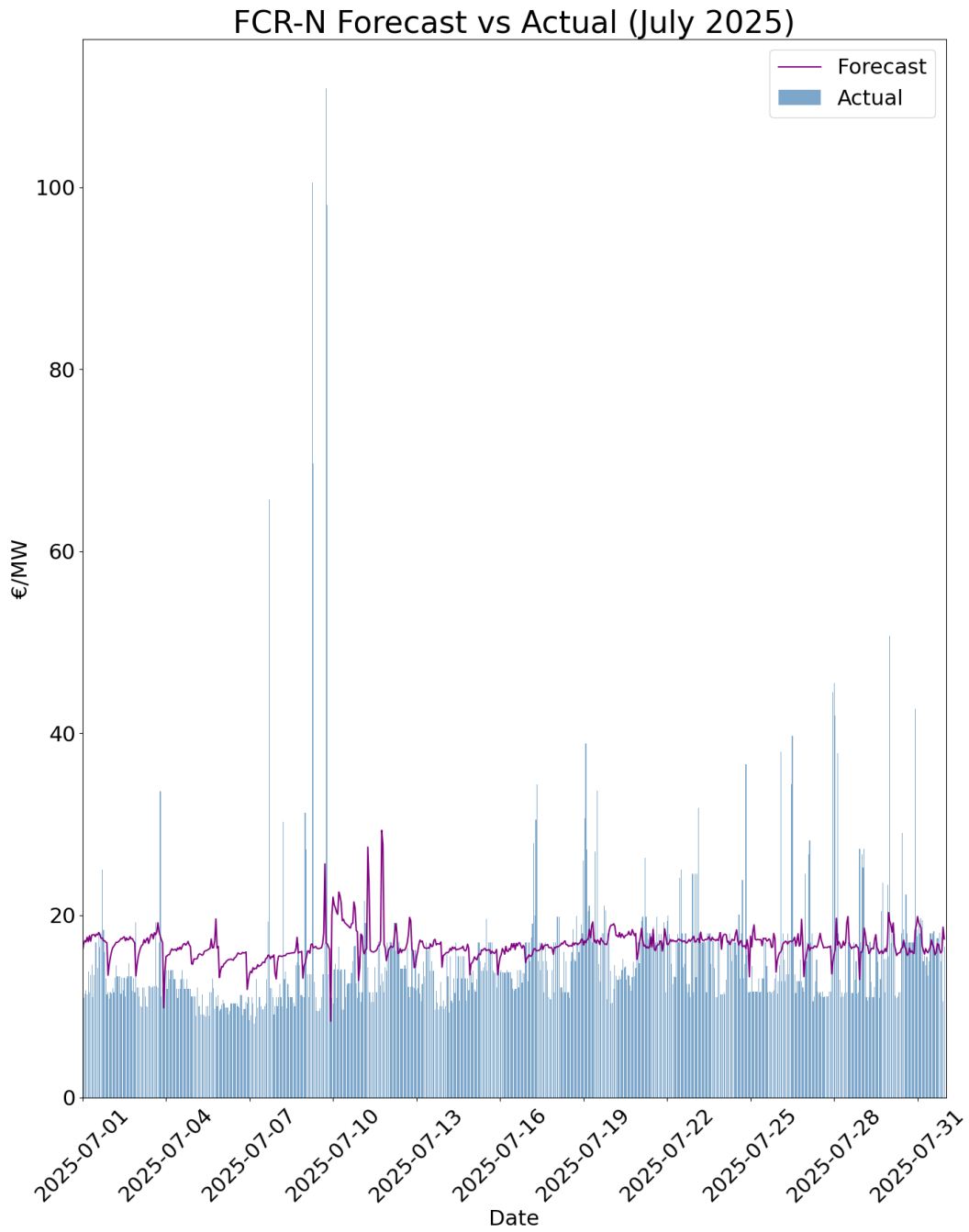
With the heuristics known, the model can be tested and compared. Figure 18 shows the forecast for July 2025 with automatic parameter selection. The parameters are selected again for each day like it would be in the live model for daily forecasts. The training data used to fit the model is selected to be the data of 3 months before the start of forecasts. The RMSE for the whole month is 8,28 €/MW and MAE is 4,62 €/MW. These are good values compared to the heuristic, but as can be seen from the figure the forecast has found the shape of the data only to a certain level and there is some repetitive shape that does not match the actual values. This is especially seen in the first week of July. The model also struggles to accurately detect price spikes.

The inability to find the correct shape and detect spikes in the data can be due to the model being too simple. The auto-ARIMA has selected  $d$  and  $D$  to be 0. This happens because the selection criterion chooses the smallest differencing order that makes the time series stationary. To make a more educated estimation of suitable values, it is possible to perform a more detailed manual analysis of the time series. One good option for this is autocorrelation function. The autocorrelation function shows the correlation between a time series and a copy of it with different lags. Figure 19 shows the autocorrelation graph. The light blue area shows a significance bound for the data and is placed to be at two standard deviations from 0. It can be used to estimate the significance of correlation of a certain lag.

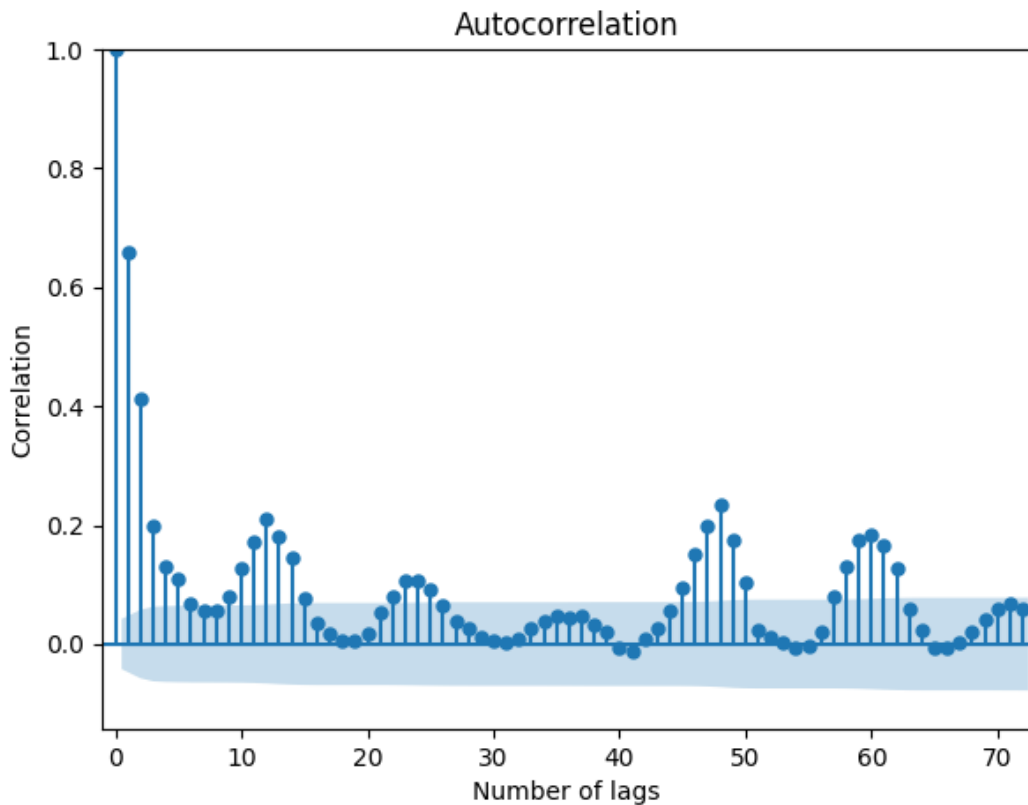
From the figure 19 it can be seen that the correlation in the first lags is dropping relatively slowly. This is a hint that there is still non-stationarity in the data, even though the ADF test result indicates stationarity. This is because the ADF test is focused on finding trends in the data, but there can still be other types of non-stationarity. To test



**Figure 17:** FCR-N heuristic for July 2025. RMSE is 10,92 €/MW and MAE 4,83 €/MW.



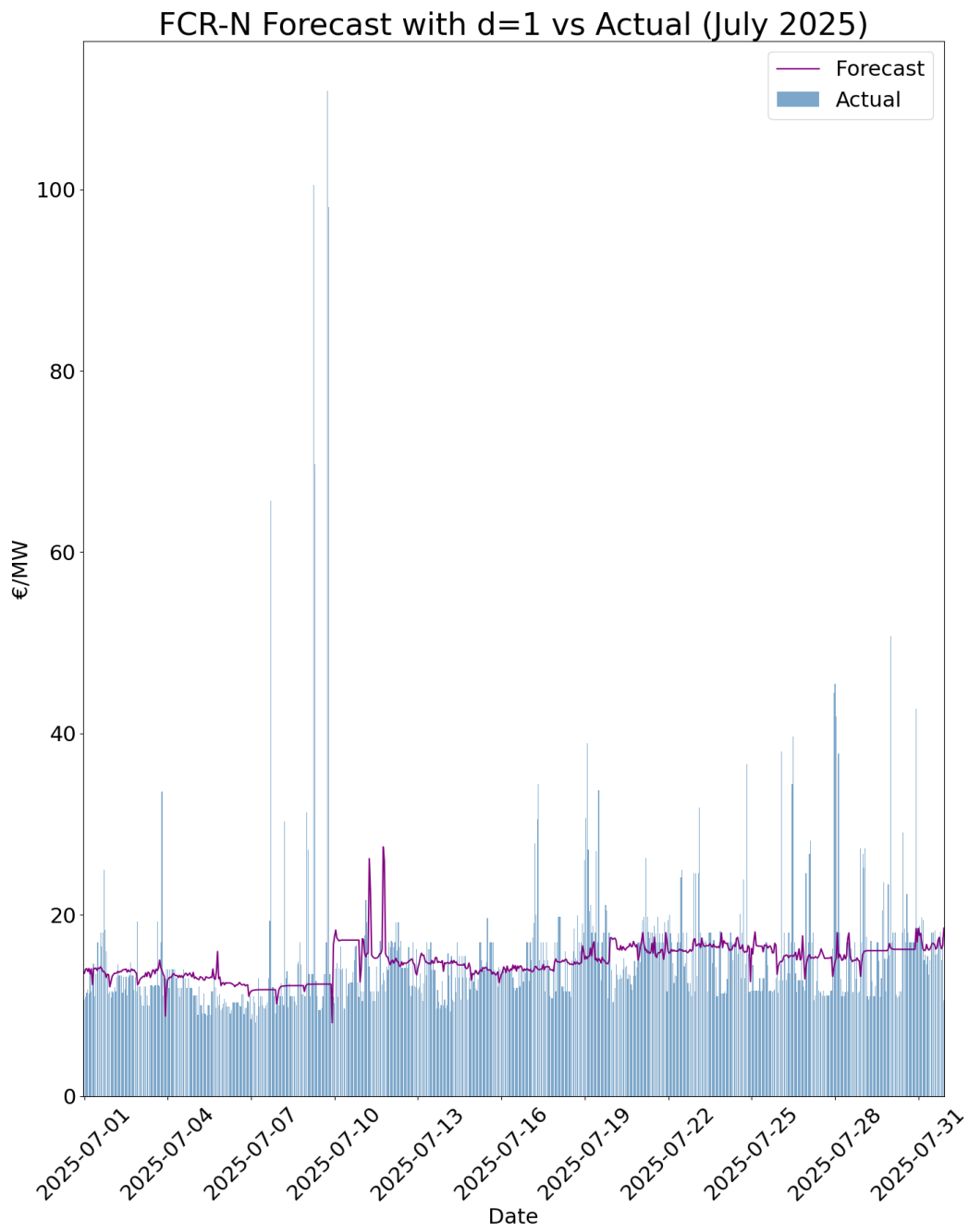
**Figure 18:** FCR-N forecast for July 2025 with auto-ARIMA. RMSE is 8,28 €/MW and MAE 4,62 €/MW.



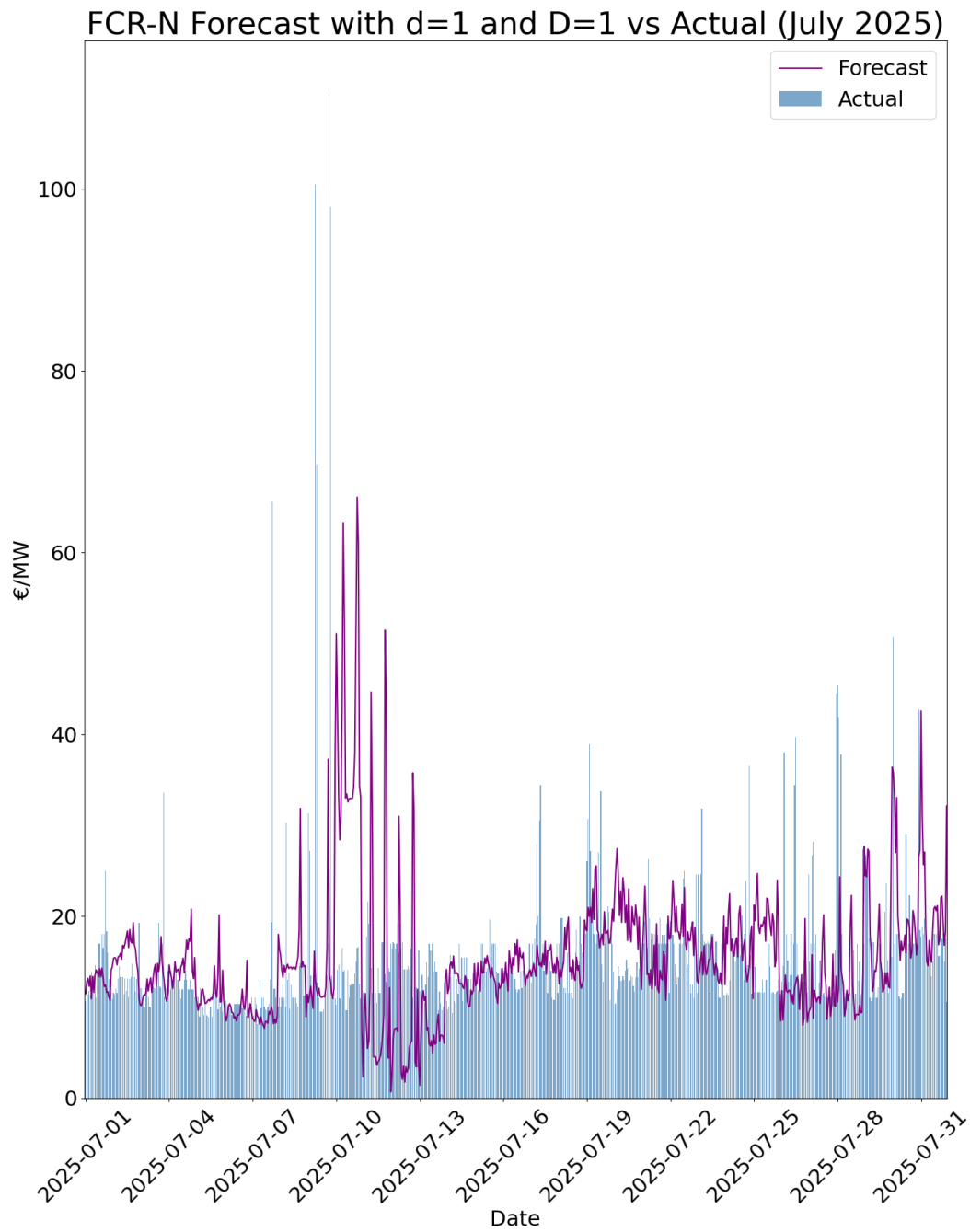
**Figure 19:** FCR-N Autocorrelation.

this hypothesis, the value of  $d$  is changed to 1 instead of automatic selection. Testing the model with  $d$  set to 1 and otherwise keeping it the same as in figure 18 results in RMSE 8,19 €/MW and MAE 3,66 €/MW which is a clear improvement in the MAE metric. This forecast can be seen in figure 20. From the figure it can be seen that the model has found the level and shape of the prices better but is still lacking especially in spike detection.

From the figure 19 it can also be seen that there are peaks at around lags 24 and 48. This is an indication of repeating seasonal pattern. Even though it is possible to use  $P$  and  $Q$  to handle the seasonality to some extent, removing the repeating pattern can in some cases improve the prediction performance. Setting both  $d$  and  $D$  to 1 results in a model that has a very different shape than earlier models, as can be seen in the figure 21. The RMSE for this is 10,70 €/MW and MAE 5,79 €/MW. So, even though the model found the spikes in the data quite well, the overall performance in terms of MSE and MAE got clearly worse due to many of the spikes being at wrong times.



**Figure 20:** FCR-N forecast using auto-ARIMA except for  $d$  that is set to 1. RMSE is 8,19 €/MW and MAE 3,66 €/MW.



**Figure 21:** FCR-N forecast using auto-ARIMA except for  $d$  and  $D$  that are set to 1. RMSE is 10,70 €/MW and MAE 5,79 €/MW.

Table 4 gathers the metrics from different tested models. As can be seen, the best model in terms of MSE and MAE is the one where  $d$  is set to 1 and other parameters are automatically selected. Based on these metrics, this is the model that is selected to be used in the experiment. This model can potentially be improved further by adding exogenous variables.

**Table 4:** Performances of different autoregressive models for FCR-N prediction for July 2025

Model	RMSE €/MW	MAE €/MW	Notes
Heuristic	10,92	4,83	Using previous day values as the forecast
Auto-ARIMA	8,28	4,62	The model seems to be too simple to capture the market dynamics
Auto-ARIMA with $d=1$	8,19	3,66	RMSE is relatively low and MAE particularly low. The market dynamics are captured better than with the above model. This is the model used in the experiment.
Auto-ARIMA with $d=1$ and $D=1$	10,70	5,79	The forecast has similar peaks to real data but they are often with wrong timing which makes the model inaccurate.

## 7.2 Evaluating the Forecasting Model Performance with Different Exogenous Variable Selections

This section examines the effect of adding exogenous variables to the best-performing model identified in the previous chapter: the auto-ARIMA model with  $d = 1$ . The selection of exogenous variables differs from other tuning processes because the number of potential variables and their combinations is large, making exhaustive testing infeasible. Therefore, domain knowledge is used to identify and preselect a small set of promising variables for evaluation.

Based on domain knowledge, three potential exogenous variables are chosen for testing: day-ahead solar power generation forecast, day-ahead wind power generation forecast, and the weekday (represented as a number) of the day being predicted. Solar and wind power generation forecasts are chosen due to solar and wind power being in a central role in determining the need for reserve capacity. Solar and wind power do not have capabilities to resist changes in the grid frequency and are also unpredictable

by nature. This causes a high need for reserve capacity to balance the grid when necessary. The weekday is chosen to take into account also the weekly cycles in the data in addition to the daily cycles that are handled with seasonality in the model. Weekly cycles are caused by electricity consumption behavior that is different, for example, in the weekend compared to other days.

Figure 22 shows the result of adding solar power generation forecast as an exogenous variable to the model where  $d$  is set to 1. As can be seen, the daily cycle is now detected better but the overall performance decreased slightly resulting in RMSE 8,23 €/MW and MAE 3,74 €/MW.

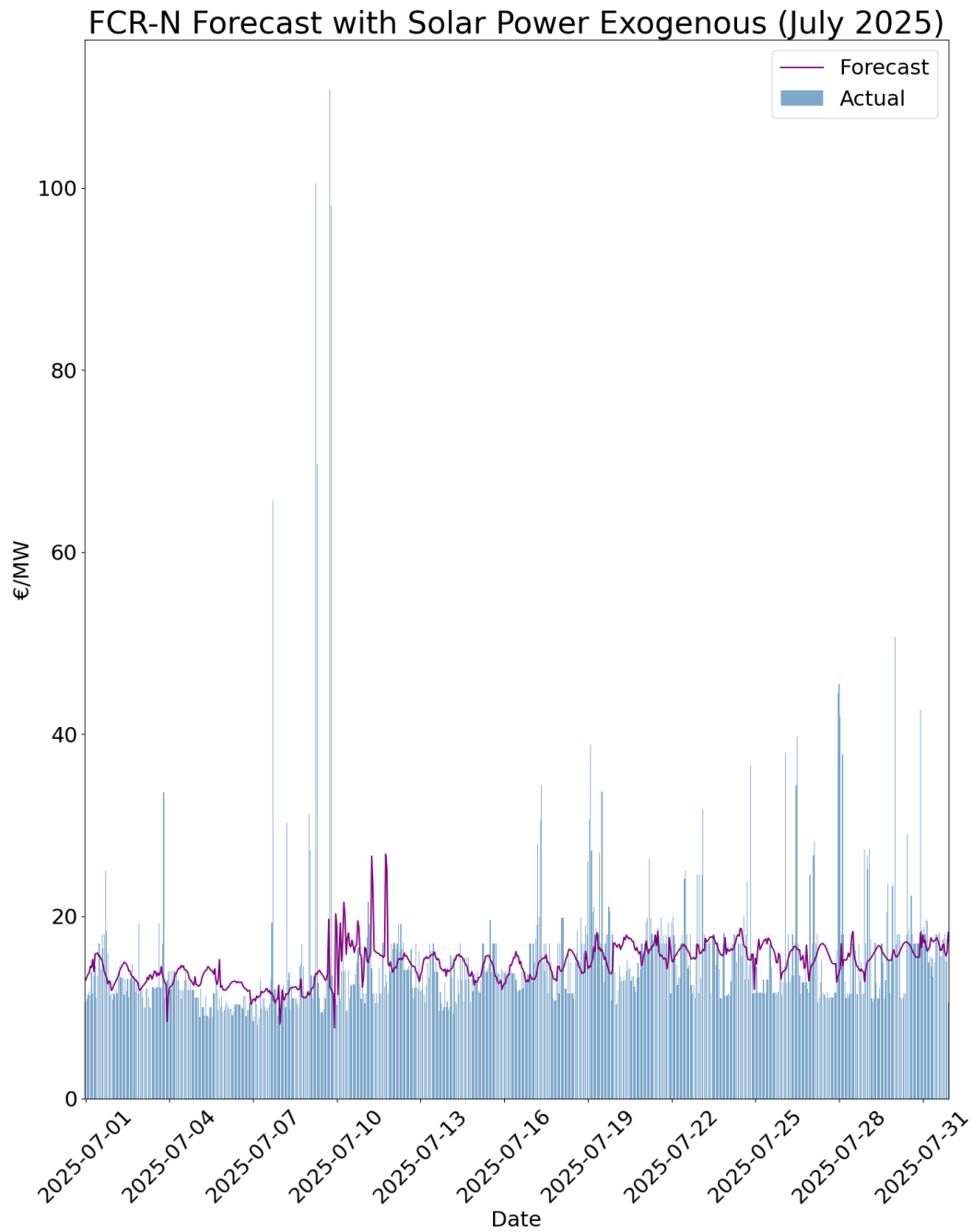
Figure 23 shows the result of adding wind power generation as an exogenous variable. As can be expected, the wind power is not cyclic in the same way as the solar power generation. It also seems that the wind power generation is not correlating very strongly with the FCR-N price because, for example, in the first week of July the forecast is deviating quite a lot from the general level of the actual prices. The RMSE is 8,28 €/MW and MAE 3,97 €/MW meaning that the results are worse than with solar power generation as an exogenous variable.

Figure 24 shows the result of using weekday number as an exogenous variable. This seems to help the model because both metrics are better than with the model without exogenous variables. RMSE is 8,17 €/MW and MAE 3,63 €/MW. From the figure it can be seen that the day number helps finding the correct level of prices but the daily cycles and peaks are not found very well.

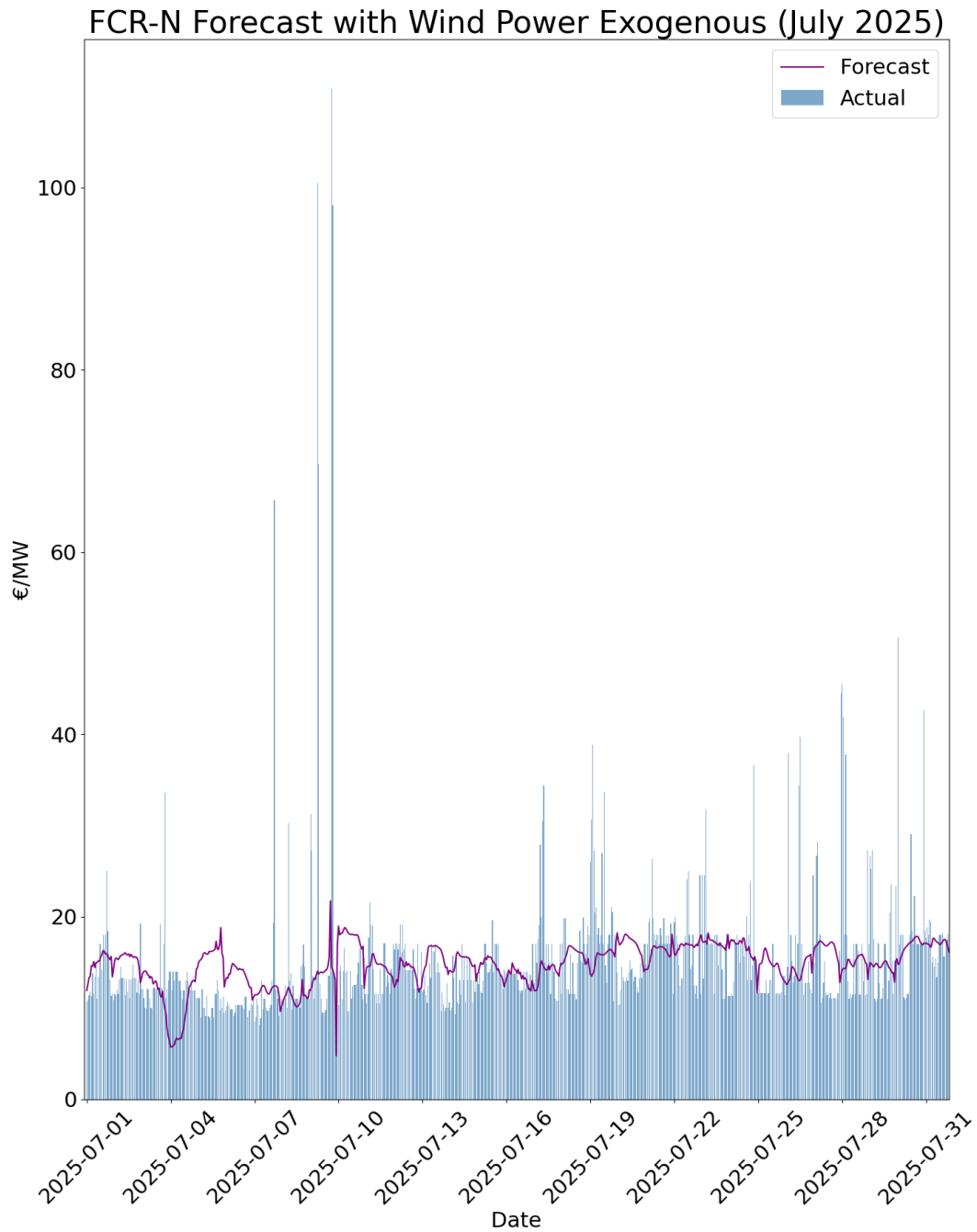
Table 5 collects the results from different exogenous variables. As can be seen from the table, the only exogenous variable that improved the performance on average was the weekday number.

**Table 5:** Performances of models with different exogenous variables for FCR-N prediction for July 2025

<b>Exogenous variable</b>	<b>RMSE €/MW</b>	<b>MAE €/MW</b>	<b>Notes</b>
No exogenous variables	8,19	3,66	Best model from previous chapter with $d=1$
Solar power generation forecast	8,23	3,74	The daily dynamic of solar power is visible
Wind power generation forecast	8,28	3,97	The forecast deviates quite a lot from actual values at some points.
Weekday number	8,17	3,63	The forecast is better than without exogenous variables.

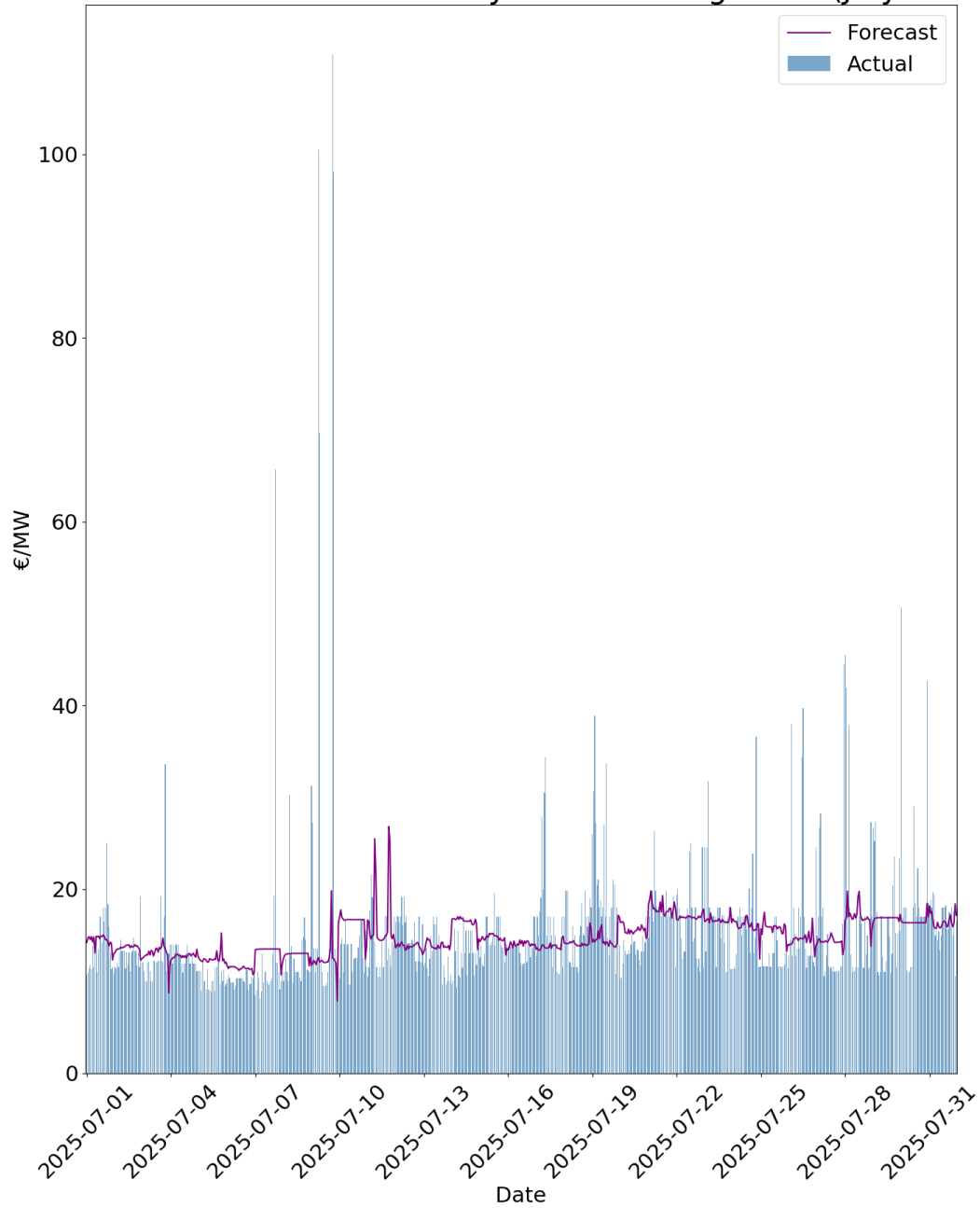


**Figure 22:** FCR-N forecast using auto-ARIMA with  $d$  set to 1 and solar power generation forecast as an exogenous variable. RMSE is 8,23 €/MW and MAE 3,74 €/MW.



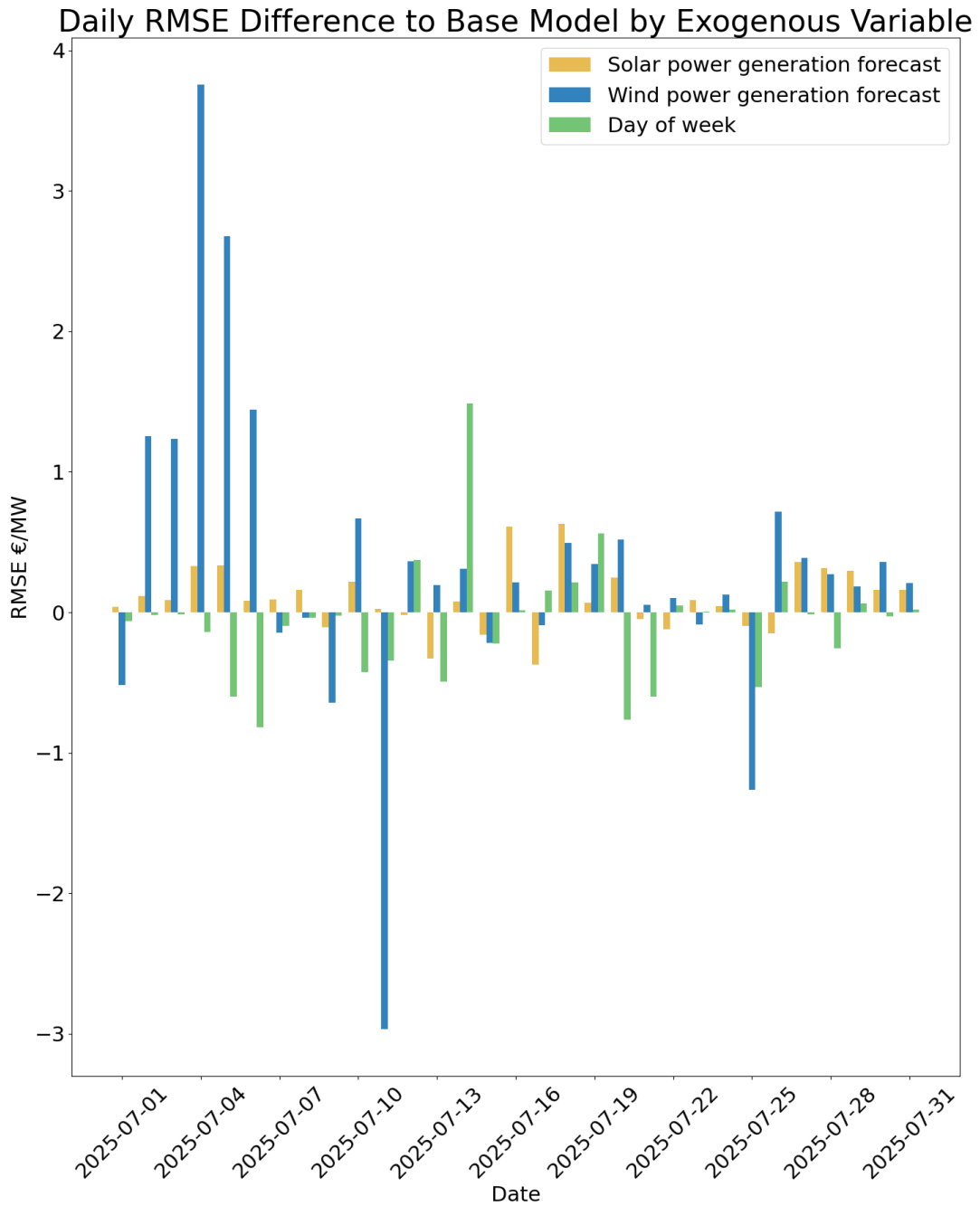
**Figure 23:** FCR-N forecast using auto-ARIMA with  $d$  set to 1 and wind power generation forecast as an exogenous variable. RMSE is 8,28 €/MW and MAE 3,97 €/MW.

FCR-N Forecast with Weekday Number Exogenous (July 2025)

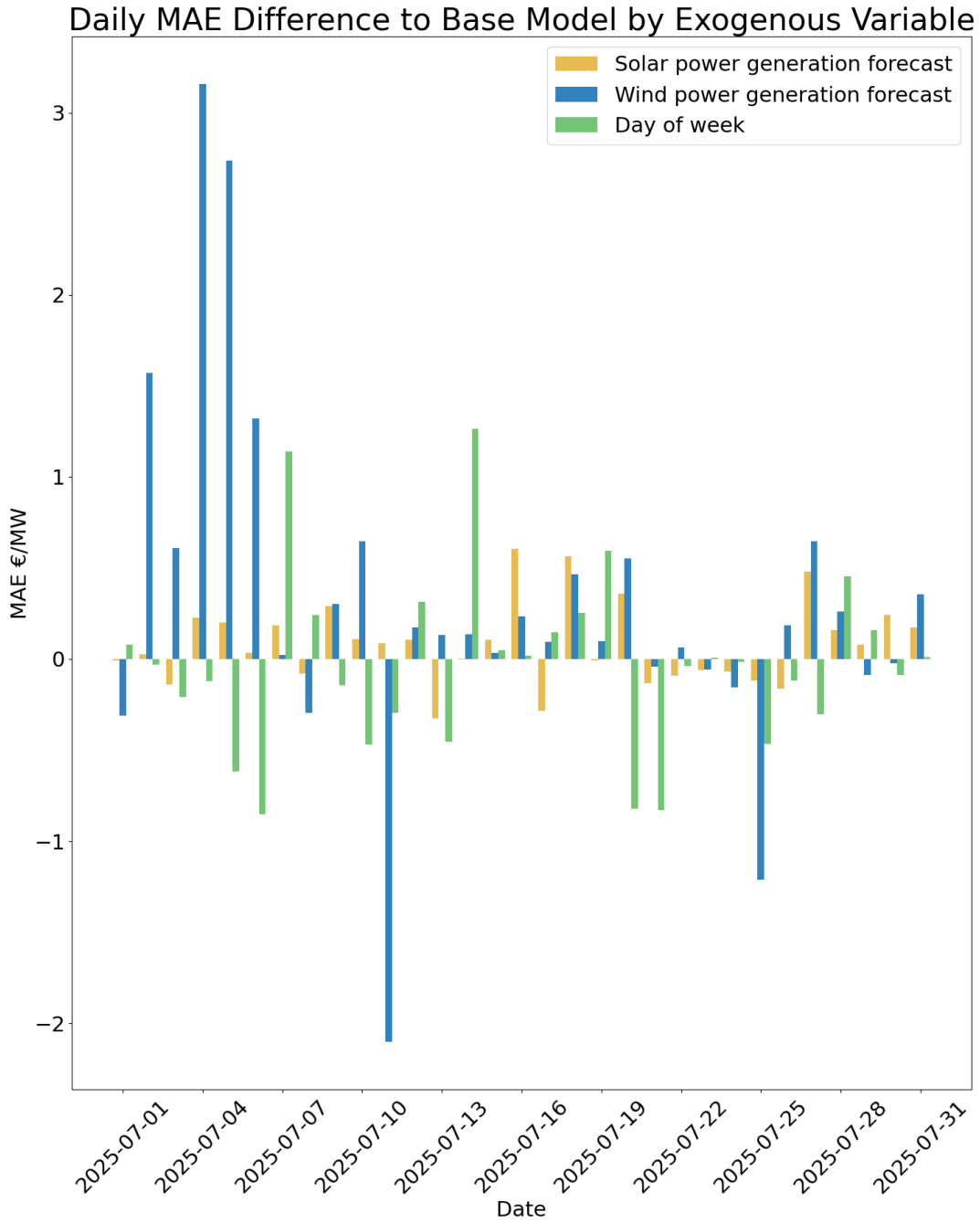


**Figure 24:** FCR-N forecast using auto-ARIMA with  $d$  set to 1 and weekday number as an exogenous variable. RMSE is 8,17 €/MW and MAE 3,63 €/MW.

The exogenous variables can be compared in more detail to better understand their effects. Figure 25 compares the daily RMSE values of models with exogenous variables to the base model without exogenous variables. Figure 26 does the same with MAE. As can be seen, all of the variables have both negative and positive effects on the error metrics depending on the day. Wind power generation forecast has a strong effect on the results on many days but often to a wrong direction so it is not really useful. Solar power generation forecast and day of the week have both a moderate effect on the results but the effect seems to be quite evenly distributed to both directions. Based on these results leaving these exogenous variables out of the model could possibly be the best solution even though the weekday number slightly improved the metrics over the whole month. By leaving the variables out, some unexpected behavior is avoided, and the model training time is decreased. So, the final model used in the experiment is an automatically tuned SARIMA model with  $d$  set to 1.



**Figure 25:** Daily RMSE difference of models with different exogenous variables compared to the model without exogenous variables. Negative numbers mean better performance compared to the base model.



**Figure 26:** Daily MAE difference of models with different exogenous variables compared to the model without exogenous variables. Negative numbers mean better performance compared to the base model.

## 8 Conclusions and Further Work

The first research question of this thesis examined how hybrid cloud architecture can be leveraged to improve the availability of an electricity market forecasting system. Based on the literature review and the experiment, a hybrid cloud solution can achieve high availability and also cost efficiency by combining at least two different cloud environments and utilizing their complementary characteristics. In the experiment, this was implemented by combining the local private cloud with a public cloud.

The second research question focused on how a SARIMAX model can be configured for reserve market forecasting in the Finnish FCR-N market. This was experimented based on a literature review of the functionality of the SARIMAX model. Based on the experiment, a suitable way of using the SARIMAX is daily retraining of the model with automatic parameter selection to achieve dynamic adaptation to the daily changing input data. The inclusion of exogenous variables did not yield significant performance improvements, and their use was therefore not justified given the additional training time required. The final model clearly outperformed the comparison heuristic forecast in both RMSE and MAE metrics.

The third research question addressed how a hybrid cloud architecture can support a mission-critical time series forecasting application through diverse redundancy. This was experimented by applying hybrid cloud architecture in combination with distinct implementations in different cloud environments to achieve diverse redundancy. Based on the experimentation, this approach does improve the system availability in several scenarios including cases where the training and predicting with the more complex local model cannot complete within real-time requirements, but the simpler version deployed in the cloud can.

Overall, these findings support the hypothesis that hybrid cloud architecture is applicable for real-time mission critical systems in electricity market participation. The hybrid cloud setup was both investigated and implemented, and the literature review confirmed the need for a real-time performance in this application. Potential improvements in forecasting accuracy were identified through the use of more complex price forecasting models, demonstrating the advantages of a hybrid cloud architecture in which computationally intensive models can run cost-effectively on local infrastructure, while simpler, highly available models run in the cloud. Finally, this diversely redundant setup with different computing platforms and different forecasting models was shown to significantly increase the likelihood that at least one model would complete within the real-time constraints.

Several subjects remain for future research and development. First, a more robust implementation of the orchestration tool should be explored. This would be important to increase the availability in scenarios where the issue would be due to the environment or orchestration tool instead of the forecasting model. Second, the forecasting model could be further improved. One promising direction is to use the combination of SARIMAX and GARCH models as, for example, in [55] and [57]. Alternatively, or in addition, a more extensive investigation of the possible exogenous variables or combinations of them could improve the performance of the selected model further.

## References

- [1] Anneli Frantti. *Lots of news on the reserve front*. Accessed: 2025-08-11. 2020. URL: <https://www.fingridlehti.fi/en/lots-of-news-on-the-reserve-front/#c02bf9fa>.
- [2] C. Giovanelli et al. “Towards an aggregator that exploits big data to bid on frequency containment reserve market”. In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. 2017, pp. 7514–7519. DOI: [10.1109/IECON.2017.8217316](https://doi.org/10.1109/IECON.2017.8217316).
- [3] C. Giovanelli et al. “Exploiting Artificial Neural Networks for the Prediction of Ancillary Energy Market Prices”. In: *Energies* 11.7 (2018). ISSN: 1996-1073. DOI: [10.3390/en11071906](https://doi.org/10.3390/en11071906). URL: <https://www.mdpi.com/1996-1073/11/7/1906>.
- [4] S. Aleem et al. “Empirical Investigation of Key Factors for SaaS Architecture”. In: *IEEE Transactions on Cloud Computing* 9.3 (2021), pp. 1037–1049. DOI: [10.1109/TCC.2019.2906299](https://doi.org/10.1109/TCC.2019.2906299).
- [5] N. Ekwe-Ekwe and L. Amos. “The State of FaaS: An Analysis of Public Functions-as-a-Service Providers”. In: *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. 2024, pp. 430–438. DOI: [10.1109/CLOUD62652.2024.00055](https://doi.org/10.1109/CLOUD62652.2024.00055).
- [6] K. K. Azumah, L. T. Sørensen, and R. Tadayoni. “Hybrid Cloud Service Selection Strategies: A Qualitative Meta-Analysis”. In: *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*. 2018, pp. 1–8. DOI: [10.1109/ICASTECH.2018.8506887](https://doi.org/10.1109/ICASTECH.2018.8506887).
- [7] S. Sok et al. “Optimization of Compute Costs in Hybrid Clouds with Full Rescheduling”. In: *2020 IEEE International Conference on Smart Cloud (SmartCloud)*. 2020, pp. 35–40. DOI: [10.1109/SmartCloud49737.2020.00016](https://doi.org/10.1109/SmartCloud49737.2020.00016).
- [8] A. Gupta and A. Kumar. “Mid Term Daily Load Forecasting using ARIMA, Wavelet-ARIMA and Machine Learning”. In: *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. 2020, pp. 1–5. DOI: [10.1109/EEEIC/ICPSEurope49358.2020.9160563](https://doi.org/10.1109/EEEIC/ICPSEurope49358.2020.9160563).
- [9] U. M. Sirisha, M. C. Belavagi, and G. Attigeri. “Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison”. In: *IEEE Access* 10 (2022), pp. 124715–124727. DOI: [10.1109/ACCESS.2022.3224938](https://doi.org/10.1109/ACCESS.2022.3224938).
- [10] V. K, A. R. Deepti, and F. Basith. “Forecasting the Consumer Price Index using SARIMAX Modeling”. In: *2024 Second International Conference on Inventive Computing and Informatics (ICICI)*. 2024, pp. 477–483. DOI: [10.1109/ICICI62254.2024.00083](https://doi.org/10.1109/ICICI62254.2024.00083).

- [11] M. Deb and A. Choudhury. “Hybrid Cloud: A New Paradigm in Cloud Computing”. In: *Machine Learning Techniques and Analytics for Cloud Security*. 2022, pp. 1–23. DOI: [10.1002/9781119764113.ch1](https://doi.org/10.1002/9781119764113.ch1).
- [12] S. Barhate and M. Dhore. “Hybrid Cloud: A Cost Optimised Solution To Cloud Interoperability”. In: *2020 International Conference on Innovative Trends in Information Technology (ICITIIT)*. 2020, pp. 1–5. DOI: [10.1109/ICITIIT49094.2020.9071563](https://doi.org/10.1109/ICITIIT49094.2020.9071563).
- [13] R. Gohil and H. Patel. “Comparative Analysis of Cloud Platform: Amazon Web Service, Microsoft Azure, And Google Cloud Provider: A Review”. In: *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2024, pp. 1–5. DOI: [10.1109/ICCCNT61001.2024.10725914](https://doi.org/10.1109/ICCCNT61001.2024.10725914).
- [14] LUMI. *LUMI Documentation*. Accessed: 2025-08-6. 2025. URL: <https://docs.lumi-supercomputer.eu/>.
- [15] M. Saraswat and R. Tripathi. “Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms”. In: *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*. 2020, pp. 300–305. DOI: [10.1109/SMART50582.2020.9337157](https://doi.org/10.1109/SMART50582.2020.9337157).
- [16] J. Kohler. “Comparison of a FaaS- and SaaS-Based ETL Process in a Cloud Environment”. In: *2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*. 2023, pp. 01–07. DOI: [10.1109/CloudTech58737.2023.10366090](https://doi.org/10.1109/CloudTech58737.2023.10366090).
- [17] R. Veuvolu et al. “Cloud Computing Based (Serverless computing) using Serverless architecture for Dynamic Web Hosting and cost Optimization”. In: *2023 International Conference on Computer Communication and Informatics (ICCCI)*. 2023, pp. 1–6. DOI: [10.1109/ICCCI56745.2023.10128286](https://doi.org/10.1109/ICCCI56745.2023.10128286).
- [18] X. Zhang, G. Li, and H. Tan. “FISH: Alleviating Cold Start for Serverless Computing with Memory Constraints”. In: *2024 10th International Conference on Big Data Computing and Communications (BigCom)*. 2024, pp. 34–41. DOI: [10.1109/BIGCOM65357.2024.00014](https://doi.org/10.1109/BIGCOM65357.2024.00014).
- [19] F. Liu and Y. Niu. “Demystifying the Cost of Serverless Computing: Towards a Win-Win Deal”. In: *IEEE Transactions on Parallel and Distributed Systems* 35.1 (2024), pp. 59–72. DOI: [10.1109/TPDS.2023.3330849](https://doi.org/10.1109/TPDS.2023.3330849).
- [20] S. M. Razavian et al. “An analysis of vendor lock-in problem in cloud storage”. In: *ICCKE 2013*. 2013, pp. 331–335. DOI: [10.1109/ICCKE.2013.6682808](https://doi.org/10.1109/ICCKE.2013.6682808).
- [21] Y. Lin. “Minimizing Cold Start Time on Serverless Platforms Based on Time Series Prediction Methods”. In: *2024 IEEE 2nd International Conference on Control, Electronics and Computer Technology (ICCECT)*. 2024, pp. 996–1001. DOI: [10.1109/ICCECT60629.2024.10545789](https://doi.org/10.1109/ICCECT60629.2024.10545789).

- [22] S. H. Alisha et al. “Analyzing Cloud Performance Optimization: Strategies to Enhance Cold Start Latency”. In: *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*. 2024, pp. 487–492. DOI: [10.1109/ICOSEC61587.2024.10722462](https://doi.org/10.1109/ICOSEC61587.2024.10722462).
- [23] D. Haputhanthri et al. “Solar Irradiance Nowcasting for Virtual Power Plants Using Multimodal Long Short-Term Memory Networks”. In: *Frontiers in Energy Research* Volume 9 - 2021 (2021). ISSN: 2296-598X. DOI: [10.3389/fenrg.2021.722212](https://doi.org/10.3389/fenrg.2021.722212). URL: <https://www.frontiersin.org/journals/energy-research/articles/10.3389/fenrg.2021.722212>.
- [24] Fingrid. *Terms and conditions for providers of Automatic Frequency Restoration Reserves (aFRR)*. Accessed: 2025-08-6. 2025. URL: <https://www.fingrid.fi/globalassets/dokumentit/en/electricity-market/reserves/attachment-1-terms-and-conditions-for-providers-of-automatic-frequency-restoration-reserves-afrr-valid-from-5.6.2025.pdf>.
- [25] Google Cloud. *Vertex AI Platform*. Accessed: 2025-08-01. 2025. URL: <https://cloud.google.com/vertex-ai>.
- [26] Microsoft Azure. *Azure Machine Learning*. Accessed: 2025-08-01. 2025. URL: <https://azure.microsoft.com/en-us/products/machine-learning>.
- [27] X. Li et al. “Discovery of Floating-Point Differences Between NVIDIA and AMD GPUs”. In: *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2024, pp. 663–666. DOI: [10.1109/CCGrid59990.2024.00083](https://doi.org/10.1109/CCGrid59990.2024.00083).
- [28] Google Cloud. *GPU machine types*. Accessed: 2025-08-06. 2025. URL: <https://cloud.google.com/compute/docs/gpus>.
- [29] LUMI. *GPU nodes - LUMI-G*. Accessed: 2025-08-6. 2025. URL: <https://docs.lumi-supercomputer.eu/hardware/lumig/>.
- [30] Microsoft Azure. *Graphics processing unit (GPU) virtual machine (VM) on Azure Stack Hub*. Accessed: 2025-08-6. 2025. URL: <https://learn.microsoft.com/en-us/azure-stack/user/gpu-vm-about?view=azs-2501>.
- [31] E. Sturru and O. Kulikova. “Orchestrating Hybrid Cloud Deployment: An Overview”. In: *Computer* 47.6 (2014), pp. 85–87. DOI: [10.1109/MC.2014.159](https://doi.org/10.1109/MC.2014.159).
- [32] D. S. Linthicum. “Emerging Hybrid Cloud Patterns”. In: *IEEE Cloud Computing* 3.1 (2016), pp. 88–91. DOI: [10.1109/MCC.2016.22](https://doi.org/10.1109/MCC.2016.22).
- [33] D. Sitaram et al. “Orchestration Based Hybrid or Multi Clouds and Interoperability Standardization”. In: *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 2018, pp. 67–71. DOI: [10.1109/CCEM.2018.00018](https://doi.org/10.1109/CCEM.2018.00018).

- [34] R.-C. Chioreanu et al. “Implementing and securing a hybrid cloud for a healthcare information system”. In: *2014 11th International Symposium on Electronics and Telecommunications (ISETC)*. 2014, pp. 1–4. DOI: [10.1109/ISETC.2014.7010776](https://doi.org/10.1109/ISETC.2014.7010776).
- [35] Y.-H. Kuo, Y.-L. Jeng, and J.-N. Chen. “A Hybrid Cloud Storage Architecture for Service Operational High Availability”. In: *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. 2013, pp. 487–492. DOI: [10.1109/COMPSACW.2013.94](https://doi.org/10.1109/COMPSACW.2013.94).
- [36] M. J. Amiri et al. “SeeMoRe: A Fault-Tolerant Protocol for Hybrid Cloud Environments”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1345–1356. DOI: [10.1109/ICDE48307.2020.00120](https://doi.org/10.1109/ICDE48307.2020.00120).
- [37] S. Yasuda, C. Lee, and S. Date. “An Adaptive Cloud Bursting Job Scheduler based on Deep Reinforcement Learning”. In: *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. 2021, pp. 217–224. DOI: [10.1109/HPBDIS53214.2021.9658447](https://doi.org/10.1109/HPBDIS53214.2021.9658447).
- [38] S. Chen et al. “A Proxy Based Connection Mechanism for Hybrid Cloud Virtual Network”. In: *2017 IEEE 3rd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (Hpsc), and IEEE International Conference on Intelligent Data and Security (IDS)*. 2017, pp. 80–85. DOI: [10.1109/BigDataSecurity.2017.55](https://doi.org/10.1109/BigDataSecurity.2017.55).
- [39] M. Artac et al. “DevOps: Introducing Infrastructure-as-Code”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017, pp. 497–498. DOI: [10.1109/ICSE-C.2017.162](https://doi.org/10.1109/ICSE-C.2017.162).
- [40] Q. Liao. “Modelling CI/CD Pipeline Through Agent-Based Simulation”. In: *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2020, pp. 155–156. DOI: [10.1109/ISSREW51248.2020.00059](https://doi.org/10.1109/ISSREW51248.2020.00059).
- [41] C. Pahl. “Containerization and the PaaS Cloud”. In: *IEEE Cloud Computing* 2.3 (2015), pp. 24–31. DOI: [10.1109/MCC.2015.51](https://doi.org/10.1109/MCC.2015.51).
- [42] J. Mehta et al. “Adaptive ETL: Secure and Cloud Native Framework for Supply Chain Data Management”. In: *2025 10th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. 2025, pp. 473–479. DOI: [10.1109/ICCCBDA64898.2025.11030417](https://doi.org/10.1109/ICCCBDA64898.2025.11030417).
- [43] R. Chowdhury et al. “A Framework for Automated Monitoring and Orchestration of Cloud-Native applications”. In: *2020 International Symposium on Networks, Computers and Communications (ISNCC)*. 2020, pp. 1–6. DOI: [10.1109/ISNCC49221.2020.9297238](https://doi.org/10.1109/ISNCC49221.2020.9297238).
- [44] A. Zhou et al. “Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization”. In: *IEEE Transactions on Services Computing* 10.6 (2017), pp. 902–913. DOI: [10.1109/TSC.2016.2519898](https://doi.org/10.1109/TSC.2016.2519898).

- [45] D. SureshPatil, R. V. Mane, and V. Ghorpade. “Improving the Availability and Reducing Redundancy using Deduplication of Cloud Storage System”. In: *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. 2017, pp. 1–5. DOI: [10.1109/ICCUBEA.2017.8463856](https://doi.org/10.1109/ICCUBEA.2017.8463856).
- [46] A. Mahmud et al. “Enhancing Cloud Survival: Strategies Leveraging Diversity and Redundancy of Virtual Machine”. In: *2024 IEEE Cloud Summit*. 2024, pp. 157–162. DOI: [10.1109/Cloud-Summit61220.2024.00033](https://doi.org/10.1109/Cloud-Summit61220.2024.00033).
- [47] B. Mao, S. Wu, and H. Jiang. “Improving Storage Availability in Cloud-of-Clouds with Hybrid Redundant Data Distribution”. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. 2015, pp. 633–642. DOI: [10.1109/IPDPS.2015.47](https://doi.org/10.1109/IPDPS.2015.47).
- [48] P. Junghanns, B. Fabian, and T. Ermakova. “Engineering of secure multi-cloud storage”. In: *Computers in Industry* 83 (2016), pp. 108–120. ISSN: 0166-3615. DOI: <https://doi.org/10.1016/j.compind.2016.09.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0166361516301749>.
- [49] J. Riley et al. “A High-Availability Cloud for Research Computing”. In: *Computer* 50.6 (2017), pp. 92–95. DOI: [10.1109/MC.2017.182](https://doi.org/10.1109/MC.2017.182).
- [50] Apache Airflow. *Architecture Overview*. Accessed: 2025-08-18. 2025. URL: <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>.
- [51] R. E. Sperl and S. M. Chung. “Two-Step Anomaly Detection for Time Series Data”. In: *2019 International Conference on Data and Software Engineering (ICoDSE)*. 2019, pp. 1–5. DOI: [10.1109/ICoDSE48700.2019.9092751](https://doi.org/10.1109/ICoDSE48700.2019.9092751).
- [52] S. I. Vagropoulos et al. “Comparison of SARIMAX, SARIMA, modified SARIMA and ANN-based models for short-term PV generation forecasting”. In: *2016 IEEE International Energy Conference (ENERGYCON)*. 2016, pp. 1–6. DOI: [10.1109/ENERGYCON.2016.7514029](https://doi.org/10.1109/ENERGYCON.2016.7514029).
- [53] A. T. Williams and S. M. Chung. “Enhanced Multi-SARIMA Model for Anomaly Detection in Multi-Seasonal Time Series Data”. In: *2024 IEEE International Conference on Data and Software Engineering (ICoDSE)*. 2024, pp. 120–125. DOI: [10.1109/ICoDSE63307.2024.10829881](https://doi.org/10.1109/ICoDSE63307.2024.10829881).
- [54] I. Wanady, A. Viswanath, and K. Mahata. “Solar Forecasting for Power System Operator”. In: *2018 IEEE Electrical Power and Energy Conference (EPEC)*. 2018, pp. 1–7. DOI: [10.1109/EPEC.2018.8598379](https://doi.org/10.1109/EPEC.2018.8598379).
- [55] S. Dai et al. “Study on prediction of energy storage penetration rate for electric futures based on ARMA-GARCH model”. In: *2023 8th International Conference on Power and Renewable Energy (ICPRE)*. 2023, pp. 1204–1208. DOI: [10.1109/ICPRE59655.2023.10353831](https://doi.org/10.1109/ICPRE59655.2023.10353831).

- [56] L. Run, L. X. Min, and Z. X. Lu. “Research and Comparison of ARIMA and Grey Prediction Models for Subway Traffic Forecasting”. In: *2020 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*. 2020, pp. 63–67. DOI: [10.1109/ICICAS51530.2020.00020](https://doi.org/10.1109/ICICAS51530.2020.00020).
- [57] A. Gupta et al. “Very Short term Wind Power Prediction Using Hybrid Univariate ARIMA-GARCH Model”. In: *2019 8th International Conference on Power Systems (ICPS)*. 2019, pp. 1–6. DOI: [10.1109/ICPS48983.2019.9067611](https://doi.org/10.1109/ICPS48983.2019.9067611).
- [58] C. R. Bhat et al. “SARIMA Techniques for Predictive Resource Provisioning in Cloud Environments”. In: *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)*. 2023, pp. 1–5. DOI: [10.1109/ICCEBS58601.2023.10449163](https://doi.org/10.1109/ICCEBS58601.2023.10449163).
- [59] P. Rajan and K. V. Chandrakala. “Statistical Model Approach of Electricity Price Forecasting for Indian Electricity Market”. In: *2021 IEEE Madras Section Conference (MASCON)*. 2021, pp. 1–5. DOI: [10.1109/MASCON51689.2021.9563474](https://doi.org/10.1109/MASCON51689.2021.9563474).
- [60] Fingrid. *Reserves*. Accessed: 2025-08-27. 2025. URL: <https://www.fingrid.fi/en/electricity-market/reserves/>.
- [61] Fingrid. *mFRR, manuaalinen taajuuden palautusreservi*. Accessed: 2025-08-27. 2025. URL: <https://www.fingrid.fi/sahkomarkkinat/reservit/reservituotteet-ja-markkinoille-osallistuminen/mfrr-manuaalinen-taajuuden-palautusreservi/#hinnoittelu>.
- [62] Fingrid. *aFRR, automaattinen taajuuden palautusreservi*. Accessed: 2025-08-27. 2025. URL: <https://www.fingrid.fi/sahkomarkkinat/reservit/reservituotteet-ja-markkinoille-osallistuminen/afrr-automaattinen-taajuuden-palautusreservi/>.
- [63] Fingrid. *Sähköjärjestelmän reservit*. Accessed: 2025-08-27. 2025. URL: <https://www.fingrid.fi/sahkomarkkinat/reservit/sahkojarjestelman-reservit/#reservituotteiden-toimintaperiaatteet>.
- [64] Taylor G Smith. *pmdarima documentation*. Accessed: 2025-09-10. 2023. URL: <https://alkaline-ml.com/pmdarima/index.html>.
- [65] S. P. Sharma, J. R., and K. Deepa. “Forecasting India S&P BSE SENSEX and USA S&P-500 Benchmark Indices Using SARIMAX and Facebook Prophet Library”. In: *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2022, pp. 1523–1530. DOI: [10.1109/ICICCS53718.2022.9788127](https://doi.org/10.1109/ICICCS53718.2022.9788127).
- [66] A. P. Behera et al. “Predicting Future Call Volume Using ARIMA Models”. In: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2021, pp. 1351–1354. DOI: [10.1109/ICICCS51141.2021.9432314](https://doi.org/10.1109/ICICCS51141.2021.9432314).

- [67] B. Li et al. “Short-Term Load-Forecasting Method Based on Wavelet Decomposition With Second-Order Gray Neural Network Model Combined With ADF Test”. In: *IEEE Access* 5 (2017), pp. 16324–16331. DOI: [10.1109/ACCESS.2017.2738029](https://doi.org/10.1109/ACCESS.2017.2738029).
- [68] D. R. Osborn et al. “SEASONALITY AND THE ORDER OF INTEGRATION FOR CONSUMPTION”. In: *Oxford Bulletin of Economics and Statistics* 50.4 (1988), pp. 361–377. DOI: <https://doi.org/10.1111/j.1468-0084.1988.mp50004002.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1468-0084.1988.mp50004002.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0084.1988.mp50004002.x>.
- [69] H. Akaike. “Information Theory and an Extension of the Maximum Likelihood Principle”. In: *Proceedings of the 2nd International Symposium on Information Theory*. Ed. by B. N. Petrov and F. Csaki. Budapest: Akademiai Kiado, 1973, pp. 267–281.
- [70] R. J. Hyndman and Y. Khandakar. “Automatic Time Series Forecasting: The forecast Package for R”. In: *Journal of Statistical Software* 27.3 (2008), pp. 1–22. DOI: [10.18637/jss.v027.i03](https://doi.org/10.18637/jss.v027.i03). URL: <https://www.jstatsoft.org/index.php/jss/article/view/v027i03>.
- [71] The Apache Software Foundation. *Apache Airflow*. Accessed: 2025-09-22. 2025. URL: <https://hub.docker.com/r/apache/airflow>.
- [72] E. Dhib et al. “Impact of Seasonal ARIMA workload prediction model on QoE for Massively Multiplayers Online Gaming”. In: *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*. 2016, pp. 737–741. DOI: [10.1109/ICMCS.2016.7905664](https://doi.org/10.1109/ICMCS.2016.7905664).
- [73] Luosongzeren. “Research on Power Prediction Model of Solar Photovoltaic System Based on ARIMA Time Series”. In: *2023 IEEE International Conference on Image Processing and Computer Applications (ICIPCA)*. 2023, pp. 954–957. DOI: [10.1109/ICIPCA59209.2023.10257925](https://doi.org/10.1109/ICIPCA59209.2023.10257925).