Software Defined Networking Controlled Energy Optimization through Traffic Prediction on Microwave Access Network

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Gabriele Rivi**

Student ID: 994131
Advisor: Prof. Davide Martinenghi
Academic Year: 2022-23

Abstract

The work proposed in this Master's Thesis presents a new approach at achieving energy optimization on components deployed across a Mobile Access Network using Traffic Prediction.
Specifically, the study proposes a solution employing a probabilistic Deep Learning forecasting model that use nodes' historical data to provide expected future traffic patterns. The expected traffic is then used to propose a shutdown schedule for network component, ultimately reducing energy consumption.
This work also produces a working and customizable architecture, prototyping a pipeline to develop customer-centric solutions, identifying the building principles and aspects to consider to include AI-services successfully in a Software Defined Architecture.

Keywords: traffic prediction, deep learning, probabilistic forecasting, mobile access network, energy efficiency, resource scheduling

Abstract in lingua italiana

Il lavoro proposto in questa tesi presenta un nuovo approccio per ottenere l'ottimizzazione energetica su componenti distribuiti in una rete di accesso Mobile utilizzando la predizione del traffico. In particolare, lo studio propone una soluzione che impiega un modello di previsione basato su Deep Learning probabilistico che utilizza i dati storici dei nodi per fornire modelli attesi di traffico futuro. Il traffico previsto viene quindi utilizzato per proporre un programma di spegnimento per i componenti di rete, riducendo così il consumo energetico.

Questo lavoro produce anche un'architettura funzionante e personalizzabile, prototipizzando una pipeline per sviluppare soluzioni centrate sul cliente, identificando i principi costruttivi e gli aspetti da considerare per includere con successo i servizi di intelligenza artificiale in una Software Defined Architecture.

Parole chiave: predizione del traffico, deep learning, predizione probabilistica, rete di accesso mobile, efficienza energetica, programmazione delle risorse

Contents

1 | Introduction

In the past, mobile networks of medium to big sizes have been managed and configured manually, with error-prone and highly inefficient processes. With the modern-day upscaling of internet traffic and data transport relying on manual processes for networks that grow daily is no longer practical. In fact, by February 2023, over 432 operators have launched LTE Fixed Wireless Access services (FWA), and 523 operators across 155 countries have been investing in the field of 5G [22].
In recent years, a new approach to network management has emerged: Software Defined Networking (SDN). SDN divides the data plane from the control plane, improving the efficiency of configuration and planning tasks and enabling centralized management for better network utilization. Scalability, a critical feature in modern networks, is a key characteristic of SDN, ensuring that networks remain manageable regardless of their size [44]

The separation of network forwarding processes from its control enables process automation and performance optimization, concealing the underlying technology layers and architectures, and resulting in seamless operations. Regarding 5G networks, the most diffused transmission technology would be optical fiber, given its low-latency properties and low-energy requirements. However, it shows some drawbacks in physical instalment time and constraints, plus the inability to reach areas like islands or to create underwater pathways. To fill this void, microwave links have emerged as an alternative and complementary technology to fiber networks. These radio links support both point-to-point and point-to-multipoint communication with the possibility to adapt signal frequencies to the need. The links are easily installed and can allow for networks to reach areas that would not be reachable by cabled links, effectively extending the underlying optical fiber infrastructure. Some providers are preparing for a big shift, adopting an increasing number of these components, while others are already extensively using them.

One main drawback of the use of microwave technologies over optical fiber, is the high requirements in terms of energy. One big challenge in the context of SDN is to optimize not only traffic pathing and network configurations but also enable power reduction and energy saving when possible, as highlighted in recent industry studies [34]. Some of these components across a specific network might be underutilized in terms of capacity, and being able to automatically detect and allocate resources to the actual demand would improve the microwave access networks' energy consumption.

This thesis sets out to tackle this issue following a Machine Learning perspective. It specifically aims to develop an effective strategy to propose the shutdown schedules of individual carriers (transmission channels) within a single node. This becomes particularly relevant when these carriers are expected to experience periods of low traffic activity.
Specifically, the core objective of this research is to explore the potential of new Global Deep Learning architectures and evaluate their suitability for the task of forecasting multiple series. The main emphasis is the integration of probabilistic predictions, which brings a higher degree of adaptability and reliability, especially in industrial settings.
However, an inherent challenge with predictive algorithms is the risk associated with placing firm trust in model-generated outputs. To tackle this issue, the proposed solution centers on the idea of embracing uncertainty. By adopting this approach, the study aims to showcase the practical advantages of this innovative methodology in achieving valuable business outcomes.

1.1. Problem statement

In the evolution of modern mobile networks, Radio Links components enabling FWA are vastly employed (Ericsson counting more than 10 million transceivers globally [34]) and one of the next big objectives is to optimize the power consumption of these radio nodes. The flexibility of Mini Links (microwave transceiver) is granted by the possibility of performing carrier aggregation and other spectrum modulation schemes strategies, to save up energy when the traffic demand is relatively low [17].
To achieve improved performances in terms of energy savings the plan is to employ an AI model to trim down excess power consumption from what is allocated.
Specifically, the main components of the architecture to achieve this process are:

- **Traffic prediction service**: a microservice is designed to host a model capable of predicting the traffic on each interface within a mobile network for the next day. The length of the prediction output is determined at the business level and is set to one day. This service operates independently of shutdown scheduling because its output can be applied to various use cases that require traffic estimates. For example, it can be used for visualizing forecasted capacity requirements of network nodes, particularly for predictive maintenance purposes.

- **Deep Sleep scheduler**: a microservice following a strategic approach for planning the shutdowns of specific carriers within radio links. It leverages topology data from the network to make informed decisions. The primary goal of this strategy is to eventually shut down carriers that are expected to require less capacity than initially assigned when the node allows for carrier aggregation (presented in Section 2.1.3). The approach for this strategy is deterministic due to numerous constraints coming from equipment and protocols.

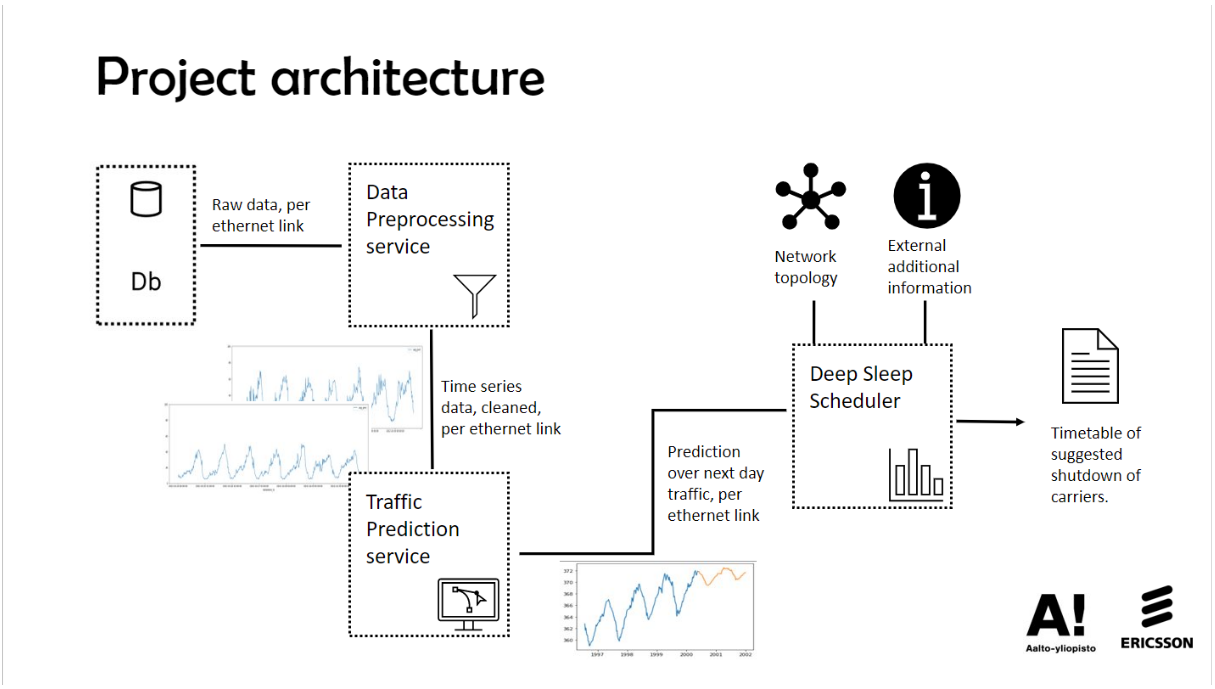

Project architecture
Db
Raw data, per ethernet link
Data Preprocessing service
Time series data, cleaned, per ethernet link
Traffic Prediction service
Prediction over next day traffic, per ethernet link
Network topology
External additional information
Deep Sleep Scheduler
Timetable of suggested shutdown of carriers.
Aalto-yliopisto
ERICSSON

Figure 1.1: Architecture sketch for the pipeline of Traffic predictor and Deep Sleep scheduler.

To successfully implement these services and achieve a pipeline as represented in Figure 1.1 there are some questions to answer and towards which the research should be oriented:

- **Is the available data useful?**
A thorough evaluation of the state of the data that can be used for this project needs to be executed, to understand which of the information about the radio can be used and if any preventive check needs to be conducted on the data. What is the target variable for the forecasting task? Are there any covariates that can be used to help and provide additional knowledge to the model? Is the data ready to use or need to go through a set of preprocessing operations?

- **Which DL architecture can best answer to this use case?**
  Deep Learning as a field progresses almost daily, but not all the novel and newly proposed architecture are successful and fit every case. Part of the research should focus on finding a subset of the best candidates to be tested and evaluated in this project, keeping in mind the compatibility of the architecture with the problem statement. Are the model output and the processing they employ understandable and interpretable in any way? What are the techniques and new approaches that these models implement that make them competitive and innovative in the task of time series forecasting?

- **What are the necessary steps to ensure that the whole architecture is flexible and can seamlessly integrate with the original product?**
  A study of the main characteristics of Software Defined Networking, microservice-based architectures, a reliable plan for the features required for the service creation, and how to leverage the results obtained in the ML study. All of these questions are necessary to make a concrete and functional product that bridges the initial understanding of the problem statement, goes through the requirements engineering elaboration for the Machine Learning model development, and ultimately ends in the deployment through microservice in a cloud-based environment.

1.2. Structure of the Thesis

Chapter 2 provides most of the needed knowledge to understand this project, in a hierarchical way: starting with the general contextual information and specifics of the field of application in Section 2.1, followed by a detailed description of the forecasting problem in Section 2.2, and ending with a mention of a previous attempt at tackling this problem in Section 2.3.

Chapter 3 presents all the experiments conducted within the project and the motivation behind the choices made. Specifically, the chapter will be divided into these subsections: data description and data preprocessing, clustering and data analysis, model training and selection.

Chapter 4 shows the results of the experiments concerning the development of a model and discusses the findings and motivation.

In Chapter 5, we provide a concise overview of the implementation details of the microservices that constitute the architecture when the model is deployed for inference, as depicted in Figure 1.1. Additionally, Chapter 6 offers insights into potential future developments.

2 | Theoretical foundation

This chapter presents an overview of all the relevant information needed to understand what is the scope of the project, what is the environment of the application and what are the design principles that are followed during the development.
Specifically, the background knowledge offered in this chapter can be further divided into three sections.
Section 2.1 presents the necessary information about the background of this project, relative to networking equipment and the architectures mentioned across this project (specifically Software Defined Networking and Microservice Architectures).
Section 2.2 presents the theoretical foundation of the most relevant topic in the field of time series forecasting, starting from a general overview of the classification of the various methods, and then presenting three models' architectures that are going to be compared and evaluated in this project.
Lastly, Section 2.3 presents a previous attempt developed to tackle the same problem that is considered in this thesis project.

The content this chapter provides serves to better frame the problem statement highlighted in the previous chapter. Particularly, the introduction of the machine learning aspects improves the understanding of the problem and defines the necessary steps to pursue a successful implementation, which is described in the next chapters.

2.1. Contextual knowledge about networks

This section aims to provide contextual information about the field of telecommunication networks. All the foundational concepts of this project regarding network infrastructure, paradigms and elements that are treated throughout the thesis are presented here with a high-level overview, to allow the reader to understand the domain of application.

2.1.1. Software Defined Networking

As briefly mentioned in the introduction, SDN represents an innovative framework through which is possible to achieve an automated and more efficient management of networks and their elements. Despite its increasing popularity, there is not yet a clear and general definition of what is SDN. The most acknowledged definition is the one given by the Open Networking Foundation (ONF), an industrial organization founded by network operators, service providers and vendors, and is rather short:

"Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable." [36]

The ONF is an open-source organization born to research and share information about Software Defined Networking, collaborating with industries and academia, and being in contact with OpenFlow Network Research Center (an organization with the same purpose, but created and populated by educational and research institutes).
Apart from this brief definition, SDN is defined also as dynamic, manageable, cost-effective and adaptable. These characteristics are compatible with what is progressively required by the development of the network communication industry, with the rapid advancement of 5G.

The main aspects characterizing the SDN architecture are the decoupling of control and data planes, and the programmability of the control plane.
Network infrastructures consist of various hardware components such as routers, switches and diverse communication links. These components are often vendor-specific and communication happens through specialized interfaces or with various protocols. These systems can be considered closed, and once deployed on a network, performing upgrades becomes fairly complicated. This process started when communication networks began to include components that are beyond simple packet forwarding elements. These elements, known as middleboxes, hinder the evolution of networks through a process known as "transport layer ossification" [16].

SDN architecture aims at removing intelligence from networking devices, routing towards a device-agnostic architecture, where all the management is hosted into a centralized controller. It is possible to identify three distinct layers in a general SDN architecture [44],[38],[33],[27]:

Infrastructure/Data Layer

Following what was already mentioned about middleboxes, the data layer contains all the network elements responsible for forwarding and switching packets, both physically and virtually. These elements are connected through disparate technologies, varying from copper wires and optical fiber to radio links, and are the ones to actually build the foundation of the network. In the SDN architecture, such elements are seen as forwarding hardware that is accessible through a unified interface, independent of the vendor or the specific protocol used by the elements. All the mediation is entrusted to the control layer.

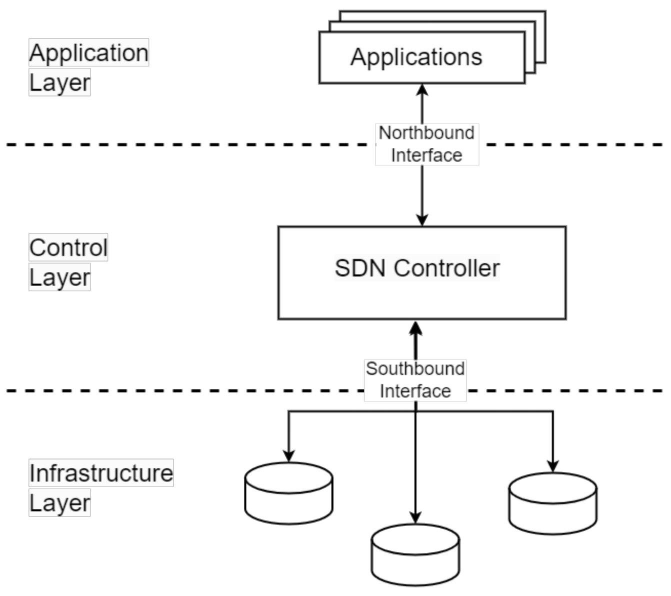


Figure 2.1: Sketch of the general architecture of SDN, following ONF.

Control Layer

The control layer consists of one (or multiple) controller units, managing the SDN correct functioning. They are mainly responsible for handling traffic flows and routing protocols. The control layer communicates with the infrastructure/data layer through a set of APIs which are referred to as **southbound interface** (SBI). The controller provides an interface accessible through a high-level language by the application layer, through which is then possible to implement management tasks and enable new functionalities by pushing policies to the data layer. Controllers are scalable and the offloading of the control capabilities from the infrastructure layer does not affect network performances. Controllers could be modelled to be either centralized (resulting in a possible single point of failure) or distributed, still offering SDN applications a unified view of the network.

Specifically, in the architecture proposed by Telecom Infra Project [2], to facilitate scaling of the system and specialization of components the architecture is hierarchical.
The bottom side of the control layer will consist of three logical elements, each referred to as a controller but responsible for a single domain of the components deployed across the infrastructure layer: Microwave, IP/MPLS and Optical.
All of the controllers are vendor-agnostic and contain the logic to handle all possible signals received by their specific domain, as highlighted in Figure 2.2.
Then, on the upper side, a hierarchical controller receives all the communications from the domain-specific controllers and communicates in a technology-agnostic way the demands coming from the application layer.

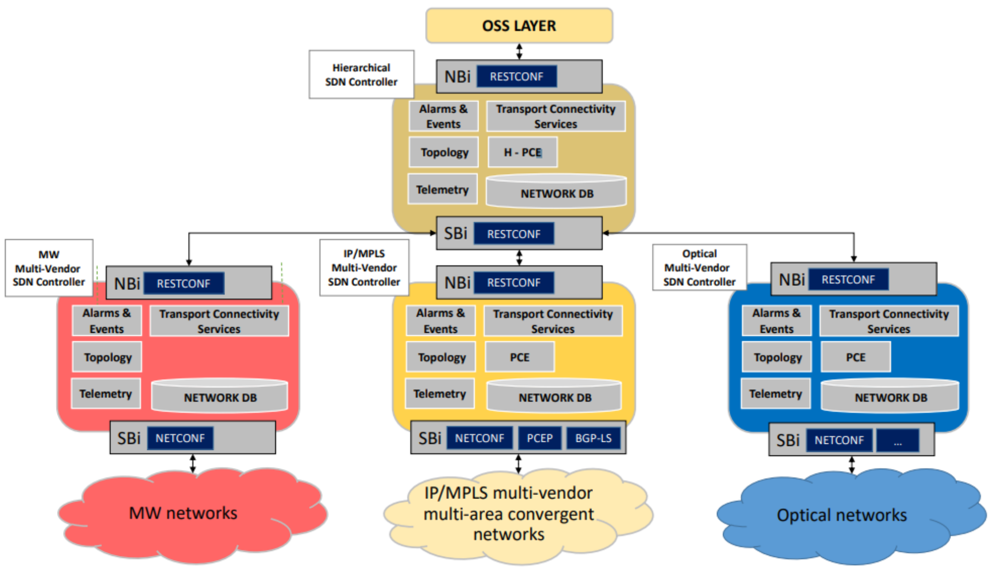

OSS LAYER
Hierarchical SDN Controller
NBi RESTCONF
Alarms & Events
Transport Connectivity Services
Topology
H - PCE
Telemetry
NETWORK DB
SBi RESTCONF
MW Multi-Vendor SDN Controller
IP/MPLS Multi-Vendor SDN Controller
Optical Multi-Vendor SDN Controller
NBi RESTCONF
Alarms & Events
Transport Connectivity Services
Topology
PCE
Telemetry
NETWORK DB
SBi NETCONF
SBi NETCONF PCEP BGP-LS
SBi NETCONF ...
MW networks
IP/MPLS multi-vendor multi-area convergent networks
Optical networks

Figure 2.2: Architecture for the control layer, as designed by TIP [2].

Application Layer

The application layer, positioned above the control layer, facilitates the utilization of SDN applications and software for network management and automation, employing a global view of the network. This layer makes it possible to programmatically organise the physical network through high-level programming, in a PaaS fashion. The application layer communicates with the hierarchical controller through a set of APIs which are referred to as **northbound interface** (NBI), making it possible to extract control level information or push changes regarding network policies. The hierarchical controller then forwards these demands to the domain-specific controllers.
Applications of SDN in networks enable flexibility, offering services for programmability of network components. The network configuration can be stored and modified without having to access the deployed modules individually, and information can be directly retrieved and stored to provide analytics and statistics. Adaptive and dynamic routing, network security, network maintenance and virtualization are some examples of the services that are easily made available through the SDN architecture.

2.1.2. Microservice-based Architecture

A Microservice-based architecture (MSA) is a software approach used to enable the development of big projects in a reliable and maintainable way, reducing overall complexity and time-to-market. To give a comprehensive definition:

"A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices" [31].

The key aspects of microservices and microservice-architecture are modularization, and most importantly reusability and replaceability. One component should be responsible for only one or a few tasks, and rely on the other interconnected components to be responsible for the other processes.

Shifting from a common Service-oriented Architecture (SoA) to a microservice-based one represents a step towards facilitating scaling in terms of availability, safety and speed [31],[8].
The main benefits of adopting this architecture are flexibility, having the possibility to update and upgrade the product when already released, enabling a rapid development of a first version to launch on the market, and then progressively adding services to the offer based on business requests, customer needs and internal decisions.
Shifting or starting with a MSA implies also some technical challenges, such as having to consider the whole infrastructure automation, the communication of these independent services and the overall debugging in a distributed setting. Not only on the technical side, but MSA requires some precautions also at an organizational level, having more teams interacting with each other and depending on each other's work [14].

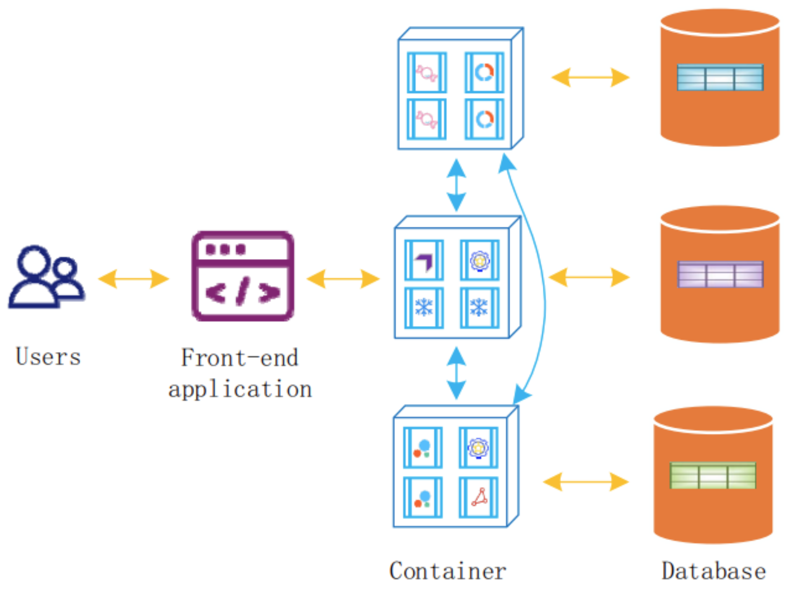


Figure 2.3: Sketch of a general MSA [28].

Figure 2.3 highlights the structure of an MSA, where the "server-side" is not a monolithic application but is composed of interacting microservices, each of them running in separate **containers**. The container is a lightweight approach for virtualization, which is able to sandbox the execution of processes increasing performances and limiting the complexity of management. The container wraps the application and the environment on which the application depends, and is standardized on a portable image [28]. This way, every application is prepared once in its image, and then is possible to have it run anywhere. All the containers are then managed and orchestrated by specific platforms, that vary based on the framework used to implement the system. Containers also enable incredible scaling capabilities, and an adaptive offer of the service w.r.t. the demand coming from the specific application, making an efficient alternative to static systems.

2.1.3. Microwave Radio Link

The family of components around which this project is developed is the *MINI-LINK 6000*, including both radio antennas and short/long haul solutions to create a microwave network backbone, ranging from 4 to 80GHz spectrum. This family of components introduces the capability of performing carrier aggregation, where multiple radio channels (carrier) are possibly managed and supported by a single MINI-LINK. This process makes it possible to double the capacity without modifying the tower footprint, hence the number of hardware installed is not increased. This feature is software-enabled, as well as signal modulation and a more efficient spectrum utilization, using a MIMO strategy (Multiple-Input Multiple-Output). MIMO is an established antenna technology, where multiple streams are used in the same channel, allowing for higher capacity [17], [18]. The introduction of these new approaches opens the path to a new strategy for power consumption optimization and power saving. Energy consumption for microwave components varies a lot depending on the traffic demand, and it is not expected to function at full capacity over a full day.

Having two or more carriers handled by the same hardware and the possibility to modulate the signals make it possible to put some channels in the so-called **"Deep Sleep"**, when the traffic can be easily handled by a single carrier.

With an AI-driven Deep Sleep strategy for the scheduling of carrier shutdown, it is expected that on a 5-year forecast, over a network of 5000 double carrier Mini-Link, the energy saving would sum up to 10 GWh, comparable to a 20% energy saving respect to normal functioning of the network, without affecting the performances. [17]

2.2. Machine Learning Methods for Time Series Forecasting

This section offers an exploration of the primary methods and historical evolution in the field of time series forecasting. It begins by introducing the key categorization aspects of the problem. Next, it delves into the early statistical approaches that were developed. Finally, it explores the application of deep learning in addressing prediction challenges.
Following this foundational understanding of the problem, it proceeds to examine and discuss the architectures of the three models under analysis in this study. This discussion covers their advantages and any newly introduced features.

2.2.1. Time-Series forecasting

The problem of time-series forecasting is fairly novel, originating not earlier than 1960. Before that, the go-to method was to employ a regression on time summing to a recurrent seasonality [10]. But before presenting detailed time-series methods, the first thing to understand is how to classify time-series data.
By definition, "A *time series* is a set of observations measured sequentially through time" [12]. This definition includes both continuous and discrete measurements. The majority of time series methods and applications are built around discrete time series, where measurements are either equally sampled over time, or aggregated to obtain equally-spaced time intervals in the measurements. The problem of aggregation often poses a challenge, having to decide which is the correct aggregation window for the specific task of forecasting.
The forecasting of a time series is one of the possible applications in the field of time series analysis, and its objective is to estimate the future values of the variable considered in the series [12].

2.2.2. Initial analysis

Before approaching the forecasting task, it is a good practice to perform an initial analysis to understand the general characteristics of the time series, and then proceed with the data cleaning. Other than plotting, which is the first tool to understand the nature of the data considered (and should always be the first tool to be used) there are two components that have to be analysed and be taken into consideration:

- **Seasonality:** identified as recurrent patterns that are equally spaced in time, usually yearly, monthly, weekly or even daily. This behaviour implies some cyclic recurrence of data [40],[12].

- **Trend:** identified as a sustained and maintained upward or downward movement of the series, resulting in growth or decline. There is not a satisfactory definition of this phenomenon, but can be broadly identified by a long-term shift in the mean of the series [40],[12].

2.2.3. Univariate vs Multivariate time-series forecasting

Generally, the aim of time series prediction is to build a model responsible for forecasting future values based on the observed data. To do so, either one or multiple variables can be considered for the task, and a specific distinction helps in identifying the problem [6]:

- **Univariate time series:** in these series, the observations collected reflect a single variable over time. Prediction of univariate time series are based solely on the available past data of the single variable.
- **Multivariate time series:** in multivariate time series analysis, the objective is to forecast the value of a specific variable. This prediction is not solely based on the historical data of the variable in question but also leverages information derived from observed correlation patterns between that variable and other variables considered in the model, often referred to as covariates. The addition of external variables may help the model in understanding how the behaviour of the target is influenced by the behaviour of the covariates.

2.2.4. Forecasting Methods

This section presents a brief history walkthrough and explanation of the most famous forecasting methods, retracing the emergence of new methodologies from the 1960 today.

Exponential Smoothing - ES

The ES methods first originated in World War II by a US Navy analyst, Brown. The method derives from the problem of developing a tracking model for the fire control information on submarines [21]. The original model was developed for continuous data, but was later extended and broadened to consider discrete data, then also trend and seasonality (in this case it gets the name of Holt-Winters method, from its creators). The method itself relies on forecasting future values as a weighted linear sum of past observations, where the weights are exponentially decreasing, following a geometric decreasing ratio [25].

AutoRegressive Integrated Moving Average - ARIMA

The ARIMA models are an extension and aggregation of far simpler models that were already available, respectively:

- **AR Model:** an autoregressive model is defined as:

  $$y_t = c + \phi y_{t-1} + \varepsilon_t.$$

  Where y_t is the value at time t, c is a constant, ϕ is a coefficient and $\varepsilon_t \sim N(0, \sigma^2)$ is a noise term. This model has an order, p, establishing how many prior samples are taken into consideration for the prediction:

  $$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i}.$$

- **MA Model:** a moving average model is defined as:

  $$y_t = c + \theta \varepsilon_{t-1}$$

  This model results as a linear regression of the current value w.r.t. previous observed noise term. Similarly to the AR model, can be expanded to order q and generalized as:

  $$y_t = c + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}.$$

These two models have been then combined in the commonly known Box-Jenkins models, from the name of the two authors, creating the ARMA models family [13]:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t.$$

The "Integrated" part of this model comes from the necessity of having a stationary time series in order for the model to work accordingly. ARMA models work with time series that do not show seasonality or trend characteristics, hence their mean and variation do not vary over time. The Integration is the step needed to transform any time series that is not stationary, into a stationary one. Specifically, the transformation process is called *differencing*, where the result time series is the d-th difference. This means that after d differentiation steps the series becomes stationary and can be forecasted through an ARMA model [32].

It is fair to note that all the exponential smoothing methods are analogous to some of the methods found in the Box-Jenkins models so these models are considered an extension of the previous work delivered by Brown, Holt and Winters [13].

Forecasting with a Deep Learning approach

The Deep Learning approach to Time Series forecasting is mainly relying on Artificial Neural Networks (ANN). Recent years have seen an incredible development of applications of ANNs in various fields, most notably in the field of imaging and computer vision, and in the last couple of years of natural language processing as well, with enormous breakthroughs like ChatGPT and GPT4. Given this popularity, a recent trend shows researchers starting to apply Deep Learning models to the task of time series forecasting. Even if in the past years has been difficult to see a pure DL model outperform a pure statistical approach to a problem, more recent and powerful models are able to perform way better than established statistical methods [29]. The results assessed in the fourth edition of the Makridakis Competition for time series forecasting methods show a shift in the trend, where the two best-performing models are an ensemble of statistical approaches and novel DL architectures [29].

From then on, with the aforementioned new discoveries in Sequence-to-Sequence (Seq2Seq) modelling (models that take a sequence-shaped input and return a sequence in output), there has been some successful application of ANN on forecasting, with increasingly better results.

In Artificial Neural Networks, the behaviour is inspired and similar to the one of human brains, where connections are built between "intelligent" components, the neurons, organized in layers and communicating with each other propagating information through weighted links. During each training iteration, the parameters of the network are tweaked and tuned, as well as the connections that are most important to the problem, and the weights themselves [25].

The advantage of training a deep learning model is that the model itself does not rely solely on the historical data seen, but the learned approximation of the functions allows for higher resiliency in cases where the data domain slightly changes (e.g., an interface that was highly utilized, becomes almost inactive). Having learned disparate patterns and relationships in the data, the deep learning model can successfully interpret and model the data, independently of its origin.

Specifically for time-based Seq2Seq problems, some ad-hoc architecture and components have been developed, like Recurrent Neural Networks (RNN), Gated Recurrent Units (GRU) and Long-Short Term Memory (LSTM). These are specialized Neural Networks where pieces of information are stored in the hidden layers in specific components and are kept into consideration to "tune" and predict the output with a higher accuracy, creating a combination of the current input and stored information [26].

More than that, models like the Transformer by Vaswani et al.[42] have brought again up the level of predictions and the accuracy, with the introduction of the mechanism of Attention, consisting in aggregating temporal features in a dynamic weights sum, explicitly telling the model on which part of the input sequence to focus [42],[26].

Generally, previous work has proven the possibility of employing DL models to the task of time series forecasting, but however promising this may appear, there are certain drawbacks to consider. For example, these models take time and resources to train, and are basically black box objects, limiting explainability and interpretability. They are suited for a Global approach to the problem of prediction, but in some cases they still might be performing worse than more classical statistical methods.

2.2.5. Local vs Global approach

In the context of multiple time series forecasting, two main approaches are acknowledged to address the task. The more prevalent use case in literature is the problem of forecasting a single time series, either univariate or multivariate. In this case, the dataset is characterized by a rather long sequence of data for a specific variable over months or years, at different granularity (e.g. the price variation of a stock). Differently from this case, in the problem of having multiple time series is common to have thousands of time series representing the same time-dependent variable, but for different objects (e.g. the sales per specific product in a shop). The series are characterized by their similarity or some form of relatedness. In this scenario, every single time series needs to be predicted by either one or multiple models [30].

The two approaches mentioned differ in how they tackle the problem. The **local** methodology considers each time series as an independent object and will produce a model for each one to be forecasted. The **global** approach produces a single model able to handle all the different time series, learning recurrent and common patterns and forecasting each of them. Local models like exponential smoothing and ARIMA, mentioned in Section 2.2.4, are able only to learn parameters to account for a single seasonality (on a time series) and not others, so ideally the proportion is 1 to 1 with models and time series. They build the prediction solely on the past observation of the single time series and rely a lot on the patterns that are showing.

With the global approach, a single model (usually coming from Deep Learning) is trained with multiple different time series, and thanks to this is able to learn relationships between patterns and catch on to all the different seasonalities and characteristics in the different time series. This capability is peculiar to Neural Network models, where learning the optimal parameters takes time but results in a general model able to reuse and employ at best every detail that has been trained on and learned to generalize, creating an accurate approximation of the hypothetical function assumed to be generating the input. Both approaches present benefits and disadvantages: notably, the local approach has a bigger footprint and is strongly dependent on the time series' past data, so doesn't account for sudden changes or drift in the data itself. That also means that these models would be unable to predict new time series considered by the system, where past data would not be available. This is known as the "cold start" problem.
The global approach on the other hand requires a lot of data and computational power, resulting in longer time requirements, to train and learn patterns. Nevertheless, the model resulting from this approach is able to treat different time series independently of their origin (as long as they are not completely uncorrelated to the data present in the training set) and is also more resilient to drifts and changes in the data domain. Also, it would be able to address new time series considered by the system when the necessary amount of input samples are collected for that new component.

2.2.6. Models Literature

To answer the research question related to the exploration of new deep learning approaches and architectures, the first analysis that has to be performed is about the peculiarity of these new models, and how they are able to bring innovation to the task of time series forecasting.
In this section, the papers presenting three new architectures are reviewed in their structural features.
All the architectures analyzed support both deterministic and probabilistic forecasting,

and have obtained consistent results in literature.
Three different models are presented: Temporal Convolutional Network, N-Beats and N-HiTS.

Temporal Convolutional Network - TCN

In the field of sequence modelling, and specifically time series forecasting, the go-to approach for most cases in the past has been to use Recurrent Neural Networks, LSTM and GRU-based architectures. Simultaneously another approach has become widely recognized, especially in the field of audio synthesis and natural language processing, the Temporal Convolutional Network.
Recently published work has shown successful results in the employment of this methodology also in the problem of time series forecasting, in various applications, such as "TCN for an effective weather forecasting using time-series data from the local weather station" [24]. The term TCN does not identify one specific architecture, but in general, refers to the approach of using Convolutional Neural Network with other expedients to perform sequence-to-sequence modeling.
Various interpretations of this approach have been published and recognized as functioning, such as the one from Chen et al. [11], or in the work by Wan et al. [15] where the architecture was proven to be effective also in the task of multivariate time series forecasting.

Generally, the aforementioned architectures share some common structural details, but the specific implementation that is used in this project and that will be explained in this section refers to the work of Shaojie et al. presented in the paper "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling" [4].

In the paper, a generic TCN architecture is presented and then compared across a set of different tasks with more commonly known Seq2Seq architectures, since there have been a lot of studies and comparisons about fully recurrent networks or hybrid models, but not many about fully convolutional networks used on the problem of forecasting.

There are mainly two aspects characterizing the generic TCN architecture:

- **Causal convolutions:** differently from what would happen in using simple convolutional layers, with causal convolutions there is no information leakage from the future to the past. A *causal convolution* ensures that each output at timestep t does not depend on any future timesteps. An example is shown in the picture.

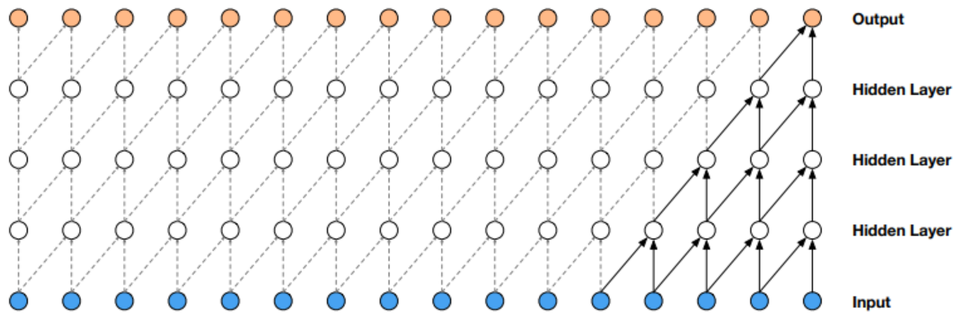


Figure 2.4: Example of input propagation across hidden layers with a causal convolution [35].

Nevertheless, if the architecture were to employ simply causal convolutions, it would be affected by problems related to difficulty in accessing a longer history size, resulting in having too deep a network or too large filters in order to be able to consider longer input sequences. To help overcome this issue where causal convolutions are able to look back only with a size linear with the depth of the network, *dilated causal convolutions* can be used to enable the use of an exponentially large reception field. The dilation is equivalent to introducing a lag between two successive filter steps. A dilation with coefficient $d = 1$ is equivalent to a simple causal convolution. Using larger coefficients allows the output to represent a wider input window, hence expanding the receptive field.

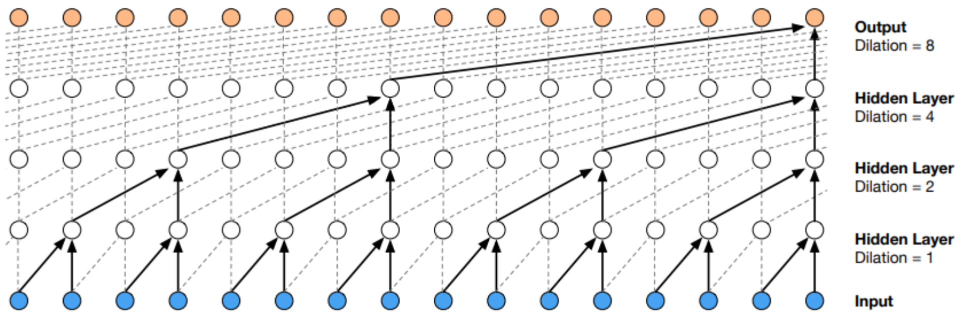


Figure 2.5: Example of input propagation across hidden layers with a dilated causal convolution [35].

This expedient enables TCN to overcome problems that are common when using Recurrent Architectures, having the ability to process input sequences **parallelly** since a prediction at a step t doesn't depend anymore on the prediction at step $t-1$ and the same filter is used in every layer.

- **Input can be of any length:** input of any size is mapped to an output sequence of the same length. This is obtained by using a one-dimensional fully convolutional network where each layer has the same length as the input one, adding zero padding to keep the subsequent layers the same size as the previous ones.

The presented architecture solves also other characteristic problems of LSTMs and GRUs. During the training process for these models memory requirements can drastically increase due to the storing of partial results, but in TCN having filters shared across the layers makes the backpropagation step depend only on the network depth.

Another detail of the architecture is the addition of a Residual Connection, mainly to stabilize the network in cases where it grows to be very deep. Instead of using a convolutional layer, a residual block is used. Every block consists of two layers of dilated causal convolution and two non-linearity (ReLU). The convolutional filters weight normalization, and a spatial dropout after each dilated convolution. Then, the input is added to the result of the convolutions after a $1x1$ convolution to ensure that the shapes are maintained.

NBeats

The N-BEATS model architecture as presented in the paper "N-BEATS: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting" [37] was first proposed in 2020 as an innovative way of dealing with the task of time series forecasting, without using any classic and established time series component or tool. It was the first model to be registered in literature to outperform statistical approaches on various datasets, like the M3 and M4 datasets from the Makridakis competitions.

The architecture proposed and related work is based on two principles:

- **Deep Neural Architecture**: provide a simple and generic architecture that could be reproduced and expanded, performing as well as or better than classical statistical approaches, on a different set of time series tasks, independently of the domain
- **Interpretable model for Time Series**: allow for an additional contribution to the explainability and interpretability of Deep Learning models allowing DL practitioners to better understand outputs, in a way similar to how it was made possible with the STL (seasonality-Trend-Level) approach.

The description of the architecture starts from the basic block. Every block is built the same way and is characterized by having a *fork schema*. The block accepts its respective input (for the first block of the model, the respective input corresponds to the general model input), and emits two outputs, the block's forward forecast $\hat{y}$ and the block's estimate of the input received, which is usually referred to as *backcast*, $\hat{x}$.

The single block consists of two parts, first a fully connected network producing both the forward and backward predictors of expansion coefficients, and a second part with two basis layers that accepts the respective forward and backward expansion coefficient. These two basis layers, projecting internally the coefficient, produce the backcast and forecast outputs as defined above.

How both of these outputs are used is explained by a double residual stacking, where the backcast output of every block is used to trim the output that the next block is seeing, removing unnecessary information thanks to the approximation provided by the previous block, and all the forecasts are used to approximate the final forecast. For a generic block l, this is how the backcast and forecast outputs are contributing:

$$x_l = x_{l-1} - \hat{x_{l-1}}.$$
$$\hat{y} = \sum_l \hat{y}_l.$$

Every previous block removes a portion of the signal that can be approximated well, easing the forecasting job down on the stream of blocks. Every forecast generated by a block is firstly aggregated at a stack level, then also at the general network level, following a hierarchical approach. Following this hierarchy enables a higher level of interpretability when the two basis layers are fixed for a specific choice, like in the case of trend and seasonality interpretability.

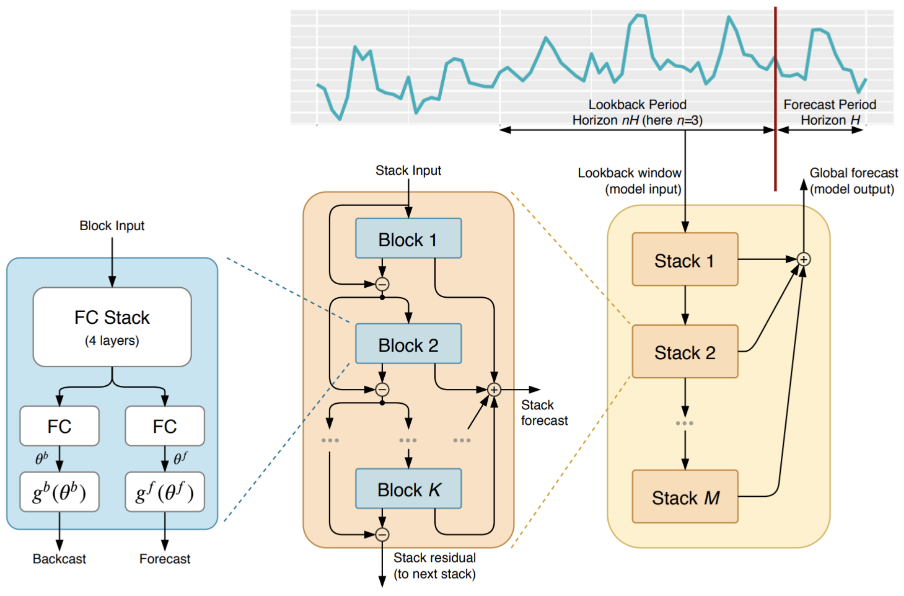


Figure 2.6: N-BEATS architecture. Block (left), Stack (middle) and general architecture (right) [37].

Regarding the second principle mentioned above, for interpretability the paper proposes two configurations for the architecture:

- **Generic architecture:** the two basis layer functions are simply linear projections of the previous layer output.
- **Interpretable architecture:** it is built starting from the architectural approach shown in Figure 2.4 and adding a structure to the basis layer at a stack level. Following the approach proposed with the STL decomposition for the study of time series, these two configurations aim to improve interpretability of the forecast.
  - **Trend model:** trend can be identified with a monotonic function. In this sense, the basis layers are proposed to be a small polynomial, a function slowly varying across the forecast window. In this sense, the forecast output for a block l in the stack s becomes:

$$\hat{y}_{s,l} = \sum_{i=0}^{p} \theta^f_{s,l,i} t^i.$$

  - **Seasonality model:** seasonality is a recurrent, cyclical fluctuation of values in a time series. The basis layers are hence proposed to be such as a periodic function, in this case, a Fourier series. The output for a block l in a stack s

then becomes:

$$\hat{y}_{s,l} = \sum_{i=0}^{H/2-1} \theta^{f}_{s,l,i} \cos(2\pi i t) + \theta^{f}_{s,l,i+\frac{H}{2}} \sin(2\pi i t) \,.$$

The interpretable architecture consists of two main components: the trend stack and the seasonality stack. By incorporating double residual stacking and the forecast/backcast fork building block principle, the trend component is removed from the input window before being passed to the seasonality stack. This approach enables the generation of separate and interpretable partial forecasts for both the trend and seasonality components.

N-HiTS

N-HiTS is a novel architecture proposed in the paper "N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting" [9]. The authors, with this paper, aimed at improving the discoveries presented with the proposal of N-BEATS, specifically addressing the problem of volatility of predictions in the case of long-horizon forecasting, improving as well the computational cost required by the model.

The improvements proposed in the paper are mainly concerning two additions that contribute to achieving higher accuracy and lower computation:

- **Multi-Rate Signal Sampling:** building over the forked architecture of the basic block explained in Section 2.2.6 and showed in Figure 2.6, the input of each block is processed directly by a MaxPool layer of kernel size k_l to redirect the focus of the specific block on components of a specific scale.
  Having a larger k_l will result in considering higher-frequency sequences from the input, whereas smaller k_l will help in considering finer grain patterns in the time series input. Effectively this means that each Fully Connected component in the block will be responsible for a different input sampling rate, maintaining the overall receptive field and helping in considering longer horizon forecasts, hierarchically addressing the different scopes of the input in different blocks.
  This process also helps in reducing the dimensionality of blocks' inputs, limiting required memory and computational cost, and overall having a lower number of parameters. After the multi-rate signal sampling provided by the Max Pool layer, a classical non-linear regression is used to produce the forward and backward interpolation coefficient, as seen in N-BEATS.

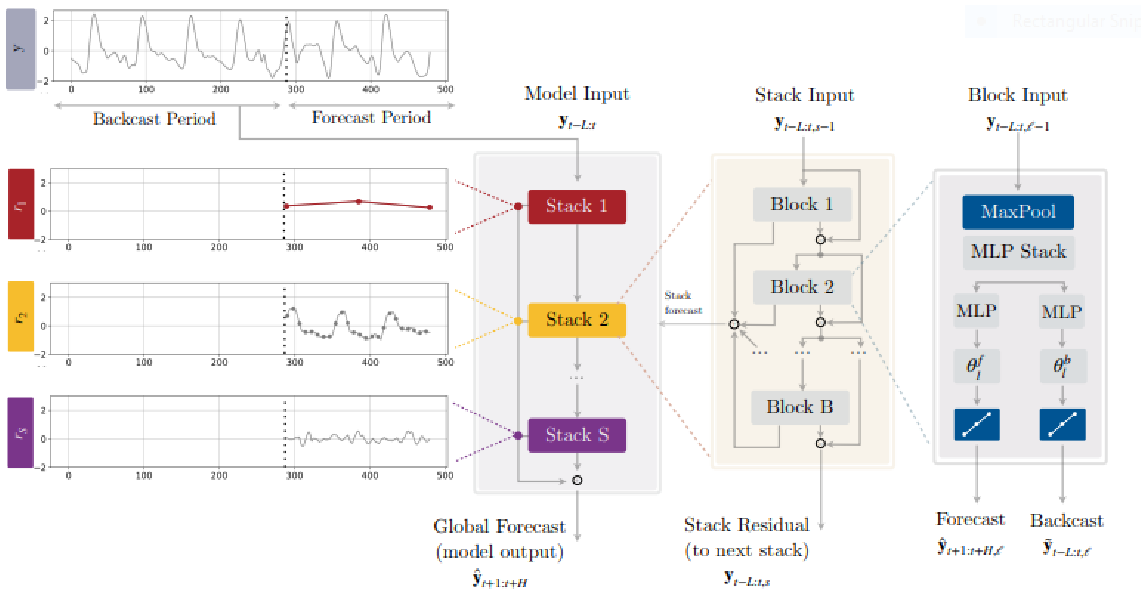


Figure 2.7: N-HiTS architecture [9]. Focus on hierarchical interpolation (shown on the stacks' outputs of the network on the left), and the multi-rate sampling with the MaxPool layer.

- **Hierarchical Interpolation:** generally, both in N-BEATS and other architectures such as Transformer-based ones, the cardinality of the prediction is the same as the horizon H. This immediately results in growing computing requirements and complexity with growing horizons. Temporal interpolation addresses this problem by defining the forward and backwards interpolation coefficients in terms of a *expressiveness ratio*, r_l, controlling the number of parameters per unit of output time. Then to compute the forecast for the horizon H and resort to the original sampling rate the basis layer is used accordingly. Hierarchical interpolation is used by tuning parameters r_l and k_l across the whole network architecture. Blocks that are closer to the input will have smaller r_l and larger k_l, focusing on the "big picture" of the input time series, with more coarse representations of the input, and deeper blocks will focus on more peculiar and higher-frequency patterns. The paper's authors propose to use exponentially increasing expressiveness ratios r_l to cover a wider range of frequency bands, still keeping down the number of network parameters.
  The backcast residual is still used as in N-BEATS to trim down the next block input, forcing the focus to be on signals that are not considered by the previous blocks.

2.3. Previous work on Deep Sleep scheduler

Previous efforts have been made to tackle the challenge of distributing resources across microwave components within a mobile radio network. In their thesis work, published in 2022 at Chalmers University of Technology, Gothenburg, Sweden, Adam Frithiofson and Marcel Vacante explore how to solve this problem by working in close collaboration with Ericsson [19].

The underlying motivation and problem statement are quite the same as the ones driving this thesis project, but the used approach differs. The authors designed their solution to this problem as a software controller responsible for managing all radio elements of a network, implementing a deterministic policy, which is intended as a different component enabling flexibility and adaptability thanks to the possibility of interchanging it.

The controller is the software enabling the modulation of the power state of the radio links, actuating the shutdown/power-up of the components, aiming at reducing energy consumption while minimizing the impact on the network's users. The proposed controller design follows four principles:

- *Congestion minimization:* take actions in order to minimize congestion in case of demand exceeding capacity available
- *Minimization of power state changes:* radio links are physical hardware components, a too aggressive policy implementing frequent power state changes could impact negatively the hardware's overall lifetime
- *High scalability:* network dimensions exhibit high variability and can extend across thousands of nodes. Therefore, the controller must be scalable to address power allocation demands irrespective of the network's scale
- *Fault handling:* introducing the controller element in the network should not impact in any way the correct functioning of the network, solving problems arising by its action on its own (following the principle of self-healing)

The architecture of the software controller follows a distributed approach, where an instance of the controller is deployed at every node of the network and is responsible for every radio link connected to that node, running on separate hardware from the node's own.

The policy is an arbitrary set of deterministic rules enforced by the controller. Generally, the policy should state that if the demand corresponds to an output of 50%, then the amount of active carriers should be enough to answer to the demand. Assuming that there are two carriers equally capable, then one of them should be on and the other off. Two different policies are explored in their project:

- **Reactive policy:** in this first simple version, only instantaneous variation on capacity are used to adapt the capacity. There are clearly flaws, where capacity cannot vary fast enough in order to answer to cases where capacity is exceeded, and could be unstable where capacity fluctuates near the threshold of activation of multiple carriers. The introduction of a threshold helps to avoid these problems in cases where the capacity shifts slowly.
- **Hybrid policy:** starting from the assumptions of the reactive policy it uses also historical data. Exploiting regularity of patterns (seasonality), two windows are considered: a *lookbehind* window looking at immediate past utilization, and a *lookahead* window where is considered the same upcoming interval of interest but in the previous week.

This overall system and especially the policies were put to the test thanks to the use of a simulator for a week. The overall energy consumption reduction registered was around 39%, showing effectiveness of the solution when put to the test.

The authors, after presenting the work, recognize that the project could be extended with more complex policies and elaborate strategies to manage the power allocation, suggesting as an example that using a Machine Learning approach could provide higher accuracy in case of more complex patterns.

3 | Methods and Model implementation

As emphasized in the thesis introduction, the problem domain pertains to Telecommunication. The utilization of Machine Learning in conjunction with SDN architecture is a relatively novel concept within the company. As such, the primary objective of this project is to create a prototype of the system shown in Figure 1.1.
This foundation can then serve as a basis for future development, especially as the supporting architecture for such services advances and becomes better equipped to accommodate more extensive projects.
This chapter contains all the exploratory analysis performed on the available data in order to understand its value and possible use, as explained in the first research questions. Furthermore, Section 3.6 shows all the steps endured for the development of a working model for the task of forecasting, able to answer the second research question highlighted in Section 1.1 specifying the need to find a probabilistic model to employ in this use case.

It is important to note that all the decisions undertaken in this project have been strongly influenced by the characteristics of the environments and platforms used for the development operations. Using a dataset consisting of real data sensible to customers, and being protected by various security levels has often slowed down the development, especially due to the limited freedom of deciding where to carry out all of these operations.

The main limitations and details about the tools used and resources available for development are illustrated in Section 3.1.

3.1. Environment and limitations

This brief section presents the tools used and obstacles met during the implementation phase and while carrying out the experiments explained later on in this Chapter.
Many of the choices taken to carry out experiments were conditioned by the environment and resources available for them.

The whole project has been developed using Python (version 3.8.10).
The exploratory analysis and the model implementation have been developed through the use of jupyter notebooks on company-hosted Jupyterhub servers. The microservices have been developed in Python, and prepared to be deployed through Docker.
The first part of the experiments has been completed on a dedicated analytics platform of Ericsson, offering direct access to a data lake storing the needed data that will be described in Section 3.2, accessed through JupyterHub. The specific resources available per user on this platform are shown in Table 3.1

Table 3.1: Resources per user on analytics platform

CPU model	8x Intel Xeon Processor (Cascadelake) @ 2.30 GHz
CPU cores	16 (actual 2, because of vCPU)
RAM	14GB

The database tables are accessed through a parallel processing database engine, Impala, using SQL queries. One of the biggest issues of operating on the analytics platform emerged already from getting access to the data. Specifically, for how the platform is hosted resources are virtualized and dynamically assigned to users upon access.
These specifics (Table 3.1) hardly correspond to the resources needed for a data science project, and apart from using tools like PySpark when possible to parallelize operations on data, the platform is not suited for the jobs of this entity. Overall, the jobs related to data preprocessing took a great amount of time to be done, and some of the tasks needed to be further divided to ease the computational burden on the system.
This was made abundantly clear when starting the experiments on model training. In this case, computing resources were not nearly enough to run training, either because of saturation of RAM memory or insufficient computing resources.
Being a virtualized service, the CPU resulted as vCPUs (virtualized CPU) and many of the supporting libraries used for the experiments did not support parallelization on mul-

tiple CPUs. In this case, where it was not possible to achieve parallelization, the available cores resulted in 2 instead of 16. Not to mention, there was no access to a GPU.

The main obstacle originated from the policies of data protection, avoiding the export of the data outside of the mentioned platform.
Receiving the clearance to export a subset of processed and anonymized data required time, but once received, the experiments on model training were conducted on another cluster owned by Ericsson, dedicated to data science. Resources for this platform are showed in Table 3.2.

Table 3.2: Resources on data science platform

CPU model	8x Intel Xeon Processor (Cascadelake) @ 2.30 GHz
CPU cores	16
RAM	32GB
GPU	NVIDIA Tesla T4
vRAM	16GB

On this platform, accessed as well through JupyterHub, the user is allowed to customize the needed resources.
The experiments are run on an instance counting 32 GB of RAM memory and having access to one T4 Tesla NVIDIA GPU, which made the training and hyperparameter optimization much faster.

Some additional details about libraries and packages used for the model experiments are mentioned in Section 3.6 and in Section 4.2.

For the data access, cleaning, selection and processing the main tools used are Impala, pandas and occasionally PySpark to speed up parallel processing jobs.

3.2. Data description

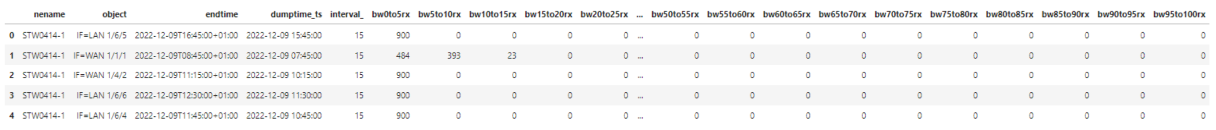
The data used to create the dataset for training and testing the models used for the prediction forecast is provided directly by Ericsson.
The data is coming from a mobile operator and is relative to a network of medium sizes.

The data has been anonymized before being used.

The data has a granularity of 15 minutes and is collected by an Ericsson Management Solution.
The main purpose of the raw data used in this project is to be collected and analyzed to have meaningful traffic counters, on which is possible to retrieve the network's status and statistics enabling specialized and focused maintenance and performance analysis. This data is generally referred to as Performance Management data, or *PM counters*.

These particular counters are stored and organized in databases, and the PM of interest for our use case is the Ethernet Link Capacity Utilization. This counter measures the **percentage of capacity that is required by a specific ethernet link that has as endpoint one microwave node**. In other terms, this PM measures the volume of traffic passing through the node in terms of capacity requested from the radios.

The database table in which the PM is stored contains, between the other fields about identifiers and timestamps, twenty columns corresponding to buckets of utilization (expressed in percentage). Every bucket of utilization has a range of 5% and contains the number of seconds that the link has spent in that specific percentage bucket.



	nename	object	endtime	dumptime_ts	interval_	bw0to5rx	bw5to10rx	bw10to15rx	bw15to20rx	bw20to25rx	...	bw50to55rx	bw55to60rx	bw60to65rx	bw65to70rx	bw70to75rx	bw75to80rx	bw80to85rx	bw85to90rx	bw90to95rx	bw95to100rx
0	STW0414-1	IF=LAN 1/6/5	2022-12-09T16:45:00+01:00	2022-12-09 15:45:00	15	900	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	STW0414-1	IF=WAN 1/1/1	2022-12-09T08:45:00+01:00	2022-12-09 07:45:00	15	484	393	23	0	0	...	0	0	0	0	0	0	0	0	0	0
2	STW0414-1	IF=WAN 1/4/2	2022-12-09T11:15:00+01:00	2022-12-09 10:15:00	15	900	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	STW0414-1	IF=LAN 1/6/6	2022-12-09T12:30:00+01:00	2022-12-09 11:30:00	15	900	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	STW0414-1	IF=LAN 1/6/4	2022-12-09T11:45:00+01:00	2022-12-09 10:45:00	15	900	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 3.1: Snapshot of the PM counters table.

Following this definition, the sum of every cell of these bucket columns on a single record in the table should sum up to a total of 900 seconds (15 minutes). Other columns are the timestamp of the measurements (both in UTC and local format), and the identifiers of the interface ("nename" - Network Element name, "object" - specific WAN interface of the network element).
The data considered for the dataset creation has been collected starting June 2022. Although the collection of these PM counters started in January of the same year, the data before June is not consistent enough to be considered useful for dataset creation, having a lot of interfaces with missing data, duplicates, and corrupted records.
This is the schema of the database table:

- *nename* - identifier for the network element;
- *object* - interface of a specific WAN interface - specifying a link - for that network element;
- *endtime* - timestamp in local time;
- *dumptime_ts* - timestamp in UTC format;
- *interval* - minutes passed from last measurement to current one;
- *bw0to5rx* - seconds spent in a bandwidth utilization from 0% to 5%;
- *bw5to10rx* - seconds spent in a bandwidth utilization from 5% to 10%

  ⋮
- *bw95to100rx* - seconds spent in a bandwidth utilization from 95% to 100%

Where "bw" stands for bandwidth.

3.3. Data preprocessing

The raw data as collected and stored is not directly employable for the task of forecasting.

The aim for this specific use case is to model the data to fall under the case of **univariate forecasting**.
Reminding the definition of univariate forecasting presented in Section 2.2.3, the aim is to predict future values of a variable using its current and past values [7], whereas in *multivariate forecasting* the aim is to predict future values of a variable using information from other variables, called covariates.
The data preprocessing section includes all the operations that have been employed to study the data, understand the possible use and how to treat it in order to have a clean sample of time series to be used in deep learning training.
The main steps highlighted in this process are:

- **Data shaping**: shifting from disaggregated data to an univariate time series.
- **Data partitioning**: reducing the whole table into smaller and more manageable structures.
- **Data pruning and imputation**: cleaning operations on the time series to remove errors.

3.3.1. Data Shaping

When the data is fetched from the database table, the schema counts more than 20 different columns, due to how the data is collected as Performance Metric.
Every record in the table refers to a specific ethernet link interface at a specific time across a day. The data is sampled at a 15-minute frequency, so every day counts 96 samples registered to the database for every interface functioning over the network. The query fetching data from the database is also used to perform the shaping, manipulating the cells of the records referring to the bucket of percentage utilization, in order to obtain a single target variable, following this formula:

$$avg\_perc = \frac{\sum_b (b_{max\_percentage} * b_{seconds})}{900}$$

Where:

- b: specific bucket of percentage capacity utilization (0-5%, 5-10%, etc.)
- $b_{max\_percentage}$: right-side limit of the bucket in question. This follows a worst-case approach, where every bucket is assumed to be involved with the maximum possible capacity in question, to avoid underestimating the capacity later on in the scheduling of the shutdowns (the granularity of the data does not allow knowing how the seconds are exactly distributed over the 15 minutes frame)
- $b_{seconds}$: the content of the bucket cell, i.e., the number of seconds spent by the interface in that capacity utilization percentage
- 900: since the measurements have a granularity of 15 minutes, every entry represents the distribution of the seconds over the percentage utilization of the past 15-minute interval, hence 900 seconds

This formula is computing a *weighted average over the seconds*, using every percentage utilization bucket, resulting in an **average percentage utilization** that can be used as a variable for forecasting.
The schema of the resulting data structure is as follows:

nename	object	dumptime_ts	avg_perc

The field **"avg_perc"** is the label for the forecasting task.

3.3.2. Data Partitioning

The data collected span over various months but cannot be loaded all at once to be processed, given the limited resources offered by the platform allowing access to the data itself. Having limited resources granted to the user, the computational power and amount of memory requested to load all at once the data in memory are far too great to enable this kind of job, even using tools for parallel processing such as PySpark.
Following a business decision, the data is partitioned by biweekly ranges, starting on Monday, according to international standards. The day on which the split is made does not affect the outcome and the results, as such information is irrelevant in this problem, but it is simply for easing the process of creating the dataset.
This design results in having as data some time series of 1344 samples, expecting one full day of prediction as output, translating to 96 samples (given the 15-minute granularity collection). Having possibly longer sequences available as input allows for trying different solutions in a later phase when the number of days used as input by the model can be tuned in order to achieve better performances.

Processing the data in this way also creates subsets that can be analyzed to study the distribution of the data and peculiarities of the whole dataset.

3.3.3. Data pruning and imputation

One last step is performed on the data to be considered ready.
Since data collection is an empirical process, where multiple elements are coordinating to send information to central authorities and then collected in a database, in some steps of this chain the data might be corrupted or go missing. The known and more often observed types of corruption of data in this process are:

- **Missing records:** timesteps recording missing sporadically or in an extended interval of time, due to miscommunications and/or hardware malfunctioning. Even maintenance and discontinuation of specific machines can cause this error;
- **Duplicate data:** some recordings appear twice or multiple times, probably for error in the write-to database;
- **Corruption:** in every record, representing a sum of the seconds over a 15-minute window, the total should be 900 seconds. In some cases, the numbers do not add up to this sum, meaning that corruption happened in the measuring process. In some cases, the corruption might be shown as an unexpected value registered, but

these cases are far more complex to handle because it is not possible to distinguish between an erroneous measurement and a real spike in the network traffic (i.e., if an unexpected peak in utilization is registered, shifting a lot from the usual pattern of the series, it is hard to label it as a malfunction in the data collection process or if it is simply a really unusual pattern showing in the data).

The preprocessing step aims also at solving these issues, starting from the partitioning highlighted in the previous paragraph. As a business decision, all the measurements of a single interface over the two weeks cannot have more than 10% missing records. This first cleaning comes from the decision to consider possible partial malfunctioning over the measurement collection process, in order to not lose information that might be useful and representative.

The second expedient employed to address the problem of missing data is the imputation of missing records. Every sequence, as previously mentioned, is expected to be composed of 1344 steps representing two full weeks, but since shorter sequences are allowed, to avoid excluding an excessive number of interfaces, the missing data is filled.
This is also why sequences are thresholded at 10% of missing elements so that they are still consistent and the proportion of filled records is considered acceptable and would not bias too much the dataset.
To perform this imputation, every sequence is resampled using the column referring to the timestep, filling all the 15-minute intervals records that are missing, in order. Then, since the resampling fills the records with null values, a strategy called Forward Fill is used. It consists of propagating forward the closest previous valid observation when a null value is encountered. In this way, the resulting sequence is somewhat smooth and consistent with the original.

As for the problem of duplicated data, even if rare and not recurrent, a simple check is performed with the aid of the columns "nename" and "object" since they are unique identifiers, and for every distinct timestep only one measurement with a specific combination of values of those two columns should appear. Therefore, every duplicated record is pruned out.

For corrupted data so far no operation is performed. From an initial analysis emerged that the number of mismatches in the seconds measured is not so high and frequent, so should not impact the overall quality of the data. A possible improvement could come from casting all values that are registered below 5 in the field of *average percentage*, to the minimum possible value allowed by the formula in Section 3.3.1, which is 5. It would not be possible to address the problem of spikes of the same field to higher values, as it would be hard to verify if they consist of error or effective spikes in the network traffic.

3.4. Clustering and initial analysis

Following the preprocessing and preliminary operations on the data presented in the previous sections, some more detailed analysis can be conducted to explore how the data is characterized.
Specifically, a clustering experiment aims at determining whether any unbalances are present in a sample dataset, and if any pattern appears consistently. Clustering operations in the domain of time series are generally used to address the problem of classification, to be able to perform it without having a labeled dataset, in an unsupervised fashion. The clustering aims at identifying and creating groups of time series which are similar in the same group (cluster) and the most different from the ones in other clusters.

The aim of this experiment is to explore the potential viability of a distinctive approach to address time series forecasting. This approach involves the creation of multiple models, each dedicated to a specific cluster. This might have benefits resulting from the high-speciality models that are trained only on specific patterns and the same family of series, but downsides might consist in time and computational requirements that are typical in a clustering procedure (i.e., the series might need to be clustered and/or classified before performing the prediction step).

Performing clustering of time series is not as straightforward as for discrete or real numeric data, and it requires special attention. The metrics that can be used vary based on the nature of the data considered, for example, if dealing with equal or different lengths, categorical or numerical values. In this specific case, after the preprocessing described in Section 3.3, all the time series have equal lengths, and the time-varying variable considered for the clustering is real numerical.

As for the chosen metric, the one that can best suit this scenario is Dynamic Time Warping (DTW). This metric is in fact able to capture patterns occurring in different time series even when shifted in time [43]. As for how DTW works, the two-time series have to be equidistant in the samples, meaning that the sampling has to be consistent. In our case, this is respected, since every time series is resampled with 15-minute intervals even

in cases where data is missing.
As said, DTW captures similar patterns in different phases, finding a minimal path between the two series. It does that through a local cost matrix that is built based on the pairwise distances between every step of the two-time series. Then, the **warping path** is found by following the lowest cost area of the matrix (valleys) [39]

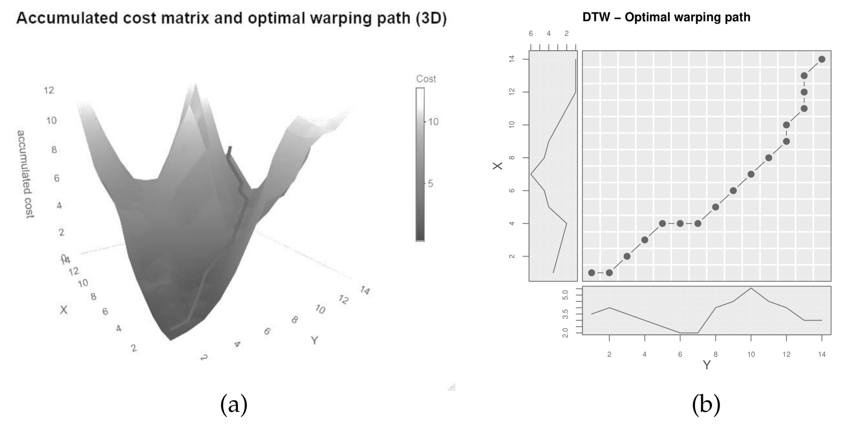


Figure 3.2: Example of pairwise local cost distance matrix and warping path computation [41].

The DTW algorithm is quite expensive from the computational standpoint, having a time complexity of $\mathcal{O}(n^2)$.
From the conducted experiments, the appropriate number of clusters seems to be 7, and the most populated clusters contain samples similar and close to almost flat time series.

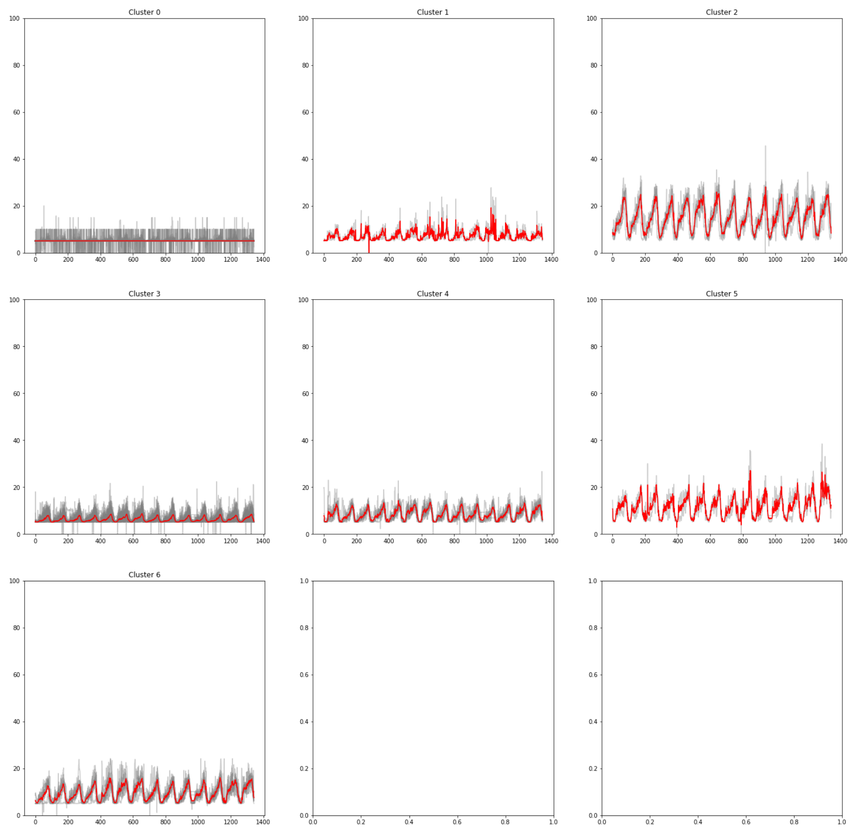


Figure 3.3: Plot of resulting clusters.

Figure 3.3 shows the elements populating the clusters in grey, and then in red the mean of the cluster itself (or centroid), descriptive of the general pattern identified in the cluster. In total, the explorative clustering has been performed on a total of 724 series, sampled randomically from the dataset. Each series is 1344 timesteps long, representing a full week of data. Each series is also normalized with a Min-Max scaler, not on the overall minimum and maximum values over the whole dataset, but following local minimum and maximum values of the series. The cluster populations after the training is as follows:

- **Cluster 0:** 660 elements
- **Cluster 1:** 2 elements
- **Cluster 2:** 5 element
- **Cluster 3:** 38 elements
- **Cluster 4:** 8 elements
- **Cluster 5:** 2 elements
- **Cluster 6:** 9 elements

The first cluster contains almost all of the series, and we can notice that they all share the same pattern of being mostly flat-lines stable at 5% average percentage utilization and occasionally showing some peaks. Some other clusters are slightly more populated than others, but the comparison between the seven groups shows firstly that the dataset is highly unbalanced towards the flat time series, but the clustering shows promising results in terms of aggregating similar series together with a good approximation.

This clustering experiment suggests that with this current dataset, it would be not worthwhile to develop multiple models, each specific for a single cluster, trying to find a balance between locality and globality. This approach might yield good results with the most populated clusters, but these are the same that given the nature of the time series could be easily forecasted using statistical approaches as simple as linear regression models.
On the other hand, for the least populated clusters, it would be more difficult to have consistent data to obtain decent performances.
It is worth noticing that given the consistency across the patterns highlighted by the clustering experiments, facing the same problem of developing a forecasting algorithm with a different set of data and/or a different network, the strategy of developing specialized multiple models could provide good results, and could be worth exploring, as how presented in the paper "Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series" [5].

3.5. Dataset tuning

The result obtained in the clustering experiment clarifies the strong unbalance that characterizes this dataset.
In order to build a sample training dataset, it would not be enough to just randomly sample time series and constitute an example subset, because checking how the capacity utilization measurements are distributed for this network, more than 90% of the registered values are between 5% and 10%. This could be noticed also in the cluster population data presented in Section 3.4.

The interesting aspect is that the data used for the clustering experiment (of which the results are shown in Figure 3.3) was sampled with a semi-random approach aiming at reducing the unbalance, where the original data is divided into three equi-widths bin using the standard deviation of the "avg_perc" variable of the time series as the deciding metric.

The standard deviation might not be the most accurate identifier to use to split time series depending on how they are subject to showing seasonality and/or patterns, but some previous experiments showed that although simple, it worked.
From the three groups created through the equi-width binning, an equal number of time series is sampled, trying to balance the sample dataset. Eventually, the dataset obtained is still unbalanced.

As a backup, following the same cleaning operations shown starting from Section 3.3.1, two additional sample datasets are created. Respectively, all the interfaces from January and February 2023 with a standard deviation greater than 1 are sampled. This ensures that all the time series showing flat horizontal lines (i.e., more than 90% of the whole available data) are not considered in the resulting dataset.

The experiments using these presented datasets and the relative results are described in Section 3.6

3.6. Model Training

The training process focuses on finding a *good enough* model that can be trained with the datasets presented in the previous section. In particular, following the **global** approach to forecasting mentioned in Section 2.2.5, the model is trained using multiple univariate time series to learn patterns and characteristics to be leveraged at inference time.

"Good enough" means that in this step of the project, the aim is not to develop a model which achieves really high scores in the forecasting task, but since the scope of the thesis project is the one of developing the whole infrastructure, from the necessary data pre-processing operations to the deployment of the microservices using Docker, the accuracy of the model can be fine-tuned also at a later step, given that the service is designed considering possible interchangeability of the model as a feature.

All the experiments have been carried out using mainly the tools offered by the library Darts [23]. This library offers ready-made implementations of the models presented in Section 2.2.6, allowing high customization of them and with a clear and simple interface to perform hyperparameter optimization.

The hyperparameter search is done using another library as support, Optuna [3], which allows for easy hyperparameter space configuration and for customization in the searching algorithm.

3.6.1. Experiments on first dataset

The first experiment revolves around fitting models to the dataset consisting of equally-sampled series from equi-widths bins, categorized through standard deviation. This dataset consists of 724 time series of 1344 samples, representing 2 weeks of data.

Even though the samples in this dataset were selected following a strategy aiming at obtaining a balanced dataset, the resulting group still seems to be strongly oriented towards time series showing a pattern of low utilization. Figure 3.4 shows how one of these series appears.

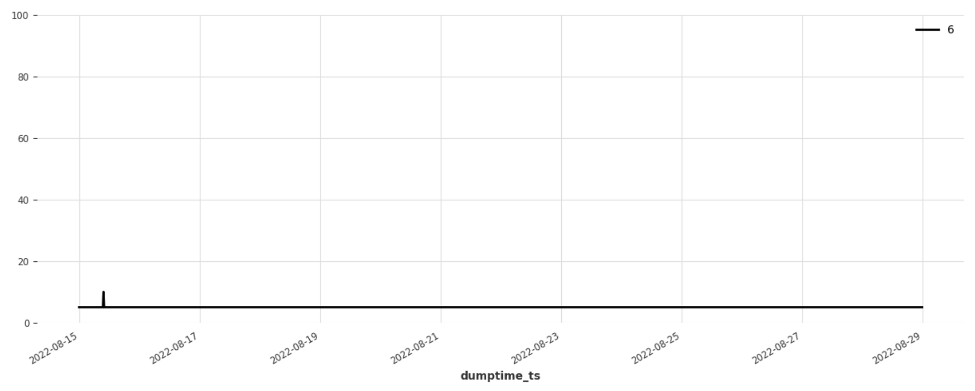


Figure 3.4: Low-traffic time series.

The models trained using this first dataset have an extremely low error (in the case of TCN, the sMAPE on the test set is 1.83 +- 4.24), but the predictions provided are not the most accurate.
The low error is due to the abundant presence of flat time series in the test set, which the model can well predict given that is a horizontal line, but the error increases in noisier and more articulated time series. This can be easily traced back to the vast majority of time series in the dataset being flat, biasing the training of the model and leading to overfitting.
Without providing meaningful data in the training set, the model is not able to learn a good approximation of a noisier time series and works best only with flat patterns.

To this regard, the dataset presented in Section 3.5 consisting of data from all interfaces from January is picked to perform the experiments, with a higher concentration of "meaningful" time series.

3.6.2. Experiments with tuned dataset

Aiming at learning the best possible approximation of the data, using a dataset populated with flat time series as described above is not very helpful. Another dataset is prepared for this case, in the specific, all the data of nodes with a registered standard deviation greater than 1 in the month of January 2023 was collected.

This adds up to a total number of 410 interfaces, each containing 2976 samples (31 days). This ensures that all series selected do not show flat patterns and horizontal lines. Even though this is a simplistic yet good enough discriminant, some series being included in the dataset are still partially flat, corresponding to interfaces that were actively used but then put into maintenance, as shown in Figure 3.5. This is a possible case of something that might happen, so these time series are not excluded.

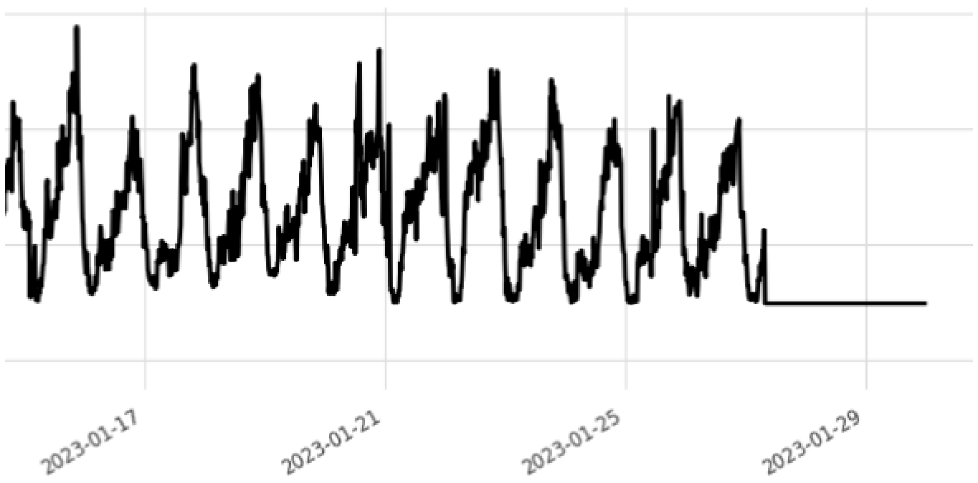


Figure 3.5: Example of variation in the time series behaviour in case of maintenance.

The same three models presented in Section 2.2.6 are trained and tested on this dataset, being Temporal Convolutional Network (TCN), N-Beats and N-HiTS. The dataset is split into training/validation/test, with the latter two having a length of 96 samples, corresponding to one day. This decision is made according to the planned output of the model at a business level, which is expected to be one day.

The models are trained using Darts, and Optuna for hyperparameter optimization, performing validation over the best hyperparameter configuration using the validation set. Once the hyperparameter tuning is completed, the best models are trained and saved, then compared using the test set to assess the best overall performance.

Every time series is scaled before training using a MinMax scaler. This operation is considered to help the training of a deep learning model, having lighter computational steps in the adjustment of the weights.

It is important to mention how the models' implementation of Darts handles the training over a customized dataset like ours. Starting from a simple example of training the model on a single time series, what happens is that the model automatically creates a set of sub-samples from the original time series. All the sub-samples share the same input and output length as mentioned for parameters in the model declaration, in order to fully exploit all the data contained in the series. With this "rolling-window" approach the training is not performed only on the last sequence of the time series, but the model gets to learn patterns present in all the time window represented by it.

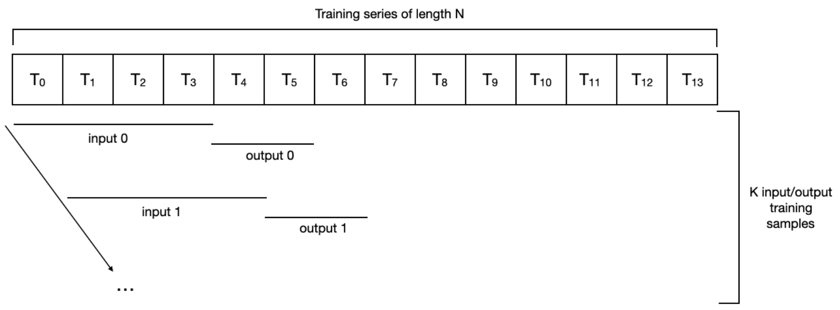

Training series of length N
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10} T_{11} T_{12} T_{13}
input 0
output 0
input 1
output 1
...
K input/output training samples

Figure 3.6: Sub-samples creation from the original time series, in the case of a single time series [23].

Figure 3.6 shows the process in the case of a single time series, but the process is the same in the case of multiple time series, as shown in Figure 3.7. The number of sub-sample created in this way is equal to:

$$tot\_samples = number\_of\_series \times (length\_original\_series - input\_length - output\_length + 1)$$

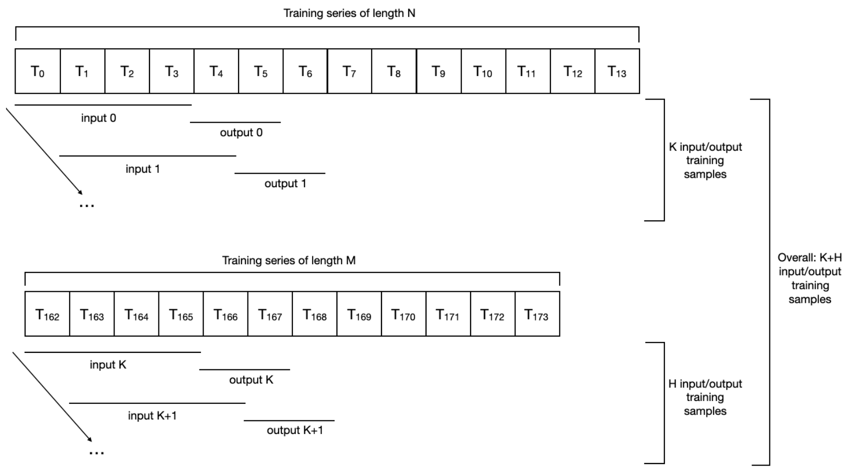

Training series of length N
T_0 T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8 T_9 T_{10} T_{11} T_{12} T_{13}
input 0
output 0
input 1
output 1
...
K input/output training samples
Training series of length M
T_{162} T_{163} T_{164} T_{165} T_{166} T_{167} T_{168} T_{169} T_{170} T_{171} T_{172} T_{173}
input K
output K
input K+1
output K+1
...
H input/output training samples
Overall: K+H input/output training samples

Figure 3.7: Sub-samples creation from the original time series, in the case of multiple time series [23].

Additionally, every model is trained using Early Stopping, monitoring the validation error on the validation set. This technique is used to avoid overfitting of the models and achieve an optimal performance during training.

The forecasting task addressed with this project is not considerably easy due to the amount of noise that the time series contains. Expecting to forecast future values with a deterministic approach and a very high accuracy would be very complicated and hard to obtain.
Rather than predicting the exact value, the mentioned models have the capability of being trained in learning the parameters of a probabilistic distribution modelling the time series and then being able to draw samples from that distribution, enabling a statistical view of the forecasting process and enabling confidence threshold estimation and visualization.

This approach takes into consideration the uncertainty of the model, and in a use case such as the one addressed for resource scheduling, it might bring higher flexibility to the resource allocation process.

Another way of taking into consideration the uncertainty of the model is achieved using **Monte Carlo Dropout** at inference time.

Both these two strategies are described in the following paragraphs.

Probabilistic Forecasting

In *deterministic forecasting* the model is trained to provide a single output value for every timestep considered for the prediction length, and the learning process aims at optimizing this value, generally resulting in finding an average output that minimizes the error. These models are interpretable in the output and they also account for uncertainty, but in a limited way.
With *probabilistic forecasting* the model is not trained in learning to output a set of finite values, but rather in learning a probability distribution approximating the time series used as a training set, and then samples an arbitrary number of possible output sequence to consider for the prediction. Hence, the result is not a single value for each timestep of the prediction length, but a number of possible values coming from different samples fetched from the output probability distribution, for every timestep of the prediction length.
In this way, statistics and more accurate analysis can be conducted on the output (e.g., taking the average and considering different quantiles) to drive informed business decisions.
With this approach, the uncertainty becomes explainable and interpretable, and the risk factor can be considered while making decisions.

In this specific case, the uncertainty that is considered adopting this methodology can be called *aleatoric uncertainty*, and in Deep Learning specifically refers to the randomness that is intrinsically contained in the problem statement. This randomness can be built up from the noise and randomness of the data, the model layers itself, and a number of other reasons. The aleatoric uncertainty is identified as such because more knowledge (i.e., more training data) doesn't reduce it. The idea of modelling all the data considered with an accuracy of the 100% is unrealistic, and the best possible option is providing an approximation. With a probabilistic model, this approximation also accounts for the aleatoric uncertainty, learning to output the parameters for a specific Probability Distribution. The model is trained by learning only the parameters of this distribution (arbitrary choice for this distribution, depending on the problem and the data domain) and learns to minimize the negative log-likelihood of the training samples.

Figure 3.8 highlights how the purple line (label "forecast") accounts for the average forecast output, whereas the pink shadows contouring the time series prediction is a visual measure of the uncertainty of the model over the prediction: the larger the width of the shadow, the higher the uncertainty. To explain how the shadow is considered, we first need to describe how the output is defined.

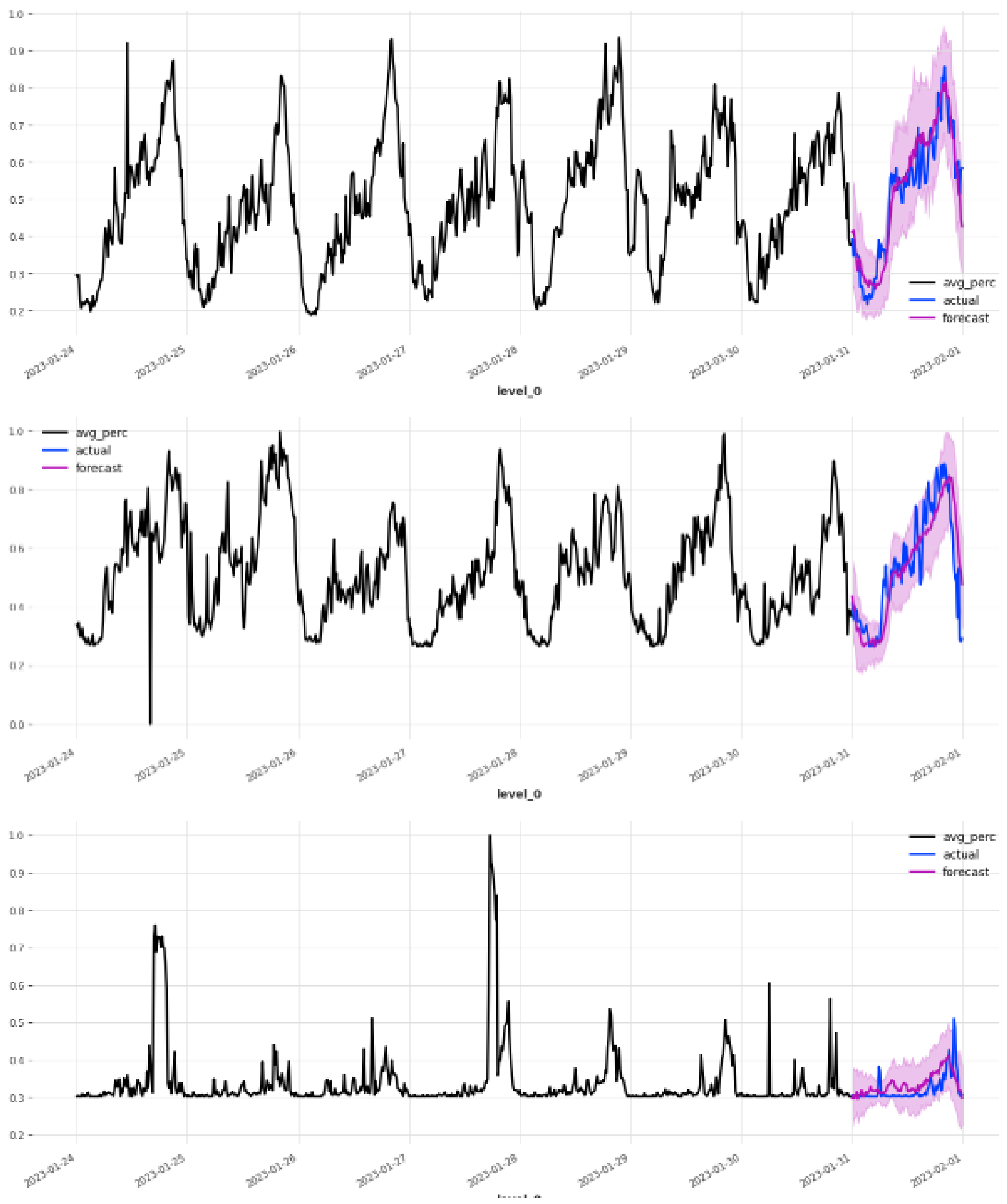


Figure 3.8: Example plot of probabilistic prediction, on three different time series.

For a deterministic model, the output results in the following:

$$\chi_{t+1,T} = \{x_{t+1}, x_{t+2}, x_{t+3}, .., x_T\}.$$

The output of the model is a single set of values, of dimension T equal to the prediction length, starting at time step index $t+1$, where time step t denotes the end of the input window.

In the case of a probabilistic model, the output is defined as:

$$\chi_{t+1,T} = \{X_{t+1}, X_{t+2}, X_{t+3}, .., X_T\}.$$

Where for every time step of the prediction length spanning indexes $\{t+1, T\}$ the output consists of a set of values of arbitrary length X_n. All the sets of values have the same length and are sampled from the probability distribution that is learned by the model (e.g., Gaussian Distribution).
Following this definition, the purple line that averages the predicted output is the average of all the timestep value-sets, and the pink shadow is drawn respectively between the 0.05 and 0.95 quantiles computed over the set of values
$\{X_{t+1}, X_{t+2}, X_{t+3}, ..., X_T\}$.
A **quantile** is defined as a cut-point where a data sample is divided into equally sized, adjacent sub-groups.
In this case, the area contained in the pink shadow accounts for 90% of the observed samples in the predicted output, and only 10% of the samples are left out (respectively 5% below the bottom line, and 5% above the top line).

Monte Carlo Dropout

The strategy of adopting Monte Carlo Dropout (MC-dropout) is possible only in Neural Networks containing dropout layers. Using this approach allows one to interpret and gather information about the model's confidence and uncertainty in the prediction, allowing the same interpretation that is possible to obtain using Bayesian's models, but without having the high-overhead coming from intensive operations [20].

This approach aims at reducing the *epistemic uncertainty*. Epistemic uncertainty (from *episteme*, Greek for knowledge), refers to the uncertainty incorporated in a model due to its parameters configuration, which could eventually be addressed by having more data points as input for the training set. In simpler terms, it could be reduced by having more *knowledge* incorporated in the network during training.

To understand MC dropout, it is best to start with the definition of the dropout layer. This particular layer is effectively used only at training time, when from one training iteration to the other certain neurons are turned off in the effort of regularizing the network training and avoiding overfitting.
Specifically, turning down randomly neurons in layers stimulates the network to find and

exploit new and under-utilized connections, avoiding the incremental exploitation of the same links between neurons, that would result in overfitting. By training the model with this approach, at inference time the output could be interpreted as the average of the predictions of an ensemble of different neural networks. The dropout at training time can be set based on a parameter, p, stating which is the probability for every neuron in a layer to be turned off.

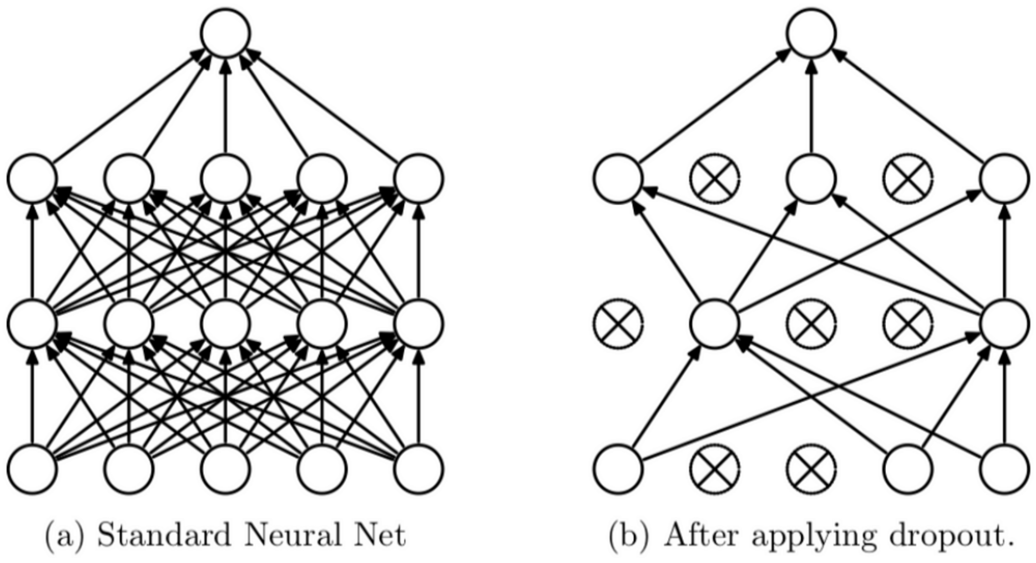


Figure 3.9: Visual explanation of how dropout layers work.

With MC-dropout, the dropout probability is considered also at inference time. The prediction is no longer deterministic, and the output consists of a different set of values, each coming from a different, random configuration of the network after considering the dropout iteratively, all of them taking the same input. What MC-dropout achieves would be comparable to sampling predictions from probabilistic distribution, referenced to as Bayesian interpretation.
The MC-dropout can be considered only in the case of probabilistic forecasting, since for every sample drawn from the output distribution, a different configuration of the network is used.

4 | Numerical experiments results

This section presents the results obtained in the different experiments explained in Section 3.6.
Firstly the chosen metric is explained and its choice motivated, then the hyperparameters configuration that resulted from the hyperparameter tuning are presented. Lastly, the scores obtained on training, validation and test set by the models.

4.1. Evaluation metric

For the task of time series forecasting the choice of an evaluation metric is not trivial and depends on the specific use case and on how the model's performances need to be interpreted. Notably, the group of evaluation metrics can be coarsely divided into two families:

- **Absolute error metric:** defined as the absolute difference between the model's predicted value and the actual value of the test set. Depending on the scale and size of the problem, a minor difference could be negligible, where a much bigger distance in absolute value would result in a high error
- **Relative error metric:** defines how much the predicted value is distant from the actual value, *relatively* to the size of the actual value.

In other words, the *absolute error* is useful in situations where the error interpretation needs to define how good the prediction is compared to the exact value, keeping the size factor in consideration. With the *relative error* it is more easily understandable how "good or bad" the prediction is, without incorporating the scale factor.

Another important aspect regards the difference between **scale-dependent** and **scale-independent** metrics. The former has the benefits of being easily interpretable and computable, but at the same time, they strongly rely on the nature of the data used in the problem, since are built around the scale of the data.
Percentage error metrics are scale-independent metrics that can be used across different

time series.
Specifically for this project the percentage error metric chosen is **sMAPE** (Symmetric Mean Absolute Percentage Error).

To present sMAPE it is easier to start from the basic formulation of this family of metrics, which is **MAPE** (Mean Absolute Percentage Error). The formula for MAPE is:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i} \times 100|$$

This metric is generally good for a number of use cases, and the go-to choice for percentage error metrics. Although, suffers from a problem regarding asymmetry.
Specifically, it benefits models that provide an under-forecast w.r.t. models that tend to over-forecast (because of the denominator term). To avoid this behaviour sMAPE can be used, where the relative error is normalised also for the forecasted value:

$$SMAPE = \frac{100}{n}\sum_{i=1}^{n} |\frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|}$$

Naturally, the lower the sMAPE the more accurate the prediction.

4.2. Hyperparameter tuning results

In this section, the results of the process of hyperparameters tuning are presented, with the optimal configuration found for every model's hyperparameters. As a recall, the optimization is done with the library Optuna [3].

Optuna bases its workflow mainly on two concepts, study and trials. A study is an object that aims at obtaining optimization of an objective function. The trial is a single execution of the objective function.
At each trial, the choice of the parameters varies spanning possible values contained in a search space defined by the user.
Specifically, the sampling of new suggested values across trials is performed through a TPE Sampler (Tree-structured Parzen Estimator Approach). This sampler follows an independent-sampling strategy, where each parameter's value is estimated independently of any other relation with the others.
During each trial, TPE fits two separate Gaussian Mixture Models (GMM): one to the set of parameter values corresponding to the best objective values and another to the set of parameter values associated with the remaining trials. Then, it selects the parameter value that maximizes the ratio between the likelihoods of the two fitted GMMs.
The process of optimization is further optimized using a pruner. The Median Pruner is used to abort a trial before its end when the intermediate result is worse than the median of the previously seen results at the same step.
Optuna computes the score for each trial using the validation set, to establish a score to select the best configuration.

The following sections show the results of the optimization process, with the relative best parameters for the models.

4.2.1. N-HiTS

Table 4.1 shows which parameters were the target of the study for N-HiTS and their relatively optimal values.

Table 4.1: Parameters for optimization of N-HiTS and optimal values found

	Optimal values
days_in	6
number_of_stacks	3
number_of_blocks	5
layer_widths	512
number_of_layers	4
dropout	0.14653058
learning_rate	0.000647999
likelihood	Gaussian
include_day_of_week	False

Figure 4.1 highlights how each parameter weights on the objective value of the study.

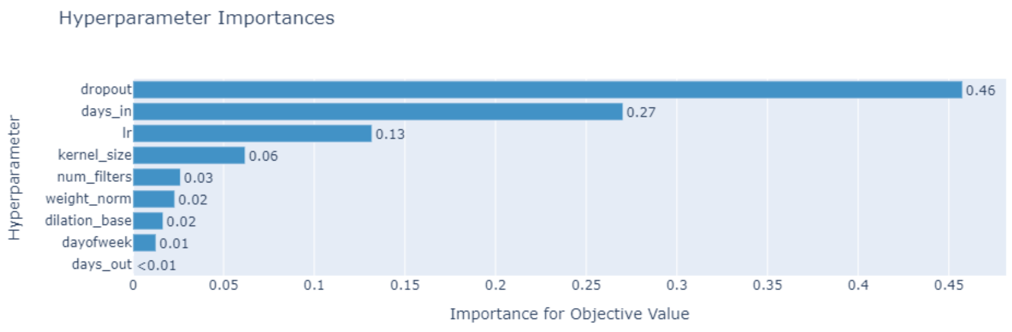


Figure 4.1: Relative parameters weights on the objective optimization.

4.2.2. Temporal Convolutional Network - TCN

Table 4.2 shows which parameters were the target of the study for TCN and their relatively optimal values.

Table 4.2: Parameters for optimization of TCN and optimal values found

	Optimal values
days_in	8
kernel_size	25
number_of_filters	23
weight_normalization	True
dilation_base	4
dropout	0.176825329
learning_rate	0.00047756767
likelihood	Gaussian
include_day_of_week	False

Figure 4.2, states how each parameter weights on the objective value of the study.

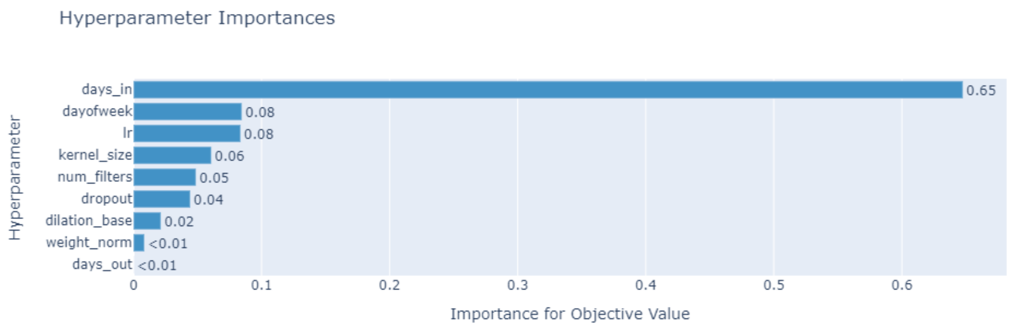


Figure 4.2: Relative parameters weights on the objective optimization.

4.2.3. N-Beats

Table 4.3 shows which parameters where the target of the study for N-Beats and their relatively optimal values.

Table 4.3: Parameters for optimization of N-Beats and optimal values found

	Optimal values
days_in	7
generic_architecture	True
number_of_stacks	4
number_of_blocks	7
layer_widths	512
dropout	0.0377747209
learning_rate	0.0009835964
likelihood	Gaussian
include_day_of_week	True

Figure 4.3 states how each parameter weights on the objective value of the study.

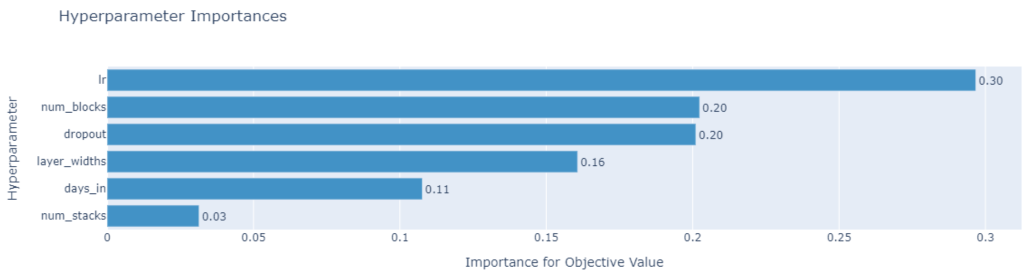


Figure 4.3: Relative parameters weights on the objective optimization.

4.3. Evaluation results

As presented in Section 4.1, the models are evaluated using sMAPE as a metric, and as the test set the last day of observations from the month of January 2023.
As a reminder, the training set used in the training process for the models consists of all the possible sub-samples created from the original time series, and after every five training iterations, an additional check using the validation set is performed in order to eventually allow Early Stopping. The early stopping technique aims at preventing model overfitting

and is achieved by monitoring the error on the validation set, and whenever the error on the validation set starts to increase while the error on the training set keeps decreasing, after a "patience" period of some iterations to assess that the trend observed is persisting, the training is stopped and the model is rolled back to its weight configuration before the divergence of the two errors.

In Table 4.4 are presented the errors respectively on train, validation and test set for the three models studied in the project, and for a deterministic baseline:

Table 4.4: Error values using sMAPE

	Training error	Validation error	Test set error
Linear Regression	13.24 ± 3.58	12.83 ± 6.84	11.89 ± 6.05
TCN	14.83 ± 3.69	16.68 ± 5.78	12.73 ± 4.47
N-HiTS	16.02 ± 3.73	17.97 ± 6.59	13.85 ± 5.69
N-Beats	143.84 ± 7.88	130.90 ± 12.24	131.36 ± 12.46

Every training set error score is computed using a technique referred to as backtesting. Backtest aims at discovering how the model would have performed on the training data, once the training has finished. The model iteratively builds input-output subsets following the approach described in Section 3.6.2, and the final score is an average of all the error scores of the predictions provided for every subset time series.

The first model presented in the table is a simple linear regression predictor fitted on the same dataset as the other models, used as a baseline to compare the performances. Notably, it performs slightly better than the more complex deep learning models. Even though the sMAPE score is lower for this model, it is mentioned that the predictions provided by the linear regressor fail to encapsulate some of the peaks and the noisy nature of the data, giving a good approximation of the mean of the series, but at the same time underestimating what would be the traffic in the prediction interval.
For a visual example refer to Figure 4.4.

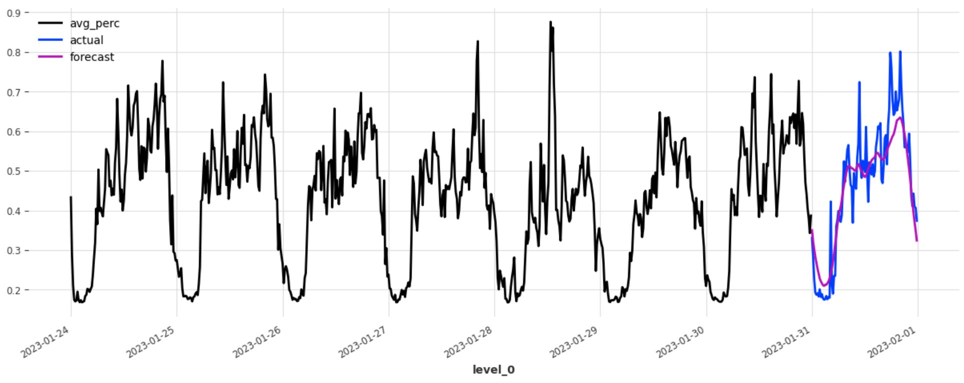


Figure 4.4: Prediction plot obtained with linear regression.

Clearly, the model gives a good approximation of the pattern of the time series capturing its average (that is the reason why the sMAPE score is relatively good), but at the same time, all the peaks are considered as noise and ignored.
Moreover, the linear regression does not provide a probabilistic output, decreasing the possible flexibility in terms of utilization and statistical analysis of the predictions.
One last aspect to consider is that the models are trained with a completely different approach, because in the training process, the probabilistic models aim at optimizing the negative log-likelihood using the probabilistic distribution they are learning, whereas a linear regression uses a loss function a common error metric, the Mean Average Error (MAE).
This difference makes it hard to compare the probabilistic models using a deterministic baseline, but the close score obtained in both cases using sMAPE suggests that also the probabilistic models have learned to approximate the time series.

As an additional benchmark for the models can be considered also the elapsed time in the inference phase, which is presented in Table 4.5. These measurements refer to the time taken to provide the prediction for the 410 time series that are contained in the test set.

Table 4.5: Elapsed time in inference phase, in seconds

	Elapsed time
Linear Regression	18.3s
N-HiTS	14.8s
TCN	56.7s

The scores for the probabilistic models shown in Table 4.4 are computed using the average of the prediction output. Every model at inference time provides 150 samples for every prediction. At first look, the performances of N-Beats are not even closely competitive with the results obtained by the other two models. TCN appears to be the best choice considering only overall performances, but considering also the training time and inference time of the models, N-HiTS can be considered a valid candidate for the selection, having a really fast prediction phase and a generally lighter training process than TCN.

One counterintuitive result that can be observed is that the performances registered on the test set are slightly better than the ones obtained on the validation set. This can be due to a lot of reasons, but given that all the models trained show this trend, it can be due to a lower noise present in the test set w.r.t. the one present in the validation set.

Overall the scores suggest that the models did not overfit on the training set and that the techniques employed to reduce the incorporation of bias in the model training were successful.
Remembering that one of the objectives stated in the research questions for this part of the project was to develop a model able to predict the traffic over the next day timescope, preferably using deep learning with a probabilistic forecasting approach, the results are considered optimal.

These results furthermore prove that new deep learning architectures are gradually and effectively reaching the same level of performance as other previously largely acquired statistical models.

After these experiments and seeing the results, N-HiTS is chosen as the model to progress in the implementation of the microservices, presented in the next sections.

5 | Microservice Implementation

This section provides a brief overview of the main implementation features of the microservices that are presented in the service architecture shown in Section 1.1.
Two microservices are implemented for this project, respectively the Traffic Prediction service and the Deep Sleep scheduler. The data processing microservice is left for future development of the project, having already extensively described all the necessary operations for the data pipeline in Section 3.3.

5.1. Traffic Prediction Microservice

The main purpose of this microservice is ultimately to produce a forecast for the expected capacity utilization for every node in a specific network over the next day, using historical data.
For the first version of this microservice the database connection is not yet implemented, and the data is loaded statically from files. Ideally, this microservice could incorporate a model trained to provide the schedule for the radio shutdowns, but at a business level, it was decided to have a service providing simply the predictions. That would open in the future possible scenarios where the predictions can be used on different use cases as the one of the Deep Sleep scheduler.
A breakdown of the algorithm implemented by the microservice follows.

Algorithm 5.1 Traffic prediction workflow

```
1:  Load model;
2:  Load data;
3:  Fetch start date;
4:  while $ENABLE\_FORECAST == True$ do
5:      dataset = prepare_dataset(data)
6:      prediction = model.predict(dataset)
7:      save(prediction)
8:      start_date.increment(hours=4)
9:      sleep(hours=4)
10: end while
```

The algorithm is not complex to understand, but some details need to be clarified:

- **Load model:** in a configuration file the user can specify which model to load and use to predict the data. The first version of the microservice supports models built using two libraries, Pytorch Forecasting (based on Pytorch Lightning) and Darts. The model can be loaded and through just some changes in the configuration file, they are ready to be used.
- **Fetch start date:** as mentioned, the first prototype of the microservice is not connected to the database. For how it is expected to work, the data should be retrieved from the database from the current moment to some days in the past (according to the input length of the model). Since this is not achievable, to test this feature a file containing a whole month's data for certain nodes is used as a test set, and the data is prepared the same way as it is described in Section 3.3. The dataset is loaded iteratively, following a rolling window approach, updating the start date of the input window according to the set prediction frequency.
- **Prepare specific dataset:** models built with different libraries need different input objects. With a check on which model is being used for the prediction, the specific data loader object is prepared.
- **Save predictions:** the prediction output is saved to an output file. The predictions are continuously overwritten since there is a mismatch in the prediction frequency and prediction output. Specifically, in this first version, a prediction is provided every 4 hours and the output length corresponds to one day. So, for every new prediction for every node, 20 hours of the already existing file are overwritten and 4 new hours are added, corresponding to the new data registered into the system. In

this way, the prediction is updated with newer and supposedly more reliable results.

In the prediction file are registered values corresponding to three quantiles, that can be decided by modifying the configuration file. With this approach the final user can decide on a greedier or more relaxed approach to the shutdown schedule, deciding to be more conservative concerning the predicted values or opting for a safer approach.

5.2. Deep Sleep Scheduler

This microservice is a subsidiary of the traffic prediction microservice just presented in Section 5.1. Specifically, whenever a new prediction is provided, the forecast file is fetched and analyzed to decide which is the best time interval in which to shut down every radio, according to the hardware specifications and other rules.

An overview of the operations workflow:

Algorithm 5.2 Deep Sleep Scheduler workflow

```
1: while $SCHEDULE == True$ do
2:     topology = read_topology()
3:     predictions = read_predictions()
4:     recent_predictions = split_recent(predictions)
5:     schedule = flag_shutdown_sequences(recent_predictions)
6:     save(schedule)
7: end while
```

Analyzing deeper each instruction:

- **Load topology and prediction data:** the predictions provided by the Traffic Prediction microservice are loaded, as well as the topology data describing properties of nodes and links across the network.
  The topology dataset is a table describing relationships between all the hierarchical links that are considered in a network for microwave components. The identifiers on which the predictions are provided are at an ethernet-link level, but multiple sub-layers are present between the ethernet link and the carriers link, so the chain of relationship between those has to be reconstructed.
- **Split prediction data:** the predictions file is a concatenation of all the predictions for every node, continuously overwritten by the other microservice. The prediction

data is split into groups based on the node identifiers, and only the newly added predictions are fetched (the last 96 samples in chronological order).

- **Flag shutdown sequences:** the node identifier is cross-checked with the topology to identify how many carriers are active on that node, and if the shutdown can be put in place. If the number of carriers is not 1, then the two longest consecutive sequences of predicted values under a certain threshold (50% average utilization) are flagged with the field "OFF" on an additional column. The same process is repeated for every quantile prediction column that is included in the prediction file.
- **Save schedule:** the output of the scheduling is saved as an independent file each time, without following the overwriting approach of the traffic predictor. This is because the shutdown scheduling is not a completely automated process for how this service is intended. The schedule has to offer a view to the operator to understand which strategy to take. Also, it would not be convenient to update the schedule multiple times a day, both from a hardware perspective (repeatedly turning off and on components shortens their life expectancy) and from a business perspective, as it would be hard to understand which strategy would be best to follow after every new prediction is provided.

6 | Future developments

As mentioned in Chapter 5, this project is intended as a first prototype for a more comprehensive and functional product for when the whole platform is ready to host services such as this one.
Nevertheless, some features were already part of the initial plan for the project, but could not fit in the first version. This chapter aims to provide an overview of some of these features that are considered the most relevant.

Wider training dataset

The first and foremost improvement that could interest the project regards the possibility of constructing a proper training set with data representative enough of a network. It has been identified throughout the work, specifically in Section 3.3 and Section 3.6, that one of the main aspects characterizing the network's data used in this project was to be highly unbalanced, with a lot of the microwave radios in the network being under-utilized. Training a model with a dataset with this distribution can result in high bias and poor performances, especially when inferring those nodes that are not under-utilized. Having data coming from a more consistently used network could help the model learn more precise patterns.

Additional features for traffic prediction

For the first implementation, the only feature used for the model training was the historical average percentage utilization, computed as shown in Section 3.3.1. It is not to be excluded that adding additional information to the data points and using covariates or other variables might affect the performance of the model, creating more resilient and accurate forecasts.

Model ensemble

Following the results and discussion seen in Section 3.4, a possible upgrade would be to explore the strategy of mediating between local and global approaches.
The mentioned strategy proposes to cluster the dataset and identify what are the most common groups of time series, characterized by patterns and seasonality, and train a single model on every group if the data availability allows so. Although, this strategy presents many drawbacks, as listed in Section 3.4. Notably, the most important ones would be the additional effort required to develop the models and the data splitting operations to create the datasets, and also the necessity to cluster and assign input time series to specific clusters when the service will be deployed to production, consistently augmenting the operational overhead, being the clustering operations highly requiring in terms of time and computations.

Another possible approach would be similar to the cluster-dedicated models but with a coarser view. As the data exploration highlights, the number of nodes that result under-utilized is fairly high, so an immediate upgrade would be to develop a classification mechanism (could be a Machine Learning classifier for time series) to decide whether to forecast the incoming time series with a simple linear regressor in the case of an under-utilized interface, or with a trained model.
This strategy would add some overhead to the microservices deployment but could improve performances in terms of accuracy.

Deep Sleep scheduler upgrades

The deep sleep scheduler microservice as described in Section 5.2 is responsible for the creation of the shutdown schedule following a simple rule, which is to follow the suggested number of daily shutdowns that a microwave radio could go through, without affecting the life expectancy of the hardware. Practically, the two longest sequences of values that could be candidates for a period of shutdown are identified.
This is not the only rule and option available for this task. The most urgent implementation need would be to check in a cascade fashion which links are affected by the shutdown, and check if this could be propagated to the links directly chained to the ones of interest, so as to ensure that the shutdown can happen safely and that all the link chain is treated correctly.
Furthermore, the only variable used in the current scheduling is the predicted average per-

centage utilization. Using other exogenous variables (e.g., the expected people affluence in a certain geographical area, event schedules, weather forecasts etc.) paired with more information coming from the network topology (e.g., the geographical position) could provide a more informed view of the network and what can be expected for the next day. This approach would open the possibility of substituting the deterministic approach that has been used now with a Machine Learning method, where another model could be trained on predicting which is the best shutdown schedule based on all of these variables.

Microservices communication and data preprocessing service

The microservices, as presented in Chapter 5, are a first draft of what the whole service would look like, and could be intended as the backbone of the Deep Sleep scheduler service. From a software engineering standpoint, the first upgrade they would need would be a database connection, allowing them to read and write prediction data and the shutdown schedule to persistent storage, enabling real-time data injection and analysis.
Another implementation aspect to consider would be to put all the data preprocessing operations described in Section 3.3 in the form of a microservice since the raw data available on the database would not be directly usable for the forecasting task.

Continual learning and model retraining

Lastly, there could be various strategies to address situations in which the model's performances degrade over time, most likely caused by a data drift. In these cases, Machine Learning models need to undergo a process of structural retraining or fine-tuning, to be brought back to the ideal performances.

A data drift (also known as covariate shift) is identified when the observed data changes its distribution over time. In this case, it could be found in network patterns changing, and the model not being able to correctly approximate the newly seen distributions.

Rather than systematically retraining the network (or fine-tuning it) over the most recent data on a recurring schedule (e.g., every 3 months), the retraining could be triggered on-demand, when the monitored accuracy of the predictions would drop under a certain threshold.

This strategy best fits the use case of the nodes' traffic prediction, because it would be fairly rare to experience a structural data drift in terms of variations of traffic patterns in a network.

This approach could save a lot of operational costs and effort since model retraining and deployments are often complex and not trivial.

A possible solution in case of the necessity of performing retraining could consist of fetching the most recent months' data from the database of collected PM data and using them to fine-tune the model, trying to incorporate the new information over traffic patterns without losing the knowledge incorporated in the Neural Network with the previous training.

7 | Conclusions

The results and findings presented in this thesis offer significant advancement towards addressing the problem of traffic prediction and energy optimisation across a network, identified in Section 1.1. The available collected data has demonstrated its efficacy in real-world use cases, after a thorough analysis and the preprocessing operations described in Section 3.3.

Specifically, the data proved to be useful in the end, but the conducted exploratory clustering experiments identified that the PM counters collected from the network are not comprehensively capturing meaningful behaviours. Notably, a substantial portion of the analyzed dataset resulted in generally poor training results for Deep Learning models and had to at last be ignored leaving room for more meaningful and carefully picked time series.
This analysis suggests that it might be possible to serialize the production of models to be trained on different networks, but still might require manual intervention in the data study to identify the best approach to follow.

For this reason, the research was oriented towards developing a Global Forecasting model to suffice the scarcity of meaningful data.
Of the three tried models the chosen one was N-Hits, achieving an accuracy over the test set of 13.85 sMAPE and was chosen because relatively light and fast in inference operations.

Moreover, the choice of a probabilistic model has allowed to include considerations over the uncertainty of the prediction in the architecture, making the whole service flexible and adaptable to the needs of the operators.

Overall, this project establishes a framework for a more comprehensive customer-oriented service, aimed at achieving energy optimization through Artificial Intelligence integrated

into the solution.
The developed architecture offers a first customizable structure following the principle of "separation of concerns" identifying three distinct elements: a data preprocessing service, a traffic prediction service, and a deep sleep scheduler.
This choice facilitates future individual and targeted upgrades to either of the blocks.

As highlighted in Section 6 the possible upgrades are already well-defined, requiring only the expansion of the existing framework and operations.

Overall the outcomes of this project are deemed successful, having created a functional system capable of forecasting microwave nodes' average utilization within a network, to enable adaptive scheduling to save energy, and resources and optimize costs.

Serving as a prototype, this service establishes a baseline for introducing AI-driven services within the context of an SDN architecture. By offering a new approach to network management, this solution introduces fresh value propositions and opportunities for adopters embracing these services.

Bibliography

[1] CommScope Definitions: What is Fixed Wireless Access? URL https://www.commscope.com/blog/2017/commscope-definitions-what-is-fixed-wireless-access/.

[2] Open Transport SDN Architecture Whitepaper. Technical report, 2020. URL https://cdn.brandfolder.io/D8DI15S7/as/q7rnyo-fv487k-efbbqr/IPR_Policy_-.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, New York, NY, USA, 7 2019. ACM. ISBN 9781450362016. doi: 10.1145/3292500.3330701.

[4] S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. 3 2018.

[5] K. Bandara, C. Bergmeir, and S. Smyl. Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach. 10 2017. URL http://arxiv.org/abs/1710.03222.

[6] S. R. Beeram and S. Kuchibhotla. Time Series Analysis on Univariate and Multivariate Variables: A Comprehensive Survey. pages 119–126. 2021. doi: 10.1007/978-981-15-5397-4{_}13.

[7] C. Brooks. Univariate time series modelling and forecasting. In *Introductory Econometrics for Finance*, pages 206–264. Cambridge University Press, 5 2008. doi: 10.1017/CBO9780511841644.006. URL https://www.cambridge.org/core/product/identifier/CBO9780511841644A068/type/book_part.

[8] T. Cerny, M. J. Donahoo, and M. Trnka. Contextual understanding of microservice architecture. *ACM SIGAPP Applied Computing Review*, 17(4):29–45, 1 2018. ISSN 1559-6915. doi: 10.1145/3183628.3183631.

[9] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and

A. Dubrawski. N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting. 1 2022.

[10] C. Chatfield. Time-Series Forecasting. *Significance*, 2(3):131–133, 9 2005. ISSN 1740-9705. doi: 10.1111/j.1740-9713.2005.00117.x.

[11] Y. Chen, Y. Kang, Y. Chen, and Z. Wang. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 399:491–501, 7 2020. ISSN 09252312. doi: 10.1016/j.neucom.2020.03.011.

[12] Chris Chatfield. *Time-series Forecasting.* 1st edition, 2000.

[13] J. G. De Gooijer and R. J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 1 2006. ISSN 01692070. doi: 10.1016/j.ijforecast.2006.01.001.

[14] P. Di Francesco, P. Lago, and I. Malavolta. Migrating Towards Microservice Architectures: An Industrial Survey. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 29–2909. IEEE, 4 2018. ISBN 978-1-5386-6398-1. doi: 10.1109/ICSA.2018.00012.

[15] S. Du, T. Li, Y. Yang, and S. J. Horng. Multivariate time series forecasting via attention-based encoder–decoder framework. *Neurocomputing*, 388:269–279, 5 2020. ISSN 18728286. doi: 10.1016/j.neucom.2019.12.118.

[16] K. Edeline and B. Donnet. A Bottom-Up Investigation of the Transport-Layer Ossification. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 169–176, 2019. doi: 10.23919/TMA.2019.8784690.

[17] Ericsson. 5G carrier aggregation for better deployment, . URL `https://www.ericsson.com/en/ran/carrier-aggregation#:~:text=With%20Ericsson%205G%20Carrier%20Aggregation,capacity%2C%20and%20higher%20data%20speeds`.

[18] Ericsson. Boost capacity in traditional bands., . URL `https://www.ericsson.com/en/mobile-transport/boost-capacity-in-traditional-bands`.

[19] A. Frithiofson and M. Vacante. Energy Efficient Control of Microwave Networks. Technical report. URL `www.chalmers.se`.

[20] Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. 6 2015.

[21] E. S. Gardner. Exponential smoothing: The state of the art—Part II. *International*

Journal of Forecasting, 22(4):637–666, 10 2006. ISSN 01692070. doi: 10.1016/j.ijforecast.2006.03.005.

[22] Gsa. Public Networks and Operators: GAMBoD Database Snapshot, March 2023. Technical report, 2023.

[23] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. Van Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, J. Koscisz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch. Darts: User-Friendly Modern Machine Learning for Time Series. Technical report, 2022. URL `https://github.com/unit8co/darts`.

[24] P. Hewage, A. Behera, M. Trovati, E. Pereira, M. Ghahremani, F. Palmieri, and Y. Liu. Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station. *Soft Computing*, 24 (21):16453–16482, 11 2020. ISSN 1432-7643. doi: 10.1007/s00500-020-04954-0.

[25] R. Hyndman and G. Athanasopoulos. *Forecasting: principles and practices.* 3rd edition, 2018. URL `https://otexts.com/fpp2/nnetar.html`.

[26] B. Lim and S. Zohren. Time-series forecasting with deep learning: A survey, 4 2021. ISSN 1364503X.

[27] T. Lin, J. M. Kang, H. Bannazadeh, and A. Leon-Garcia. Enabling SDN applications on software-defined infrastructure. In *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World.* IEEE Computer Society, 2014. ISBN 9781479909131. doi: 10.1109/NOMS.2014.6838226.

[28] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li. Microservices: architecture, container, and challenges. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 629–635. IEEE, 12 2020. ISBN 978-1-7281-8915-4. doi: 10.1109/QRS-C51114.2020.00107.

[29] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 Competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4): 802–808, 10 2018. ISSN 01692070. doi: 10.1016/j.ijforecast.2018.06.001.

[30] P. Montero-Manso and R. J. Hyndman. Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting*, 37(4):1632–1653, 10 2021. ISSN 01692070. doi: 10.1016/j.ijforecast.2021.03.004.

[31] Nadareishvili Irakli, Mitra Ronnie, McLarty Matt, and Amundsen Mike. *Mi-*

croservice Architecture: aligning principles, practices and culture. O'Reilly Media, Inc., 1st edition, 2016. URL https://books.google.fi/books?hl=en&lr=&id=Ev2wDAAAQBAJ&oi=fnd&pg=PP1&dq=microservice&ots=MbkTljoUms&sig=2ipIWNkXIANPdvipNYXNjwdOioQ&redir_esc=y#v=onepage&q&f=false.

[32] P. Newbold. ARIMA model building and the time series analysis approach to forecasting. *Journal of Forecasting*, 2(1):23–35, 1 1983. ISSN 02776693. doi: 10.1002/for.3980020104.

[33] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617–1634, 2014. ISSN 1553877X. doi: 10.1109/SURV.2014.012214.00180.

[34] A. Olsson, B. Welander, G. Rydén, J. Flodin, J. Hansryd, M. Nilsson, M. E. M. Coldrey, M. Öhberg, and S. Axelsson. Ericsson Microwave Outlook report 2022. 2022.

[35] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. 9 2016.

[36] Open Networking Foundation. Software Defined Networking Definition.

[37] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. 5 2019.

[38] D. B. Rawat and S. R. Reddy. Software Defined Networking Architecture, Security and Energy Efficiency: A Survey, 1 2017. ISSN 1553877X.

[39] P. Senin. Dynamic Time Warping Algorithm Review. Technical report, 2008.

[40] M. B. Shrestha and G. R. Bhatta. Selecting appropriate methodological framework for time series data analysis. *Journal of Finance and Data Science*, 4(2):71–89, 6 2018. ISSN 24059188. doi: 10.1016/j.jfds.2017.11.001.

[41] J. Stübinger and D. Walter. Using Multi-Dimensional Dynamic Time Warping to Identify Time-Varying Lead-Lag Relationships. *Sensors*, 22(18), 2022. ISSN 1424-8220. doi: 10.3390/s22186884. URL https://www.mdpi.com/1424-8220/22/18/6884.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin. Attention is All you Need. In I. Guyon, U. V. Luxburg,

S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[43] T. Warren Liao. Clustering of time series data - A survey. *Pattern Recognition*, 38 (11):1857–1874, 2005. ISSN 00313203. doi: 10.1016/j.patcog.2005.01.025.

[44] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A Survey on Software-Defined Networking, 1 2015. ISSN 1553877X.

List of Figures

List of Tables

Abbreviations and Acronyms

SDN	Software Defined Networking, network architecture in which the forwarding plane is decoupled from the control plane, to enable efficient and effective control and programmability of the network
API	Application Programming Interface
SBI	SouthBound Interface
NBI	NorthBound Interface
FWA	Fixed Wireless Access, connection enabled through a wireless connection between two points, using radio links [1]
Carrier	Communication channel
DTW	Dynamic Time Warping, metric used for time series clustering
PaaS	Platform as a Service
MSA	Microservice-based Architecture
ANN	Artificial Neural Network
PM	Performance Metrics, traffic counters
RAM	Random Access Memory, short-term memory where all data used in specific operations is temporarily loaded for processing
GPU	Graphics Processing Unit, component used in massive parallel computations and high-requirement tasks
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
GRU	Gated Recurrent Units

Acknowledgements

I would like to express my gratitude to my supervisor Professor Alexander Jung, who has helped in guiding this Thesis and providing support. I am also grateful to Ericsson and the whole SDN Department for providing a thriving and supportive work environment, with colleagues always ready to lend a helping hand. A special thank goes to Raquel Rodriguez, who has closely mentored me during these past months.

I am happy to have shared the time worked in Ericsson closely with Tommaso, who has represented someone I could entrust with my doubts.

I am forever grateful to my family, who has always shown me unconditional love and support in my decisions, throughout all these years.

My gratitude goes to Ines, who has accompanied me on this journey and has always been my safe harbour, no matter what.

To Mattia, thank you for all the time spent together. I hold preciously the friendship we have created.
To all of my friends, either from university: Andrea, Flavio, Federico, Riccardo, Nicolò, Francesca; or from my hometown: Andrea, Federico, Davide, Daniele, Marco, Matteo, Anna. You all hold a very special place for me.

Lastly, thanks to all the wonderful people I met through EIT in these past two years. Sharing this experience with all of you, as friends, made a lot of difference.

Thanks to everyone I got to meet, talk and share moments with during my university years.

Milan, September 25, 2023
Gabriele Rivi