



**Università
di Genova**

**DITEN | Dipartimento
di Ingegneria Navale,
Elettrica, Elettronica e
delle Telecomunicazioni**

**Development and evaluation of flow processing
schemes in a 5G Network Data Analytics
Function**

October 2026

Master's degree in internet and multimedia engineering

Candidate

M.Baraa Mardini

Supervisor

Prof. Roberto Bruschi

Assistant Supervisor

Prof. Chiara Lombardo

Dr. Paolo Bono

Contents

Contents	2
1 Introduction	4
2 State of the Art	6
2.1 Virtualization	6
2.1.1 Network Functions Virtualization	7
2.1.2 Cloud Computing	9
2.2 The Fifth Generation of Mobile Networks	10
2.2.1 Network Data Analytical Function	12
2.3 Forecasting	14
2.3.1 ARIMA (Autoregression and Moving Average)	17
2.3.2 DeepAR	19
3 NWDAF Prototype Architecture	22
3.1 DeepAR Prototype	24
4 Evaluation of Forecasting Methods	28
4.1. Dataset preparation	28
4.2. Experiment 1: High-Frequency Effect	33
4.3. Experiment 2: Comparison of Learning Strategies	33
4.4. Experiment 3: Resource consumption	39
4.5 Experiment 4: Live deployment test	43
5 Conclusions and Future Works	47

References	49
------------------	----

1 Introduction

Mobile cellular networks have evolved from simple voice-communication into complex digital infrastructures that power everything from smart cities and industrial automation to interactive multimedia. While earlier mobile generations such as 3G and 4G delivered higher data speeds by deploying dedicated, specialized physical hardware, 5G marks a fundamental turning point. Instead of treating the network as a static connectivity pipe, 5G transforms it into a flexible, software-driven service platform. Even though it still shares some similarity with 4G system, the breaking point of this system is introduced by 3 points, virtualization, Edging and expansion flexibility. Meaning that 5G system is composed of multiple types of Network Function (NFs) running in a virtual environment instead of consuming all the resources to run the system, you run exactly what you need, in this strategy you saved resources that can be used to deploy functions as needed depend on the system needs. However, this unprecedented flexibility introduces a major operational challenge. A modern cellular network contains thousands of interconnected cells and millions of connected devices, generating massive volumes of telemetry and highly variable traffic patterns. Managing these dynamic resources through traditional, manual, or reactive methods is no longer sufficient. In a reactive setup, the network detects performance degradations, traffic congestion, or packet drops only after they have already affected active users. permanently over-provisioning resources to prevent sudden bottlenecks wastes substantial computing power and energy. To guarantee strict Service Level Agreements (SLAs) and deliver a seamless Quality of Service (QoS), cellular networks must move away from reactive troubleshooting and transition toward proactive, automated closed-loop management.

To address this challenge, the 3rd Generation Partnership Project (3GPP) standardized the Network Data Analytics Function (NWDAF). Situated at the heart of the 5G core control plane, NWDAF acts as an internal intelligence engine. It continuously collects performance measurements from diverse network functions, analyzes operational trends, and exposes predictive insights back to the core orchestrators. By knowing what traffic demands will look like before they occur, network controllers can intelligently reallocate computing resources and scale services smoothly to prevent bottlenecks before subscribers experience any service disruption or outage.

While the architectural blueprint for NWDAF is well defined, implementing a reliable, real-time forecasting engine inside the network presents few challenges. Traditional statistical techniques, such as ARIMA and Seasonal ARIMA (SARIMA), have long been the standard for time-series modeling. However, mobile cellular traffic exhibits strong, high-frequency human cycles (such as 15-minute fluctuations across day and night). For classical models, computing high-frequency seasonal equations causes heavy mathematical overhead, often taking hours or days to calculate. Furthermore, these models are strictly local: they examine a single cell or city in isolation from the rest, meaning a model deployed in one region cannot learn from or share patterns with the rest of the network. Modern deep learning architectures, such as DeepAR, solve this isolation issue by training a single global model across thousands of data streams simultaneously. Yet, deep learning models are data-greedy and computationally demanding to train. In a live telecommunications environment, predictive accuracy is only half the conquest; inference speed, memory footprint,

and CPU consumption dictate whether an algorithm is a practical solution or merely a theoretical one. Another aspect is that cellular traffic is not static, human habits change across weekdays, weekends, and holidays, and networks experience unexpected anomalies. A predictive model cannot simply be trained once and frozen. It requires a balanced continual learning strategy that absorbs new data without overreacting to short-term noise or discarding foundational historical knowledge. Predictive models developed in offline scientific environments are rarely connected directly to active telecommunications systems. Proving that automated forecasting can reliably drive network operations requires building modular, fault-tolerant, A software that directly interfaces with 5G core components in real time.

The primary objective of this thesis is to develop, evaluate, and deploy practical predictive schemes within a 5G Network Data Analytics Function NWDAF, demonstrating how deep learning can be integrated into an operational 5G environment to achieve proactive network intelligence. To accomplish this, the thesis delivers three core contributions: *i)* creating a reliable prototype, *ii)* Data Preparation, *iii)* Algorithm Benchmarking to select the winning algorithm. I implemented a prototype using the Go language, composed of a Data Collector plugin and an Analytics Engine plugin equipped with an automated accuracy safety gate. Then Steps II and III has been done using the real-world NetMob2023 dataset consisting of 2.5 TB of cellular traffic traces recorded across 20 metropolitan areas in France, I develop compression algorithm to compress the dataset into a feasible size, the second purpose of the compression algorithm was to fix the dataset gaps like Daylight Saving Time shifts, and network outages. After preparing the dataset the first spark that ignited the research was comparing SARIMA against global DeepAR across 20 cities at a 15-minute resolution, evaluating forecast accuracy, execution speed, memory, and scaling overhead. That allowed me to have a clear picture of DeepAR constrains then the second stage was investigating how neural network models should be reinforced over time. By testing different rolling-window update intervals under zero-shot conditions (forecasting unseen metropolitan areas), I analyzed the trade-off between model responsiveness and stability, establishing an optimal update window that prevents noise error spikes and provides a reliable forecasting.

2 State of the Art

5G [1] (fifth generation of mobile networks) is a newer wireless technology that provides faster speeds, lower latency, and greater network capacity than previous generations like 4G. It enables applications such as smart cities, autonomous vehicles, IoT devices, and advanced mobile services. 5G has the ability to introduce new functionalities while it remained well integrated with the old system like 4G. Virtualization and, in particular, Network Function Virtualization (NFV) are extensively used in 5G, allowing the system to use resources in a dynamic and very fixable way by transforming services in on-demand, scalable functions that can be deployed, reallocated and canceled at any time. The need for managing physical and virtual resources in a dynamic fashion calls for the introduction of intelligence within 5G networks. To this end, the Network Data Analytical Function (NWDAF) has been conceived to collect and analyze data coming from the other network functions to provide forecasts on the network behavior in attempt to improve SLA and QoS that the system offers.

2.1 Virtualization

Virtualization is a technology that allows you to create virtual versions of physical resources such as computers, storage, networks and more. Using this technology, it is possible to run multiple virtual systems on a single physical machine, this improves resource utilization, flexibility, and cost efficiency. Another important advantage brought by virtualization is that resources are easier to be moved, copied and backed up since they are not tied directly to a physical machine. This makes migration, recovery, scaling, and disaster recovery faster and simpler. A Virtual Machine (VM) is a computing environment that functions as an isolated system with its own vCPU, operating system (OS), memory, network interface, and storage, created from a pool of hardware resources. A VM can be defined by a single data file. As an isolated environment, it can be moved from one computer to another, opened in either, and be expected to work the same.

Virtualization allows VMs with multiple different operating systems to run simultaneously on a single physical device. Each operating system runs in the same way an OS or application normally would on the host hardware, so the end user's experience is nearly identical to an operating system experience running on a physical machine [2].

While virtualization regards replacing the hardware, containerization allows software or applications to be packaged in a container that shares the host OS and can be moved and run in any environment for increased flexibility. With virtualization, VMs are created to run their own OS and applications. Virtualization allows multiple operating systems to run alongside each other and share the same virtualized hardware resources from a single physical machine. Figure 1 shows different types of virtualizations.

Virtualization architecture

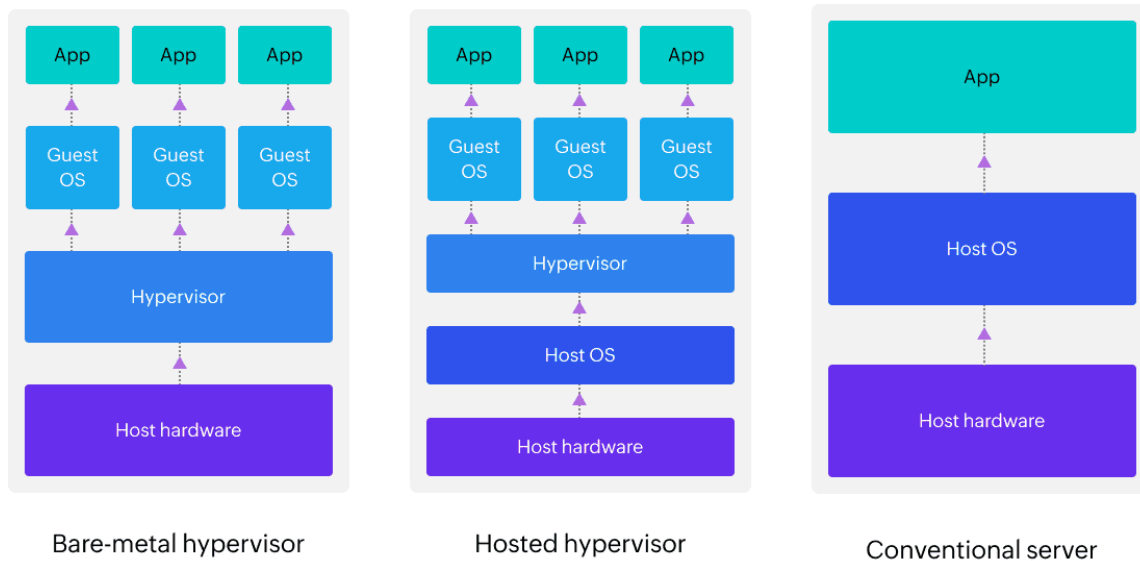


Figure 1: Different examples of virtualization.

2.1.1 Network Functions Virtualization

Network Functions Virtualization (NFV) [3] is a network architecture concept that virtualizes network functions to help organizations cut costs and optimize service deployment: while a traditional network appliance approach requires each network function such as load balancers, firewalls, gateways and routers to run on dedicated hardware, which can be expensive and difficult to scale, decoupling these network functions from hardware appliances enables service providers to greatly increase the speed of new service deployment while reducing the need for physical devices. The result is an agile network that is scalable, customizable and manageable. A centralized control panel enables network operators to automate the provisioning and orchestration of network resources and quickly respond to changing traffic patterns and network demands.

The NFV architecture shown in Figure 2 consists of three layers. Virtualized Network Functions (VNFs) are the services previously running on physical hardware (e.g., routers, firewalls, IP configuration and intrusion detection systems, etc.). Network Functions Virtualization Infrastructure (NFVI), offering connectivity to create a unified network out of multiple physical and virtual machines. Management, Automation and Network Orchestration (MANO): NFV MANO is the basic framework for managing the deployment, provisioning, monitoring and performance of virtualized network functions. NFV MANO also creates an interface for NFVI to

communicate and interact with existing operating support systems (OSS) and business support systems (BSS). MANO itself is further divided into three subsections:

- **NFV orchestration**
NFV uses virtualization technologies to deploy new network functions and provision resources to existing VNFs. NFV orchestration also serves to authenticate NFVI resource requests.
- **Virtualized infrastructure manager (VIM)**
VIM optimizes the lifecycle of software, virtual resources and physical network. VIM instances can manage multiple NFVI resources or specialize on a specific aspect as needed. A VIM keeps a record of virtual and physical resources allowing network operators to maintain operations and deploy new services.
- **Virtualized Network Function Manager(s)**
VNF standardizes virtual network functions and increases the interoperability of software-defined network (SDN) features. VNF management includes the instantiation, or creation of, instances, scaling, upgrading and termination of virtualized network functions.

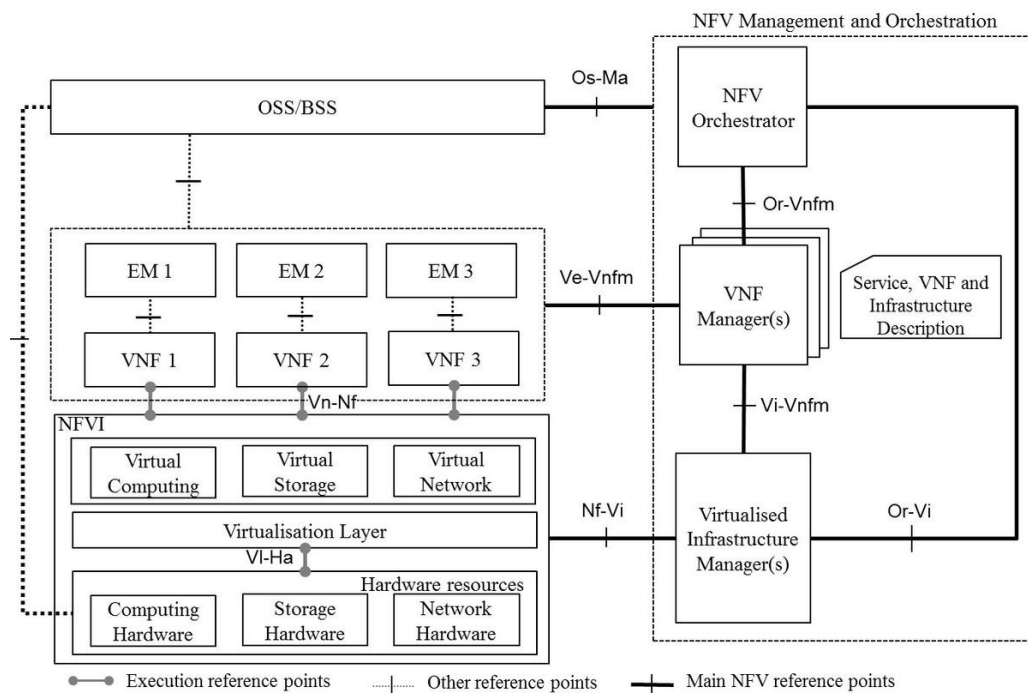


Figure 2: the NFV architecture [4].

2.1.2 Cloud Computing

Cloud computing is the on-demand access to computing resources (physical or virtual servers, data storage, networking capabilities, etc.) over the Internet with pay-per-use pricing [5]. When we use cloud services, we access remote servers, powerful mainframe computers housed in large data centers, through the internet. The cloud computing model gives you, the customer, greater flexibility and scalability compared to traditional, on-premises infrastructure. Compared to traditional on-premises IT, where a company owns and maintains physical data centers and servers to access computing power, data storage and other resources, cloud computing offers many benefits, Cost-effectiveness, Increased, speed and agility, Unlimited scalability, and enhanced strategic value.

Infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), software-as-a-service (SaaS) and serverless computing are the most common “as-as-service” cloud platform models. Most developers at large-scale organizations use some combination of all four. **IaaS** provides on-demand access to computing resources, physical and virtual servers, networking and storage over the internet. While **PaaS** provides software developers with on demand platform hardware, complete software stack, infrastructure and development tools for running, developing and managing applications without the cost, complexity and inflexibility of maintaining that platform on-premises. And **SaaS** also known as cloud-based software or cloud applications, is interactive application software hosted in the cloud. Users access SaaS through a web browser, a dedicated desktop client or an application programming interface (API) that integrates with a desktop or mobile operating system.

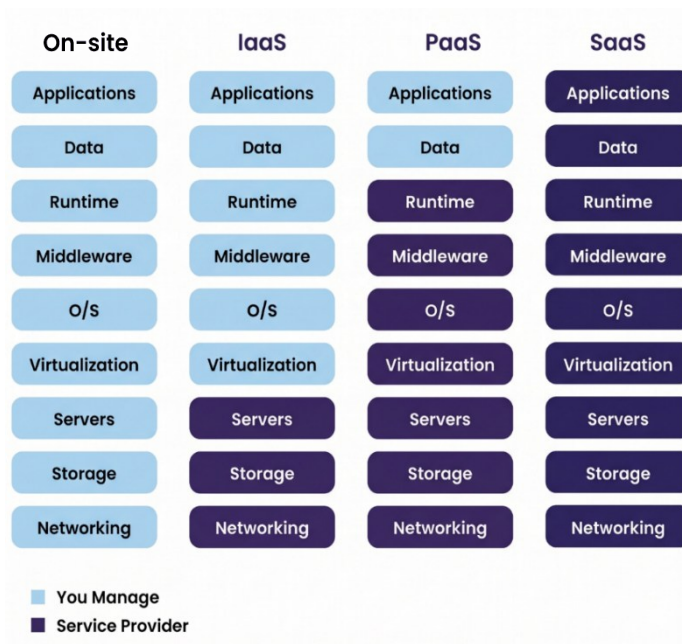


Figure 3: different cloud architectures

In Figure 3, we can notice that IaaS offers full control over IT infrastructure, allowing organizations to build and manage systems. PaaS builds on IaaS by providing a platform that simplifies the development and deployment of applications, handling the underlying infrastructure

for you. SaaS, the most widely used cloud service, delivers ready-to-use software, removing the need for management. And serverless computing, built on IaaS and PaaS, lets you focus solely on writing code.

2.2 The Fifth Generation of Mobile Networks

5G networks are supporting new applications with stringent performance requirements with respect to previous generation 4G and require very specific Key Performance Indicators (KPIs), shown in Figure 4 depending on the service offered.

Like previous mobile network generations, the core network provides connectivity to services while giving the service provider the functions they need to authenticate, authorize, and manage their customers. But, when it comes to the extended capabilities of a 5G core, it surpasses previous generations: while they had a monolithic architecture designed to support the most extreme requirements 24/7 and for all applications, the 5G architecture is service based, that means architecture elements are defined as VNFs offering their services via interfaces of a common framework and other VNFs can consume such services if authorized. For this reason, the 5G core makes it simpler to integrate new functions using a message bus concept for control plane interactions. It implements representational state transfer (REST)-ful application programming interfaces (APIs) that use hyper-text transport protocol (HTTP/2) inquiries for high-level flexibility.

One of the many advantages of a 5G core is that the location of network functions can be optimally placed to ensure the greatest efficiency, highest performance, and lowest latency, providing the best possible experience. The 5G core network transformation is part of a much larger telecommunications business transformation. Their networks are no longer viewed as a connectivity pipe for service delivery. Instead, they are a service platform providing various opportunities [6].

Past generations of mobile networks are defined by several factors including the technology used, the speed of data transmission, and network latency. The first-generation networks introduced basic wireless voice calling. The second generation introduced short message service (SMS) and multimedia message service (MMS), while the third generation significantly increased data transmission speeds from 200 kbps to 40 Mbps. The fourth generation introduced new capabilities and more than doubled data speeds to 100 Mbps. Fifth-generation networks will make speeds of 10 Gbps possible. 5G networks will bring faster speeds and greater immediacy that will be almost indistinguishable from real time. A key differentiator of 5G (vs. 4G) is the ability to support up to one million connections per square kilometer. This is the kind of density that will be necessary to support smart cities, smart homes, self-driving cars, and the growing internet of things (IoT).

Consumer and business expectations are constantly changing, placing ever-growing demands on 4G network capabilities. Both the demand for and opportunities provided by this latest generation of technology is why many telecommunications service providers are currently transitioning from a 4G to a 5G core network. Figure 4 shows the advantages of 5G system over the 4G system.

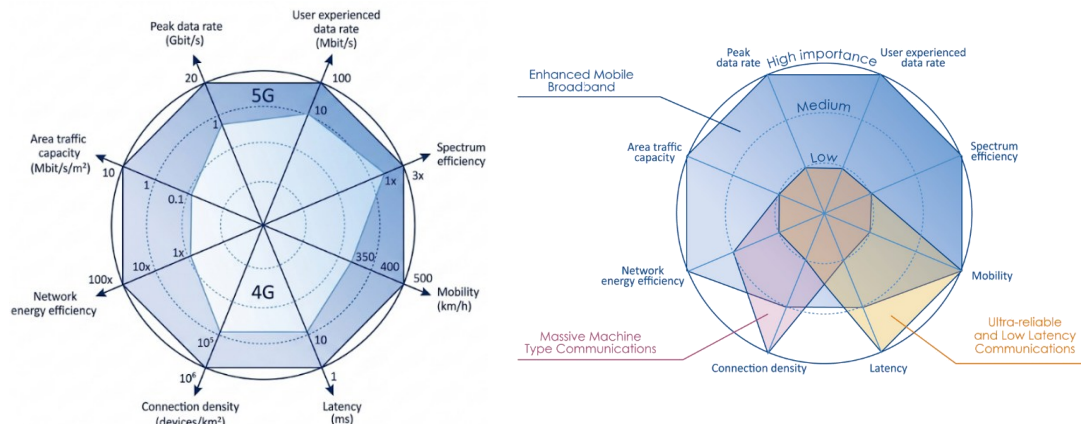


Figure 4: The Key Performance Indicators of the 5G system.

5G specifies three use case categories:

- **Enhanced mobile broadband (eMBB)**
5G enhances multimedia experiences with higher bandwidth allocation and throughput.
- **Ultra-reliable, low-latency communication (uRLLC)**
5G supports vertical industry requirements for mission-critical and latency-intolerant applications with sub-millisecond latency.
- **Massive machine type communications (mMTC)**
5G empowers cost-efficient and robust connection for large numbers of devices without network overloading.

A 5G core has a service-based architecture with orthogonal functions that allow for independent scaling, whereas EPC functions have overlapping responsibilities and are based on point-to-point architecture.

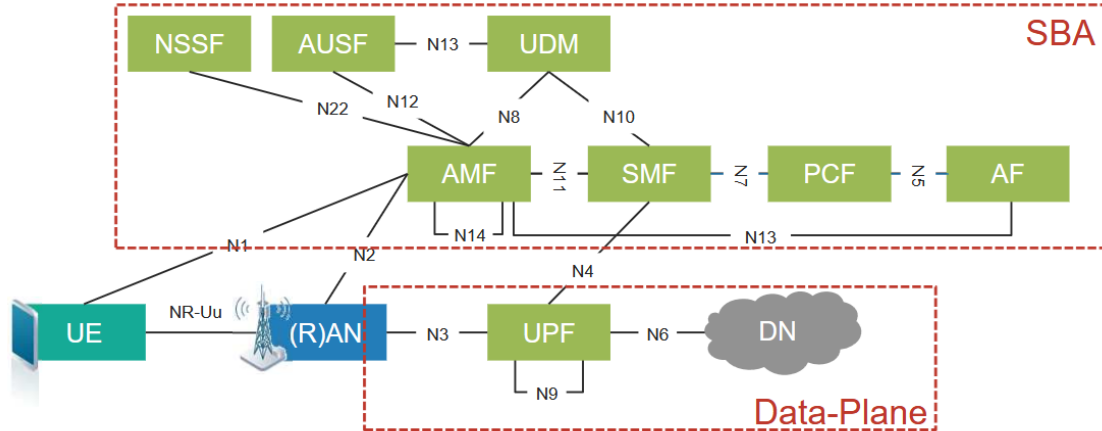


Figure 5: the internal structure of the 5G system

Using container-based technology means that the microservices that constitute a 5G core, shown in Figure 5, are packaged with everything required to operate. This includes libraries and application-specific dependencies, all within a dedicated and isolated space. Failure of an individual component or microservice will not cause the overall 5G core application to fail. When built in this manner, 5G core applications have more predictable behavior in terms of fault tolerance, scalability, and their use of the underlying cloud resources.

2.2.1 Network Data Analytical Function

A 5G network contains a vast number of devices and sensors generating an enormous amount of data. The Network Data Analytical Function (NWDAF) is a 5G core network function that collects, analyzes, and provides insights from network data. This implementation features a plugin-based architecture supporting multiple 5G core vendors and custom analytics algorithms. The analytics information provided by the NWDAF is either statistical information based on past events or predictive information. This analytics information can be used to balance the resources on the network, predict the User Equipment (UE) location and detect if the UE is in an abnormal location.

3GPP has specified a network automation architecture to leverage 5G information exposure and network data analytics. It enhances network performance, operational assurance and resource utilization and enables self-optimization and automated management [7]. NWDAF is designed to fulfill those needs in a step forward toward automated networks, also to overcome market fragmentation and proprietary solutions in the area of network analytics, streamlining the way core network data is produced and consumed, as well as generating insights and taking actions based on these insights.

Beyond specification, modern NWDAF implementations emphasize flexibility and real-time processing. architecture integration with multi-core compatibility across platforms such as Free5GC, Open5GS, HPE, and OpenAirInterface. Which makes it extremely feasible to implement predictive models ranging from classical techniques like ARIMA/SARIMA to advanced machine learning these systems enabling network automation within cloud-native, containerized environments.

As shown in Figure 6, upon reception of the analytics request from the consumer, the NWDAF collects network data, service data, management data and/or UE performance data from dedicated data sources, then processes and generates the reliable analytics information to assist the consumer to improve the user experience and optimize the network performance. In order to ensure more accurate calculation, prediction, and decision, the NWDAF supports computing the accuracy of the ML models and analytics, Federated Learning, i.e., training a ML Model across multiple decentralized NWDAF instances without exchanging and sharing the local data set in each NWDAF, and the analytics and data exchange in the roaming scenario.

The usage of the NWDAF can be beneficial in many scenarios. For instance, it can improve service experience by handling the Mean Opinion Score (MOS) for services, like the audiovisual streaming or the non-audiovisual services. NWDAF collects the Quality of Experience (QoE) metrics and performance data, e.g., the average packet delay, average packet loss rate and throughput from the Application Function. Other input data includes the QoS flow level network data from the 5GC NFs and the radio-related data. With the collected data, the NWDAF can offer diverse, multi-level analytical capabilities that support fine granularity experience prediction and analysis. For instance, the NWDAF can provide analytics for an individual UE or a group of UEs, different access types, multiple applications, as well as different network slices. The operator takes the analytics into account to measure the actual user's service experience and to identify the opportunities for network optimization.

Additionally, it can be beneficial for analyzing the load status of 5G NFs to assist the consumer in optimizing the NF selection process. Once the target NFs are indicated by the consumer, the NWDAF will interact with the NRF to collect the targeted NF's load status and the resource usage related data, e.g., the usage of virtual CPU, memory, and disk from the OAM. Further, the NWDAF can also retrieve the traffic usage reports from the UPF and the UE movement information from the AF. With this data, the NWDAF produces the NF load analytics result including the NF type, NF instance ID, NF status, NF resource usage, NF load, and NF peak load. The NF load analytics is very helpful for an operator for, e.g., capacity planning. Some NFs can use it to select less-loaded NFs, e.g., SMF selection by the AMF or UPF selection by the SMF to improve the network

In summary, since the introduction of 5G, 3GPP has already specified various intelligent network and Network Automation use cases, mainly focusing on optimizing network resources utilization and improving user experience.

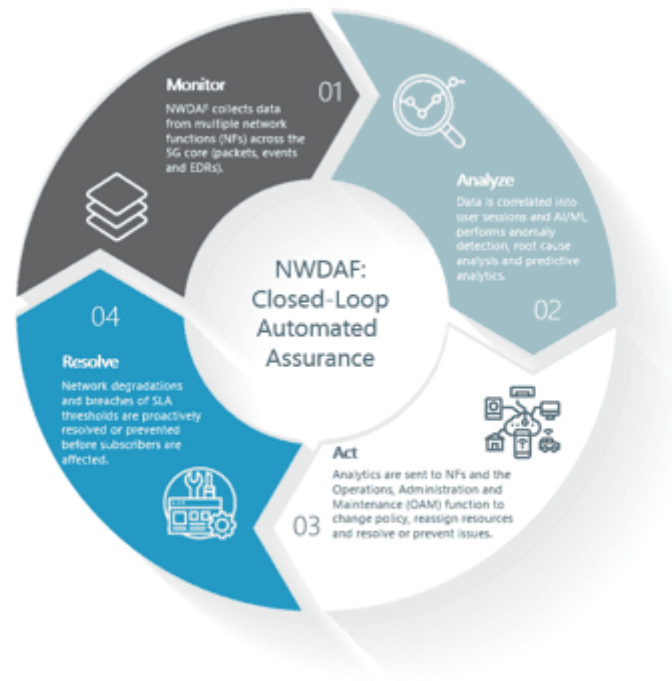


Figure 6: NWDAF process cycle

2.3 Forecasting

Forecasting is a method of predicting a future event or condition by analyzing patterns and uncovering trends in previous and current data. It employs mathematical approaches and applies statistical models to generate predictions. In this thesis, the forecasting algorithm will be used to predict the demands of the customers in order to improve the SLA and QoS. The appropriate forecasting methods largely depend on what data is available. If there is no data available, or if it is not relevant to the forecasts, then qualitative forecasting methods must be used. These methods are not purely guesswork—there are well-developed, structured approaches to obtaining good forecasts without using historical data. Quantitative forecasting can be applied when two conditions are satisfied: *i*) numerical information about the past is available, *ii*) it is reasonable to assume that some aspects of the past patterns will repeat in the future.

Time series are simply a sequence of data points in chronological order. Time series analysis is the method of identifying and characterizing patterns in the time series. The following may occur, which contain multiple characteristics and we can recognize. Trend: a trend exists when there is a

long-term increase or decrease in the data, it is not necessarily linear. Seasonality: a seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality always has a fixed and known period; a single series can have more than one seasonal pattern. Cyclic: a cycle occurs when the data exhibits rises and falls that are not of a fixed frequency. These fluctuations are usually due to economic conditions and are often related to the “business cycle”. The duration of these fluctuations is usually at least 2 years[20].

Mean method forecasts all future values are equal to the average (or “mean”) of the historical data. For naïve forecasts, we simply set all forecasts to be the value of the last observation. Another version of this method has been created which takes into consideration the seasonality and it’s called Seasonal naïve method, in this case, we set each forecast to be equal to the last observed value from the same season the fourth simple method is Drift method, It is a variation on the naïve method is to allow the forecasts to increase or decrease over time, where the amount of change over time (called the drift) is set to be the average change seen in the historical data. Thus, the forecast for time $T + h$ is given by[21] .

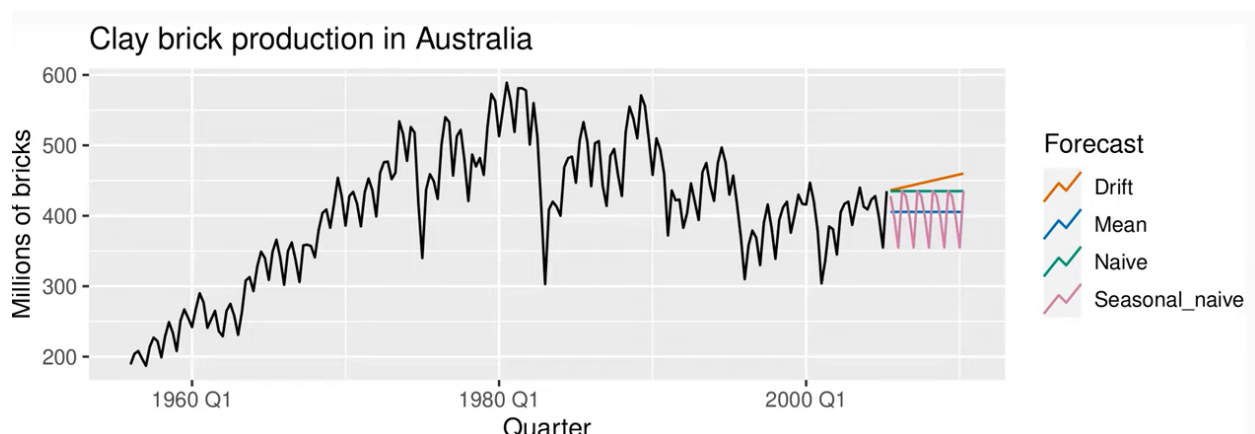


Figure 7: An example of simple forecasting methods

As shown in the Figure 7, a time series is plotted and four naïve methods are being used to predict the future pattern, as a benchmark for any future algorithm we may apply. In other words, any other algorithm must perform better than those algorithms. It is important to evaluate forecast accuracy using genuine forecasts. Consequently, the size of the residuals is not a reliable indication of how large true forecast errors are likely to be. The accuracy of forecasts can only be determined by considering how well a model performs on new data that were not used when fitting the model.



Figure 8: visual illustration of the splitting of the dataset

Finally, when choosing models, it is common practice to separate the available data into two portions, **training** and **test** data, represented in Figure 8, where the training data is used to estimate any parameters of a forecasting method and the test data is used to evaluate its accuracy. Because the test data is not used in determining the forecasts, it should provide a reliable indication of how well the model is likely to forecast on new data.

Figure 9 illustrates the series of training and test sets, where the blue observations form the training sets, and the orange observations form the test sets. The forecast accuracy is computed by averaging over the test sets. This procedure is sometimes known as “evaluation on a rolling forecasting origin” because the “origin” at which the forecast is based rolls forward in time.

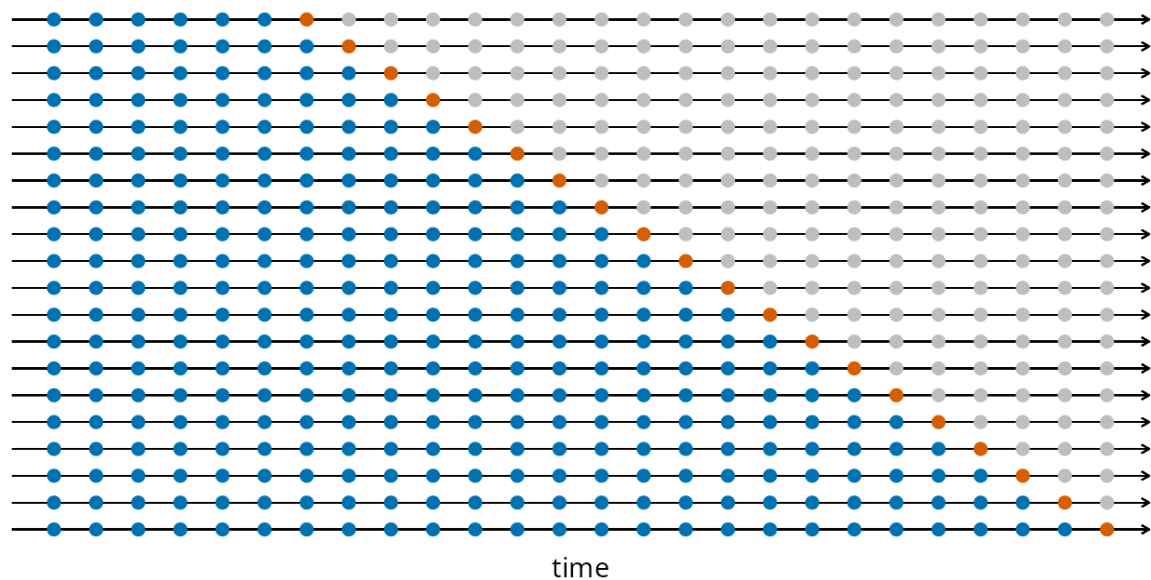


Figure 9: visual illustration of one-step forecasts

With time series forecasting, one-step forecasts may not be as relevant as multi-step forecasts. In this case, the cross-validation procedure based on a rolling forecasting origin can be modified to allow multi-step errors to be used. Suppose that we are interested in models that produce good 4-step-ahead forecasts. Then the corresponding diagram is shown in Figure 10.

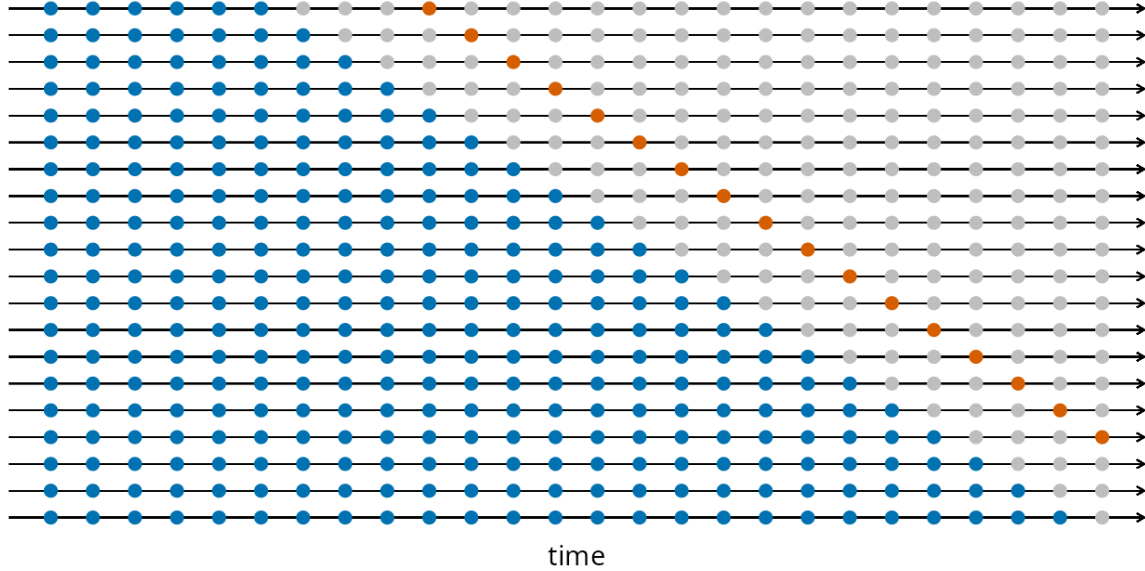


Figure 10: visual illustration of multi-step forecasts

2.3.1 ARIMA (Autoregression and Moving Average)

As its name indicates, the acronym ARIMA integrates Autoregression and Moving Average models into a single model depending on the parameters passed. These two ways of modeling change throughout the time series are related but have some key differences. In an autoregression model, we forecast the variable of interest using a linear combination of past values of the variable. The term auto-regression indicates that it is a regression of the variable against itself. This technique is similar to a linear regression model in how it uses past values as inputs to the regression. There are two types of ARIMA: no-seasonal ARIMA and Seasonal-ARIMA (SARIMA) [8]. In the remaining of the thesis, the focus will be on SARIMA since mobile traffic has considerable seasonality. The degree to which a time series is correlated with its past values can be investigated by calculating the autocorrelation, which can answer questions about whether the data exhibit randomness and how one observation is related to an immediately adjacent observation. This can give us a sense of what sort of model might best represent the data.

Another important metric in preparing to use an ARIMA model on time series data is the Partial Autocorrelation Function (PACF). An ACF shows the relationship between y_t and y_{t-k} for different values of k . If y_t and y_{t-1} are correlated, then y_{t-1} and y_{t-2} will also be correlated. But it's also possible for y_t and y_{t-2} to be correlated because they are both connected to y_{t-1} , because of any new information contained in y_{t-2} that could be used in forecasting y_t . To overcome this problem, we can use partial autocorrelations to remove a number of lag observations. These measure the relationship between y_t and y_{t-k} after removing the effects of lags one to k . So, the first partial

autocorrelation is identical to the first autocorrelation, because there is nothing between them to remove. Each partial autocorrelation can be estimated as the last coefficient in an autoregressive model. Moving average model on the other hand uses the past forecast errors rather than using past values of the forecast variable in a regression. A moving average simply averages k values in a window, where k is the size of the moving average window, and then advances the window. The forecast values are evaluated using actual values to determine the error at each step in the time series.

ARIMA model uses either the Auto-Regressive term (AR) term or the Moving Average term (MA). The ACF plot and PACF plot are oftentimes used to determine which one of these terms is most appropriate. Despite its popularity, ARIMA has certain drawbacks. We are going to mention few of them, for example ARIMA models works on linear data, it cannot predict non-stationary or non-linear data, also may not be suitable for long-term forecasting. The MA component, which partially relies on past forecast errors, causes these errors to accumulate over time, leading to less accurate forecasts as the forecast horizon increases. Similarly, for the AR component, the influence of current values on future forecasts diminishes as the forecast horizon lengthens. It is worth noticing that parameter selection is another challenge, as it introduces subjectivity and requires considerable expertise to select suitable parameters. In mobile traffic there is what is called The Curse of High-Frequency Seasonality: SARIMA relies on the seasonal parameter m to dictate the number of time steps in a single seasonal cycle, and its state-space matrices expand exponentially as m increases. Attempting to fit an Auto-ARIMA model with large m will result in massive memory overhead, which can take hours or even days to optimize a single sequence. Lastly, SARIMA is strictly a local model. It evaluates one time series in an isolation, e.g., to forecast traffic for 20 cities across 10 applications, we have to instantiate, train, and maintain 200 entirely separate SARIMA models. Furthermore, if a new application trend emerges in city, the SARIMA model for another city learns nothing from it.

Even though ARIMA Models can be powerful yet simple forecasting models, they suffer from a lot of cons considering the complexity of mobile traffic. A better alternative would be moving toward newer approaches like DeepAR.

2.3.2 DeepAR

DeepAR combines several advanced machine learning concepts, but at its core, it is simply a neural network designed to predict probability distributions. The main features of this model are Global Joint Training: as shown in Figure 11, instead of training one model per time series, DeepAR trains a single, global model across hundreds or thousands of related time series. This allows the model to learn shared patterns across the entire dataset, in case a new pattern appears in a city allows the other cities to benefit from the shared knowledge. Traditionally, most forecasting methods have been developed in the setting of forecasting individual time series. In this approach, DeepAR independently estimates model parameters from past observations for each given time series.

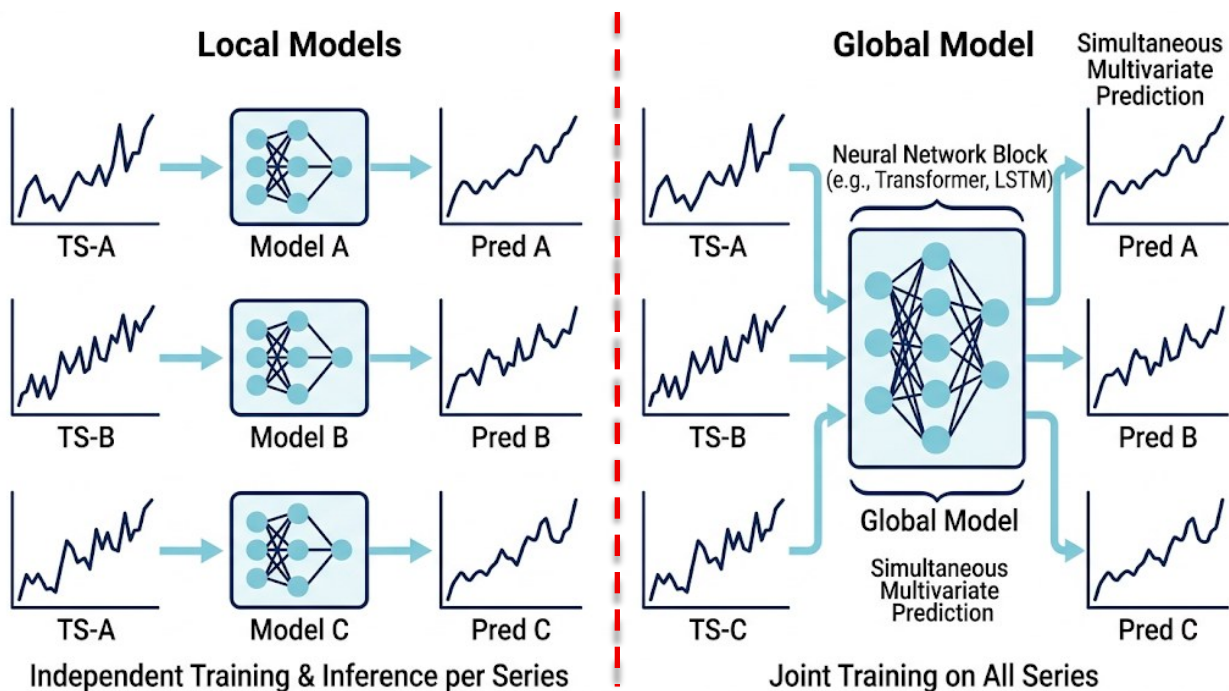


Figure 11: Comparison between local model prediction versus Global model prediction for multiple series

At the algorithmic core of DeepAR is an autoregressive recurrent neural network architecture, which can be seen in Figure 12 and is typically constructed with Long Short-Term Memory (LSTM) cells. While standard recurrent neural networks struggle to retain information over long sequences due to vanishing memory, LSTM units use dedicated internal gating mechanisms to selectively retain long-term historical context while filtering out short-term noise [23].

A defining attribute of DeepAR is that it does not generate deterministic single-point forecasts. Instead, it outputs the parameters of a specified probability distribution that models the statistical properties of the target data. For example, when modeling continuous real-valued targets, the

network maps its hidden state to the mean and standard deviation of a Gaussian distribution. For count or bursty demand data, it can output the parameters of a negative binomial distribution. This design allows DeepAR to capture the underlying uncertainty of the system dynamically rather than assuming a static noise level.

AUTOREGRESSIVE PROBABILISTIC FORECASTING LOOP (WORKFLOW DIAGRAM)

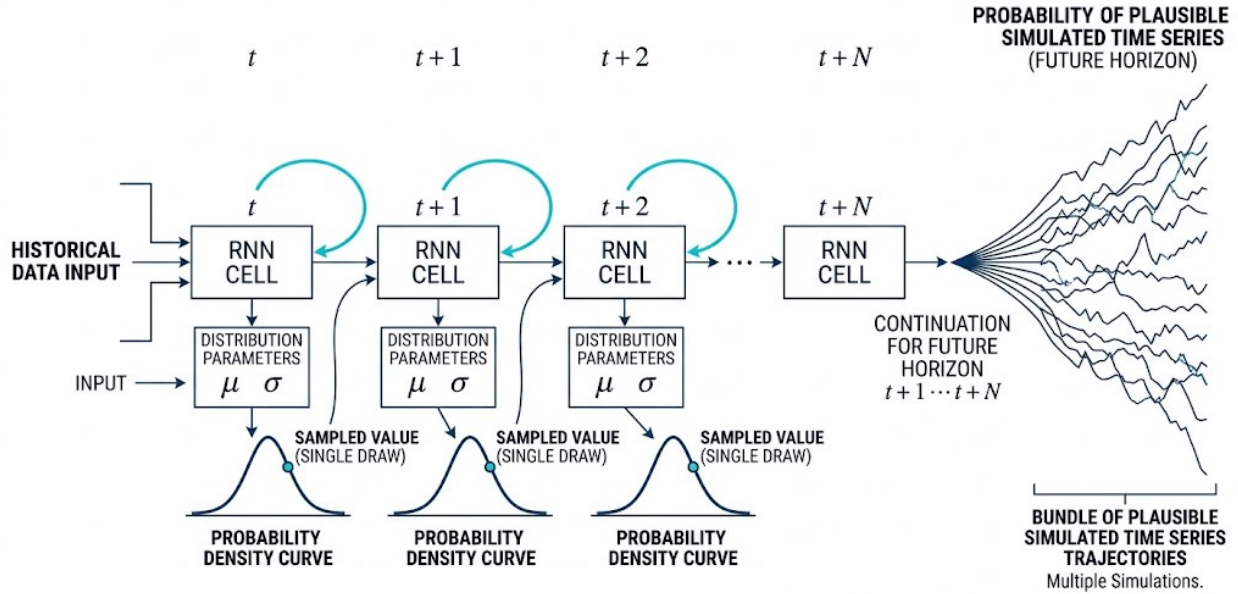


Figure 12: DeepAR forecasting architecture

This distribution centric architecture creates an intuitive distinction between training and inference. During training, the historical ground truth is fully observed, allowing the network to consume true previous values at every time step and adjust its weights by directly maximizing the log likelihood of the observed data. During inference, however, future ground-truth values are unknown. DeepAR resolves this autoregressively through ancestral sampling: at each forecast step, the model computes the distribution parameters, draws a random sample from that distribution, and feeds that sampled value back into the next step as the simulated previous observation. By repeating this sampling process many times, the model generates an ensemble of diverse future trajectories that accurately reflect cumulative uncertainty and temporal correlations over the entire forecasting horizon, see Figure 13.

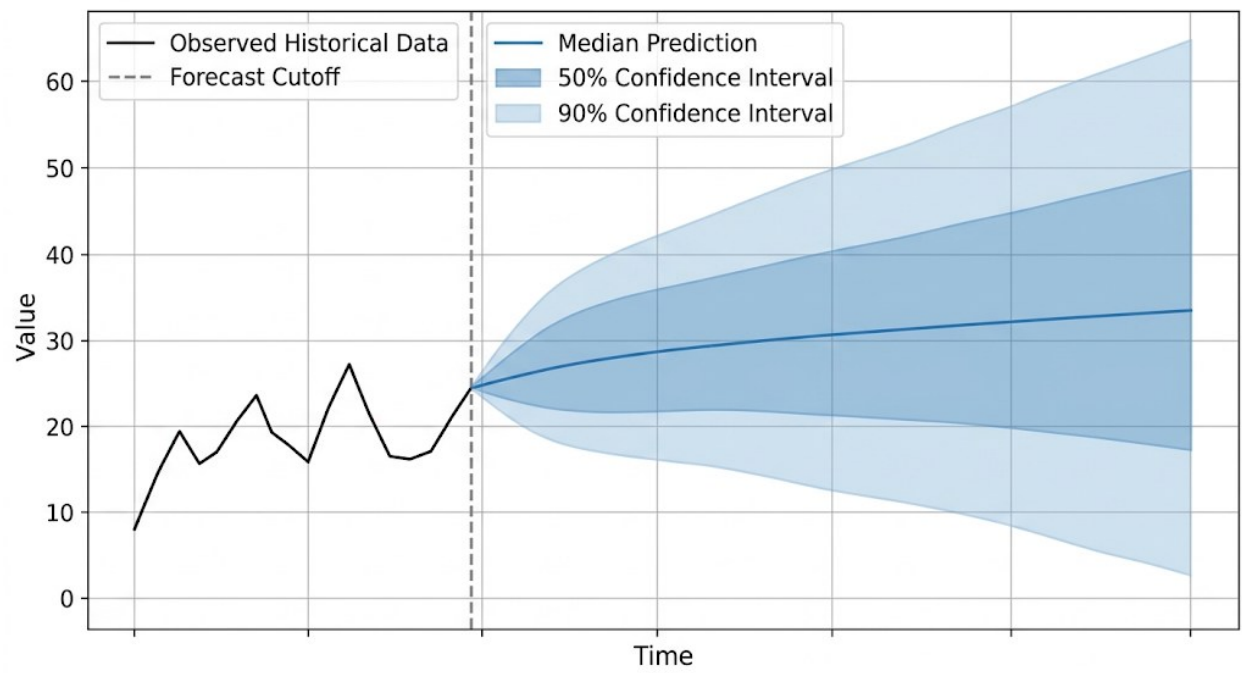


Figure 13: DeepAR prediction, illustrating how the accuracy degrade further we look toward the future

3 NWDAF Prototype Architecture

In this thesis, an open-source prototype of NWDAF [13]-[15] will be used to implement new algorithms and compare the results through a simulation of a real-life scenario. This section will describe the prototype architecture and its components, as well as the two software plugins developed in this thesis. Before jumping to the prototype, we must introduce the architecture of the NWDAF and understand its components. The architecture is illustrated for visual clarification on the right-hand side of Figure 14, and we can notice that the exposure points are in shape of plugins.

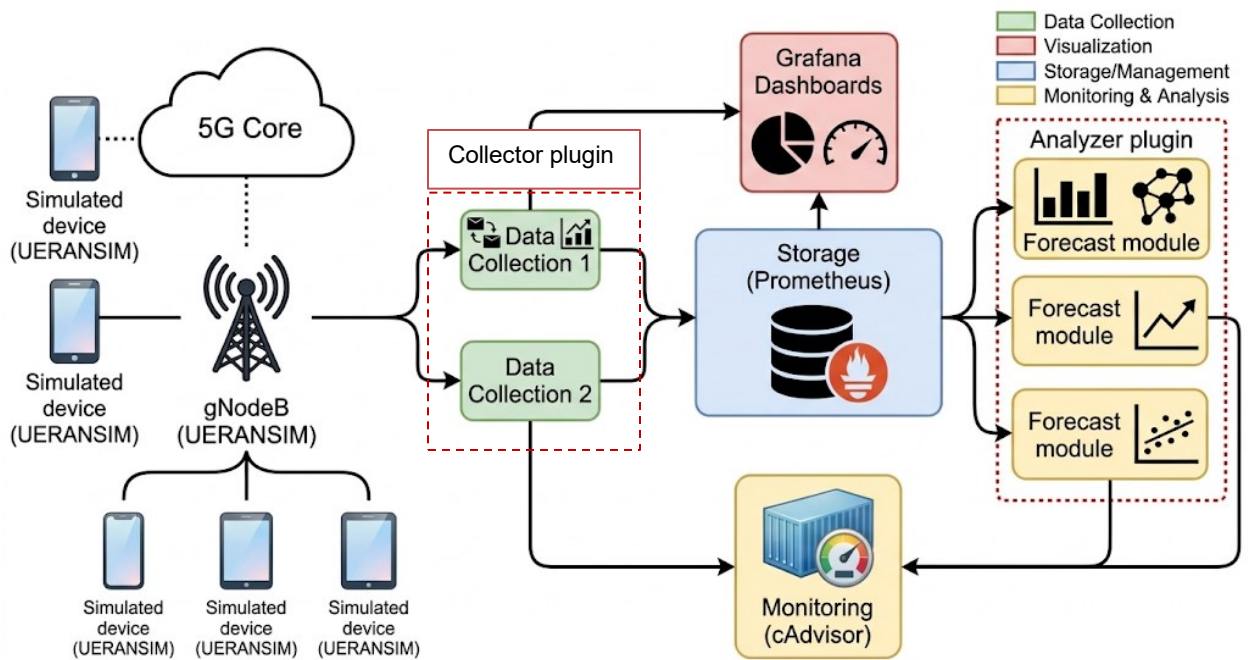


Figure 14: NWDAF Architecture

The adoption of a plugin concept brings several advantages, one of the main is process isolation: each plugin runs as an independent operating system process, rather than loading shared libraries directly into the host runtime, the system establishes clear fault domains and security boundaries. As a result, fatal exceptions or unexpected crashes within a plugin remain contained, preventing a catastrophic failure of the main application. Furthermore, executing plugins within separate memory spaces eliminates unauthorized access to host memory, simplifies resource tracking, and prevents memory leaks from propagating across the core system. This structural decoupling also grants significant flexibility, allowing individual extensions to be initialized, terminated, or dynamically reloaded without requiring a restart of the primary service. The life cycle of the plugin in general is illustrated in Figure 15.



Figure 15: The plugin life cycle

The monitoring framework is built around several independent components that collaborate smoothly without being rigid or tightly coupled. At its foundation, architecture brings together automated collectors that act as observers across the network, a central messaging hub that directs traffic between services, an analytics engine responsible for interpreting raw numbers, a temporary storage archive, and an outward-facing interface that presents the results. The information travels through this setup in a continuous, orderly journey. The process begins when the collectors gather live operational measurements directly from active network functions and deliver these raw readings to the central messaging hub. The analytics engine immediately picks up the incoming stream, filters and combines the numbers, and calculates high-level performance insights, which it routes right back through the hub. Meanwhile, a dedicated archiver listens in, saving both the original observations and the refined calculations into the storage system so that recent operational history remains easily accessible. Finally, network engineers and external monitoring tools can query and visualize these metrics in real time through web dashboards and software interfaces, giving them a clear, immediate window into how the entire system is performing.

Having established this end-to-end operation workflow, the following subsections examine the internal design and responsibilities of each fundamental architectural building block in detail.

The Data Collector serves as the platform's primary sensory system, interfacing directly with active network components to gather ongoing operational measurements. Because different parts of the network expose information in slightly different ways, the collector delegates this work to modular, independent workers tailored to specific network services. These workers query their assigned targets at regular intervals in an isolated manner, ensuring that if a single task encounters

an issue, the wider collection process remains completely stable. Once retrieved, the raw measurements are organized into a uniform structure and handed off to the central messaging system for downstream processing. After that the Analytics Engine examines these incoming measurements over time, detecting evolving trends and generating forward-looking predictions. Designed with flexibility at its core, this module allows different forecasting approaches, statistical aggregations, or machine learning models to be easily swapped in or updated without redesigning the platform. To maintain dependable and stable predictions, the analytical routines hold incoming observations in temporary working buffers until enough historical context has been gathered to perform a reliable analysis. Once new insights or future horizons are calculated, they are sent back into the messaging stream and made available for external management systems to consult on demand. Finally, the Data Archiver manages the storage and lifecycle of all telemetry, linking the streaming message flow with historical databases and outward-facing dashboards. It listens continuously to both the raw observations and the refined analytical forecasts, attaching precise arrival timestamps to keep the system organized for live monitoring. To prevent the platform from becoming congested with outdated information, the archiver automatically tracks data freshness: actively updated metrics are kept ready for visualization, while records that cease to receive fresh updates are systematically phased out and cleared from memory.

3.1 DeepAR Prototype

This section is dedicated to explaining the contribution that has been made to this thesis, namely, the implementation of DeepAR forecasting technique in NWDAF prototype described in the previous section. The description will present the prototype that has been made and how it was integrated into the 5GC system, while Section 4 will be dedicated to testing and discovering good strategy performance wise, regarding reasonable resources consumption.

As was mentioned, the NWDAF prototype is using the plugins architecture separated into two types of plugins, collectors that are responsible to collect the specific metric and save it to the database, analyzers the forecasting algorithm that is subscribed to the metrics that have been collected. Then two developed plugins will be presented in detail. The software plugin prototype was implemented using Go language to implement DeepAR, while C++ is considered the fastest between Python and Go, but it has the worst development curve. Python is amazing for prototyping speed, but when it comes to code pure performance it's the worst. For those reasons, Go Language is the best trade-off in this case between developing speed and code performance. My objective of making this prototype is to demonstrate that DeepAR algorithm is implementable and compatible with 5G core systems and offers significant improvement. The proposed prototype operates the NWDAF framework through two decoupled, complementary components: a collector plugin and a predictive analyzer plugin. Developed in Go to balance high-throughput concurrency, these

plugins interface directly with the core control plane and database layers. The Figure 16 present this structure.

Collector plugin: serves as telemetry collector for a 5G Network Data Analytics Function (NWDAF), gathering performance data across key core components: the AMF, SMF, and UPF. Its primary purpose is to monitor network activity by tracking the number of connected user devices, active data sessions, server CPU and memory usage, bandwidth (throughput in Mbps), and packet drop rates.

The plugin is built with a dual operational design: in a live deployment, it authenticates with the HPE 5G Core through REST APIs and queries an internal Prometheus database to pull real system telemetry. When a physical testbed is not available, it switches to a simulation mode that automatically generates realistic, mathematically correlated synthetic traffic—ensuring higher traffic volume naturally aligns with higher CPU load and packet drops. Finally, it can run either as a standard background plugin integrated into the NWDAF architecture or in a local debug mode that samples and prints metric updates to the console every 15 seconds for testing.

Analytics plugin: serves as the predictive intelligence engine within NWDAF, taking the raw traffic streams and transforming them into continuous, future-looking forecasts. It operates by first aligning incoming uplink and downlink measurements into synchronized time intervals, ensuring that the machine learning pipeline always receives clean, orderly data.

To avoid any operation delay when the system first starts up, the plugin immediately provides forecasts using recent observations and a pretrained model while it silently gathers enough history to begin training the actual DeepAR model. Once a sufficient number of observations has been collected, it automatically triggers a background learning task to train the DeepAR neural network on actual network patterns. As new traffic data continues to arrive over time, the plugin continually refines the model by carrying forward its existing knowledge and updating its weights in periodic increments, directly reflecting the continual learning strategy. Crucially, the plugin has an automated safety gate: every newly trained candidate model is rigorously tested against recent history, and it is only promoted to active service if its accuracy strictly outperforms both the simple baseline and the currently running model. If a candidate fails this quality check or encounters an issue during training, it is safely discarding the model without disrupting service, allowing the proven model to keep forecasting uninterrupted. Operating quietly alongside this background training, a dedicated prediction worker refreshes a multi-step forecast after every single observation interval—pausing during inactive periods when the cell is idle to provide network controllers with an accurate, real-time preview of upcoming traffic demand.

Figure 16 illustrates the execution flow and decision logic governing the analytical plugin's forecasting lifecycle. Upon initialization, the pipeline continuously ingests streaming 5G system metrics delivered by the Data Collector. To guarantee continuity and prevent downtime during initial startup or buffer flushes, the first algorithm verifies whether a sufficient volume of historical observations has been gathered. If the historical buffer is insufficient, the system activates a

fallback model to produce baseline predictions immediately, ensuring that downstream monitoring consumers receive uninterrupted telemetry while observations continue to accumulate.

Once the observation window satisfies the required data threshold, the framework triggers the training cycle for a candidate DeepAR model. To prevent performance degradation from underfit or anomalous training runs, the pipeline incorporates an Accuracy Safety Gate before model deployment. This gate evaluates whether the candidate model achieves a lower Mean Absolute Error (MAE) compared to the currently running model. If the candidate demonstrates superior accuracy, it displaces the inferior model and serves inference, otherwise, the pipeline retains the existing DeepAR model. The active model then computes multi-step predictions across the configured prediction horizon ($t + 1, \dots, t + n$) and dispatches them to the NWDAF interface and the Data Archiver, seamlessly looping back to ingest the next stream of metrics.

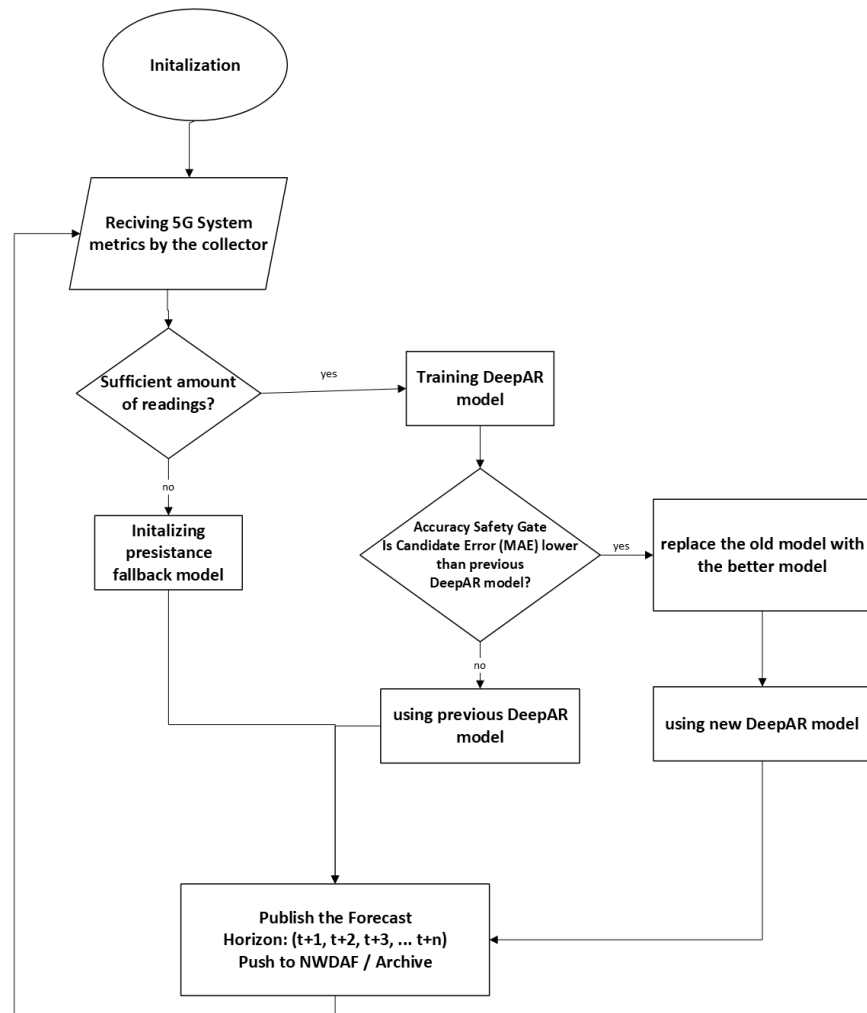


Figure 16: Analytical plugin algorithm flow

The main value of this prototype is demonstrating that advanced predictive model can be embedded directly into an active 5G network to anticipate real-world traffic demand accurately and reliably. By integrating the DeepAR algorithm into the analytics architecture, the system translates fluctuating operational measurements into continuous, forward-looking forecasts, giving network controllers the foresight needed to allocate resources and resolve bottlenecks before congestion occurs.

To ensure practical and safe operation, the prototype delivers immediate predictions upon startup, refines its knowledge in the background as new usage patterns emerge, and enforces an automatic safety check that activates updated models only when they demonstrably outperform current ones. Ultimately, this implementation is an attempt to bridge the gap between machine learning theory and telecommunication systems, in ambition to turn traditional reactive management into intelligent, proactive network control.

4 Evaluation of Forecasting Methods

This section evaluates candidate forecasting algorithms to identify the most suitable predictive engine to be implemented within the 5G Network Data Analytics Function (NWDAF) prototype. Through a systematic sequence of benchmarks, we compare classical statistical modeling against modern deep learning architectures, analyzing their predictive accuracy, computational resource footprints, and practical deployment viability. The investigation progresses from high-frequency single-series stress tests comparing SARIMA and DeepAR, to an extensive evaluation of continual rolling-window adaptation versus classical training under zero-shot testing conditions. Furthermore, physical resource profiling and an emulated live streaming deployment are conducted to measure inference latency, memory overhead, and automated self-recovery under real-world demand which will be demonstrated in section 4.4 To establish a good foundation for these comparative experiments, the raw cellular traffic traces must first be compressed, filtered, and temporally aligned, as detailed in the dataset preparation below, section 4.1.

4.1. Dataset preparation

The Dataset that has been used is the Netmob2023 dataset with a size of 2.5 Tb of traffic spoofing. It contains the cellular traffic of France (only Internet traffic), recorded over 20 major cities to create this dataset on a period of 77 days. The data is divided into 15-minute time intervals, recognizing 68 services, while each cell has a size of 100x100 m [17].

Given the size of the dataset, it was not feasible to use it for training the algorithms. Therefore, two compression techniques have been implemented taking into consideration the necessary details level and the features that can be useful for forecasting algorithms for pattern recognition process (seasonality, trend, etc.).

The first compression technique focused on the total traffic of each city combining all the cells into a single cell representing the city traffic. It has been done for each city, divided to upload and download traffic, keeping the 68 services recognizable. After applying this strategy, the features that was neglected were the geolocation of the cells this made it possible to reduce the size of the dataset from 2.5TB to 120MB. A representation of the data, divided by services for the whole country, can be seen in Figure 17, while the France national traffic can be seen in Figure 18 and Figure 19.

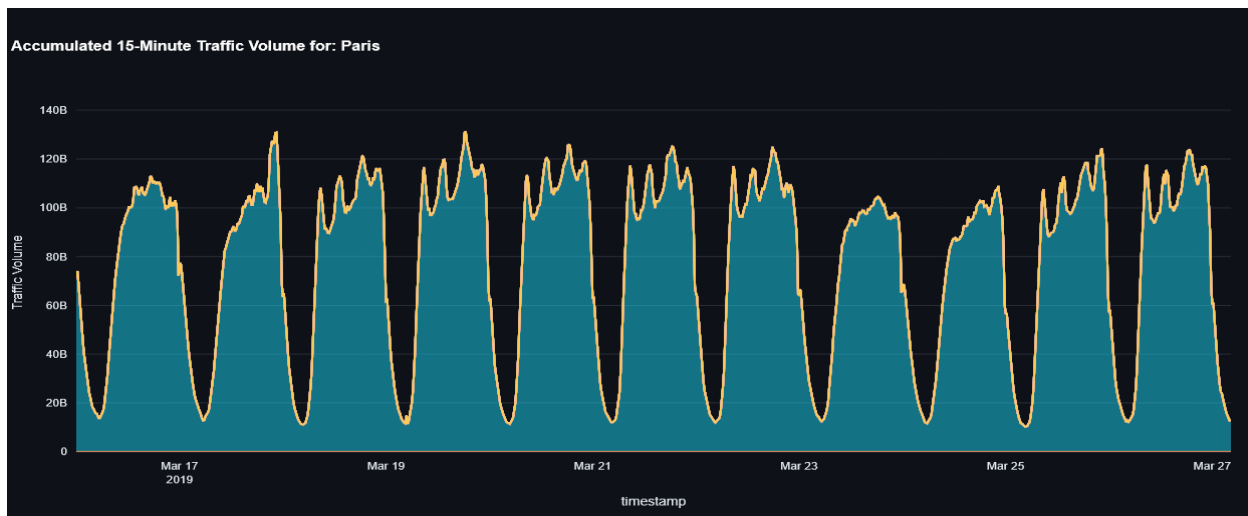


Figure 19: total traffic of a specific city, in this example Paris

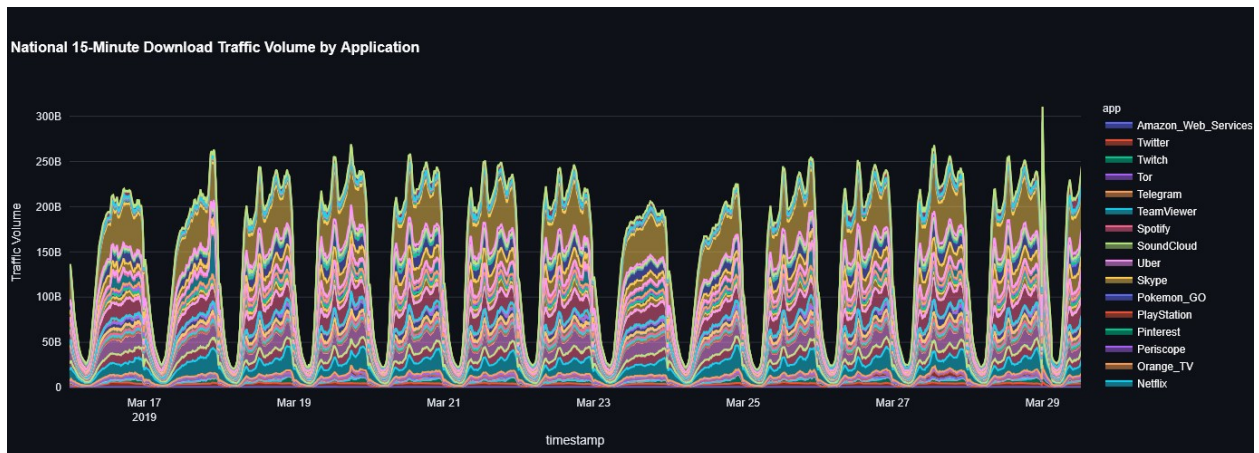


Figure 18: France total Download traffic

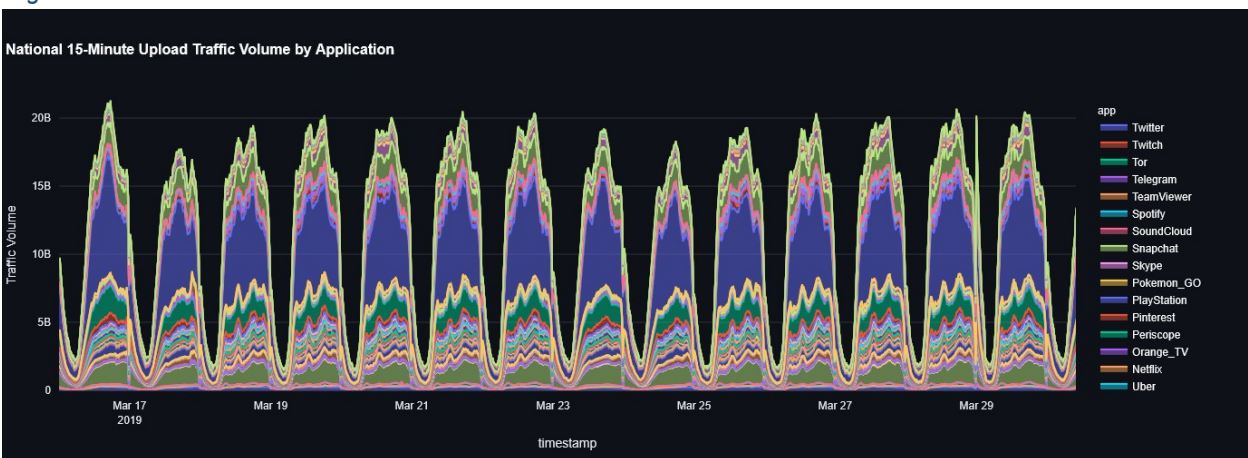


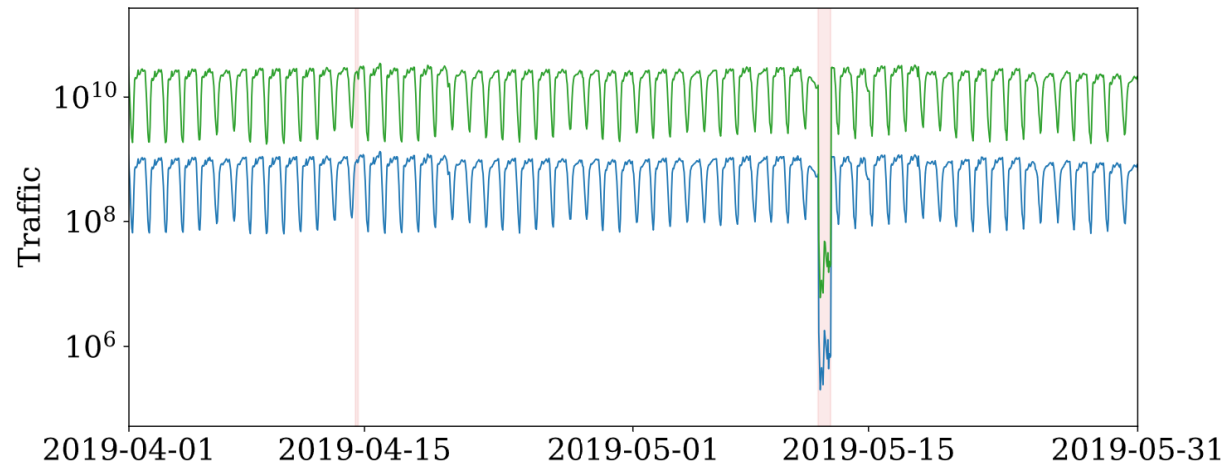
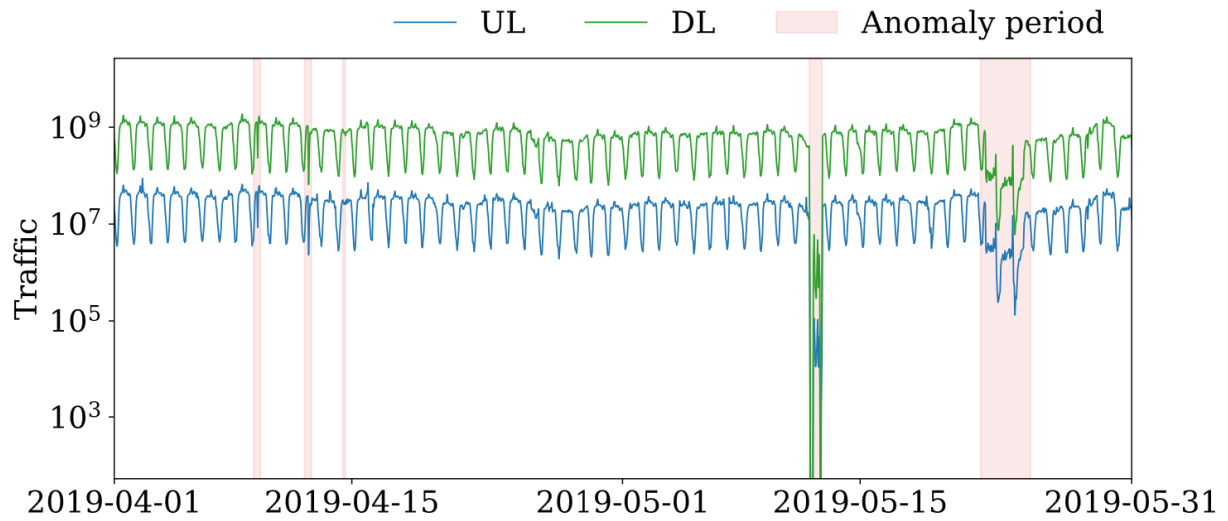
Figure 17: France total Upload traffic

Figure 17 through Figure 19 illustrate the traffic dynamics of the telemetry dataset across different spatial aggregations, transmission directions, and application profiles. While Figure 17 highlights the localized diurnal fluctuations of an individual metropolitan center (Paris), Figures 18 and 19 expand the scope to a nationwide level by decomposing France's cumulative download and upload volumes across diverse digital services. In these stacked distributions, the traffic is disaggregated into distinct application categories, including high-throughput video and content streaming (such as Netflix, Twitch, and Orange TV), enterprise cloud utilities (Amazon Web Services, TeamViewer), social networking and messaging (Twitter, Telegram, Skype), and mobile or gaming platforms (PlayStation, Pokémon GO, Uber). Together, these visualizations capture the strong periodic seasonality of network demand as well as the asymmetric bandwidth footprints inherent to different services, establishing a realistic multi-service baseline for testing the forecasting models.

Note: In accordance with the NetMob2023 dataset specifications, traffic volumes are reported in obfuscated units to ensure user privacy preservation. Since the relative distribution and trends remain intact, this scaling has no impact on model training, algorithm performance, or forecasting accuracy.

Because the NetMob23 dataset is gathered directly from a live commercial cellular network, it is characterized by anomalous events of technical nature, caused, e.g., by exceptional surges in demands, radio access network malfunctioning, network configuration errors, energy supply problems, or issues in the traffic monitoring system. Examples of anomalous events can be observed in the NetMob23 dataset in Figure 20 and Figure 21, for different combinations of cities and services. The plots show the overall traffic in UL and DL, with anomalies evidenced by light red areas. The three plots show for instance the effects of a nationwide network outage on May 12, 2019, which caused both UL and DL traffic to drop substantially in the whole network.

Figure 20 and Figure 21 are examples of applications traffic to demonstrate the anomalies. The full list of major anomalies detected can be found in the source [17].



The presence of anomalies can alter and affect the training of the models, so to overcome such an obstacle we decided to fill those anomalies with the average traffic of the other days to create smooth normal traffic. To ensure continuous and uniform temporal sequences required by deep learning architectures and autoregressive models, the identified anomalies and structural irregularities were handled through targeted preprocessing and seasonal imputation routines.

Daylight Saving Time (DST): Because the raw cellular traces follow local wall-clock time, the transition to Central European Summer Time (CEST) on March 31, 2019, resulted in a 23-hour day containing only 92 fifteen-minute observation slots (t_0 to t_{91}) instead of the standard 96 slots ($T = 96$). To prevent sequence truncation and maintain a uniform dimensionality across all time steps, the missing late-evening slots ($t_k \in \{t_{92}, t_{93}, t_{94}, t_{95}\}$, corresponding to 23:00–23:45) were reconstructed using a bidirectional seasonal mean imputation.

Given the strong weekly seasonality exhibited by mobile cellular traffic, the imputed value $\hat{x}_{c,a,d}(t_k, D_{\text{DST}})$ for city c , application a , and link direction $d \in \{\text{UL}, \text{DL}\}$ on target date D_{DST} was computed as the arithmetic mean of the corresponding slots from the preceding healthy Sunday ($D_{\text{DST}} - 7\text{d}$, March 24) and succeeding healthy Sunday ($D_{\text{DST}} + 7\text{d}$, April 7):

$$\hat{x}_{c,a,d}(t_k, D_{\text{DST}}) = \left\lfloor \frac{x_{c,a,d}(t_k, D_{\text{DST}} - 7) + x_{c,a,d}(t_k, D_{\text{DST}} + 7)}{2} \right\rfloor, \quad \forall k \in [92, 95]$$

This approach preserves the intra-day diurnal profile and volume scale of weekend cellular demand without introducing artificial zero-inflation or discontinuous step boundaries into the training sequences.

Multi-Cell Traffic Recovery and Aggregation for Outage Windows: For the nationwide outage and monitoring anomalies recorded on May 12, 2019, incomplete or corrupted records in the pre-aggregated files were purged and re-extracted directly from the raw base-station-level telemetry logs. A fault-tolerant, parallelized parsing pipeline aggregated individual spatial cell vectors into macroscopic city-level traffic matrices for each application and traffic direction:

$$X_{c,a,d}(D_{\text{target}}) = \sum_{m \in \mathcal{M}_c} \mathbf{x}_{m,a,d}(D_{\text{target}})$$

where $\mathcal{M}_c$ denotes the set of all base stations within the metropolitan area of city c . Zero-padding and vector realignment were enforced up to the boundary $T = 96$ bins per day, ensuring that physical network drops are preserved as genuine low-traffic observations rather than structural null values or malformed time-series steps.

Maintaining contiguous temporal alignment without sequence breaks is essential for maintaining consistent sliding window indices across multi-step forecasting models (such as SARIMA, DeepAR, and Temporal Fusion Transformers). By isolating calendar artifacts (such as DST shifts) through seasonal interpolation and resolving parsing faults via raw-file re-aggregation, the pipeline preserves true network dynamics while preventing gradient anomalies caused by missing timestamps.

4.2. Experiment 1: High-Frequency Effect

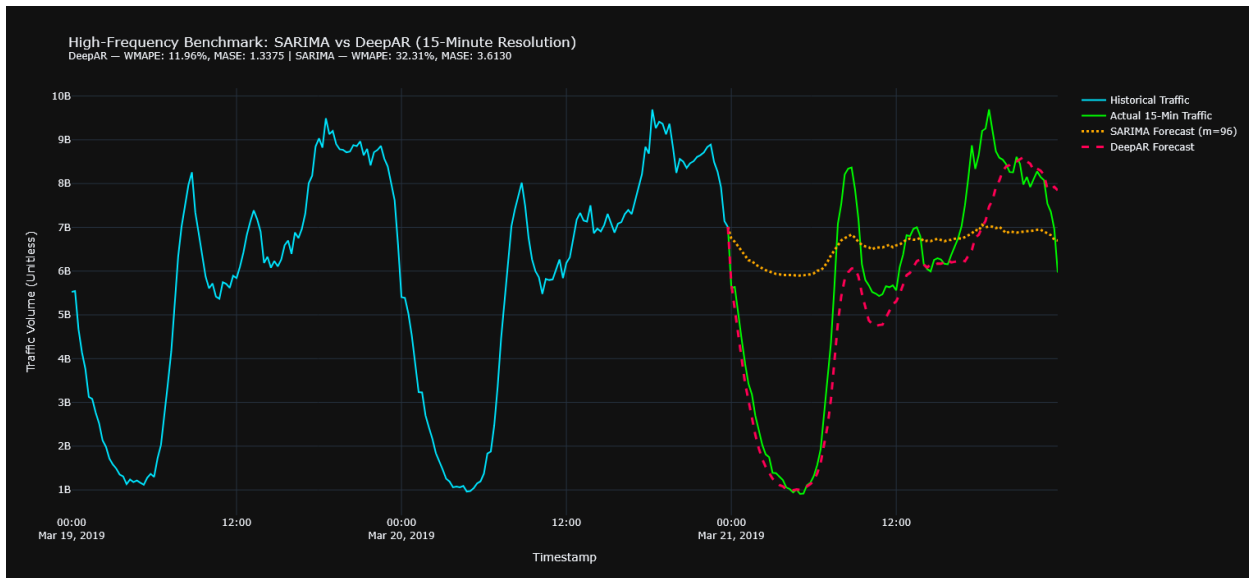


Figure 22: Experiment 1 plot

An experiment has been conducted to push SARIMA algorithm to its limit by testing it against DeepAR as shown in Figure 22, in an attempt to prove and highlight the mentioned limitations in section 2.7. The model is trained on a high-frequency (15-minute resolution, 96 daily slots) slice of the NetMob 2023 dataset—isolating 5 days of training observations (480 steps) to forecast a full 24-hour horizon (96 time steps) of Paris YouTube downlink traffic that has been compressed in Section 4.1.

We can notice from this extreme experiment over a single day 96 reading, that SARIMA fails to follow and learn because of the high frequency of data that has been recorded, even DeepAR that is much more data greedy it still preforms better than SARIMA in this specific experiment not mentioning the high computational cost of finding the right combination of SARIMA parameters

For testing the accuracy and how much benefit the new algorithm provides, few tests have been conducted. Also, there are few training and testing strategies that has to be introduced and tested.

4.3. Experiment 2: Comparison of Learning Strategies

In this experiment, we are comparing two learning strategies for DeepAR model to understand if changing the learning strategy during the training increases the accuracy or not. To do this, two models have been trained. The first model was trained in a classic way by feeding it the whole dataset. For the second model, we used a strategy called rolling window, where the model gets trained on the first 30 days, then a window takes the next 14 days to update the model weights,

and so on until we include all the days of the dataset. For testing the model in a situation similar to real-life scenarios, zero-shot test has been used to test the models on totally new city that has never been seen before by the models.

Rolling Window: Standard machine learning often uses a single train/test split (e.g., train on 70 days, test on 7 days). However, in time-series forecasting, models degrade over time as network behaviors drift. In order to solve this problem, the Rolling-Window strategy trains the model on Days d_0 to d_{0+x} and predicts Day $x+1$ and so on till the rolling window dimension is reached. Then, when the window of new samples has reached the desired dimension, the model is updated with d_{0+x} to d_{rw+x} actual data and predict day $d_{0rw+x+1}$ continuing this "rolling" prediction across the entire test set. This proves how well the algorithm adapts to chronological drift and mimics how the algorithm would be deployed in a live 5G edge controller constantly ingesting new telemetry.

Zero-Shot Forecasting: is one of the hardest tests to which a model is to be submitted; it consists of performing forecasting on data that the model has never seen before. For neural networks, this demonstrates how quickly it can adapt its global weights to a localized node.

Two different strategies will be evaluated in a Zero-Shot fashion. Out of the 20 cities from the NetMob dataset, the one used for model evaluation will be Lyon city. The first strategy model will use *Classic Batch Training*, this means that it is trained on the entire 77-day history of the 19 cities all at once in a single static batch; the training is performed on 25 epoch. The second model will use *Rolling-Window Progressive Training*, this means that the model starts by training on an initial temporal ratio of the first 30 days of the 19 cities used for training. Then it rolls forward across the timeline in sequential windows of 15 days, continuing its training across the 19 cities up to the final day. The number of training epoch for this second model is reduced for computational timing reasons such that it is simulating a real model updating itself when receiving new data.

The resulting models from the training are then used to forecast samples of the city of Lyon. The first model is used to predict all the samples from day 31 to the last day (77). for the second model, the updated version has been used resulting in four different windows as depicted in Table 1.

Evaluation window	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0-29	30-44	14.42	15.12	-00.7	1.2081	1.2667	-00.0586
Window_2	0-44	45-59	28.34	26.31	2.04	1.5665	1.4539	0.1126
Window_3	0-59	60-74	21.21	12.6	8.61	1.8239	1.0839	0.74
Window_4	0-74	75-76	36.29	33.21	3.08	2.7812	2.5453	0.2358
Static Model	0-76	30-76	21.12	17.61	3.51	1.5859	1.3225	0.2634

Table 1: the results of experiment 2

To provide a comparison with a city on which the static model was trained, Paris was used as a reference (see Table 1). An example of forecasting is shown in the Figure 23.

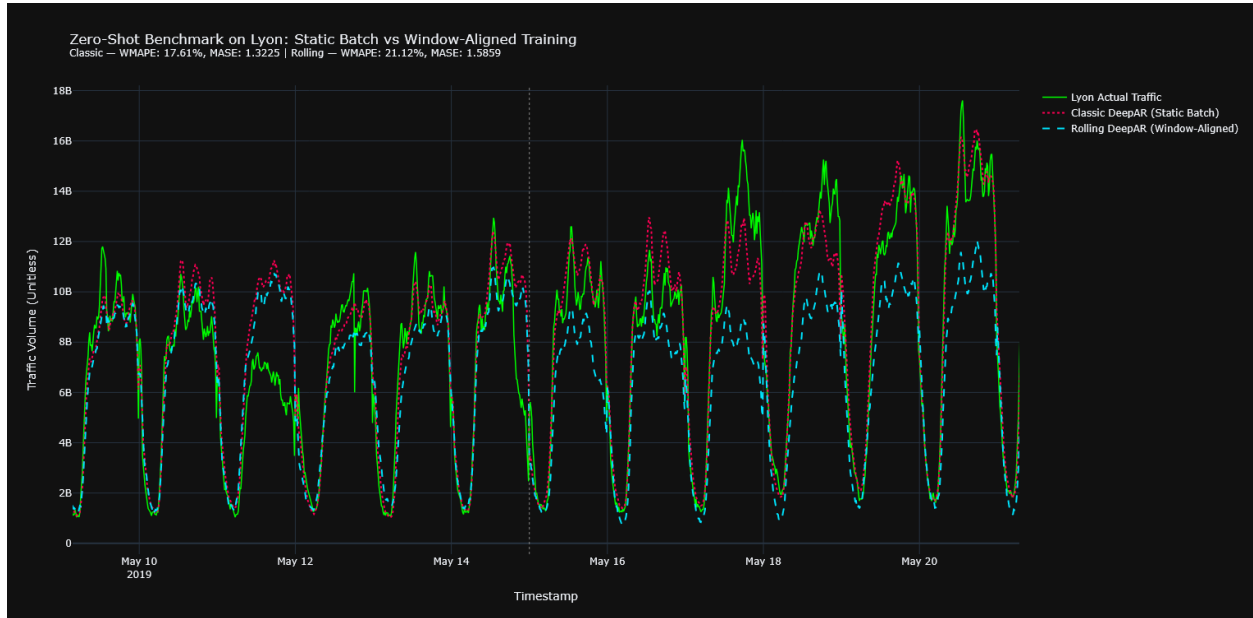


Figure 23: Experiment 2 results plot, a slice of the full plot

As Table 1 shows, the rolling window strategy didn't work as expected, the table shows in detail how the rolling window days are divided then used to train and test the model, but at the end we notice that the classic model beats the rolling strategy by 3.5% which is not negligible. Proving by that rolling window is not performing as expected. To make sure that we have given the strategy fair chances, I had to make another experiment under different conditions to make a final decision and understand the behavior of the algorithm.

In this attempt, we are going to apply the same exact strategy as before in the experiment, but the rolling window will vary during testing; this will allow us to understand what the effect of the rolling window dimension on the forecast. Also, please note that in all the following experiments we are going to use zero-shot as the evaluation test, this means that data on which the model is tested has never been seen by the models. Results are shown in following tables.

Evaluation window=5	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0-4	5-9	17.09	12.34	+4.75	1.3552	0.9784	+0.3768
Window_2	0-9	10-14	8.78	8.36	+0.42	0.8954	0.8524	+0.0430
Window_3	0-14	15-19	5.23	6.08	-0.85	0.6423	0.7468	-0.1045
Window_4	0-19	20-24	13.54	15.18	-1.64	0.7771	0.8715	-0.0943
Window_5	0-24	25-29	7.73	10.82	-3.09	0.8391	1.1739	-0.3348

Window_6	0-29	30-34	5.69	8.10	-2.41	0.7267	1.0346	-0.3079
Window_7	0-34	35-39	11.28	14.25	-2.97	0.8668	1.0952	-0.2284
Window_8	0-39	40-44	11.22	7.03	+4.19	1.0138	0.6351	+0.3787
Window_9	0-44	45-49	29.92	34.91	-4.99	1.4497	1.6913	-0.2416
Window_10	0-49	50-54	15.89	15.01	+0.88	0.9518	0.8990	+0.0528
Window_11	0-54	55-59	16.15	15.85	+0.30	0.7857	0.7712	+0.0145
Window_12	0-59	60-64	18.93	10.83	+8.10	1.4958	0.8557	+0.6401
Window_13	0-64	65-69	9.90	7.11	+2.79	1.1984	0.8606	+0.3378
Window_14	0-69	70-74	9.43	12.31	-2.88	0.7808	1.0197	-0.2389
Window_15	0-74	75-76	19.59	25.69	-6.10	1.0275	1.3474	-0.3199
Total	0-74	5-76	12.35	12.30	+0.05	0.9948	0.9910	+0.0038

Table 2: experiment result for window size of 5 days

Evaluation window=10	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0-9	10-19	9.30	7.36	+1.94	1.0354	0.8195	+0.2159
Window_2	0-19	20-29	10.23	12.91	-2.68	0.7728	0.9751	-0.2023
Window_3	0-29	30-39	8.49	10.72	-2.23	0.8282	1.0456	-0.2175
Window_4	0-39	40-49	17.63	18.25	-0.62	1.1727	1.2136	-0.0409
Window_5	0-49	50-59	16.53	15.56	+0.97	0.8823	0.8307	+0.0516
Window_6	0-59	60-69	11.16	8.48	+2.68	1.0968	0.8339	+0.2629
Window_7	0-69	70-76	12.81	16.17	-3.36	0.9229	1.1648	-0.2418
Total	0-69	10-76	11.86	12.24	-0.38	0.9568	0.9877	-0.0309

Table 3: experiment result for window size of 10 days

Evaluation window=14	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0-13	14-27	8.70	10.61	-1.91	0.7276	0.8871	-0.1596
Window_2	0-27	28-41	9.14	10.59	-1.45	0.8878	1.0288	-0.1410
Window_3	0-41	42-55	18.71	19.57	-0.86	1.0891	1.1390	-0.0499
Window_4	0-55	56-69	13.12	10.54	+2.58	1.0110	0.8121	+0.1989
Window_5	0-69	70-76	11.26	16.12	-4.86	0.8110	1.1612	-0.3502
Total	0-69	14-76	11.78	12.79	-1.01	0.9122	0.9903	-0.0781

Table 4: experiment result for window size of 14 days

Evaluation window=20	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0-19	20-39	9.98	11.85	-1.88	0.8446	1.0033	-0.1588
Window_2	0-39	40-59	17.22	17.06	+0.17	1.0395	1.0295	+0.0100
Window_3	0-59	60-76	10.98	11.55	-0.58	0.9385	0.9878	-0.0493
Total	0-59	20-76	12.37	13.24	-0.87	0.9420	1.0082	-0.0663

Table 5: experiment result for window size of 20 days

Evaluation window=25	Train Days	Test Days	Rolling WMAPE	Classic WMAPE	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
Window_1	0–24	25–49	11.98	13.46	-1.48	1.0183	1.1436	-0.1253
Window_2	0–49	50–74	11.47	11.58	-0.11	0.8752	0.8837	-0.0085
Window_3	0–74	75–76	20.37	26.89	-6.52	1.0686	1.4106	-0.3420
Total	0–74	25–76	12.07	13.12	-1.05	0.9538	1.0367	-0.0829

Table 6: experiment result for window size of 25 days

Tables 2 through 6 compare the performance of the rolling window strategy against the classic training across five different update window sizes: 5, 10, 14, 20, and 25 days. Across these tests, the rolling approach consistently delivers lower prediction errors once the update window is 10 days or longer. In Table 2, using a very short 5-day window results in almost identical total performance between the two methods, with the rolling model finishing at 12.35% WMAPE and 0.9948 MASE compared to 12.30% WMAPE and 0.9910 MASE for the classic baseline. Updating the weights every 5 days proves to be too frequent; the model overreacts to short-term traffic noise and unusual single-day anomalies—such as the large error spike in Window 9 where WMAPE reaches 29.92%, or Window 12 where the delta is +8.10%—preventing the network from settling into stable forecasting patterns.

The benefits of continual learning emerge clearly as the window widens to 10 days in Table 3 and 14 days in Table 4, where the rolling window model outperforms the classic model in almost every single testing step. The 14-day window represents the optimal configuration across the entire study, achieving the lowest overall error at 11.78% WMAPE and 0.9122 MASE, which beats the classic model by a full 1.01% in WMAPE and 0.0781 in MASE. A two-week window provides enough historical context for the model to capture weekly traffic cycles reliably while remaining flexible enough to adapt to emerging trends. When the window expands further to 20 days (Table 5) and 25 days (Table 6), the rolling model continues to outperform the static benchmark, maintaining solid negative deltas of -0.87% and -1.05% WMAPE. However, updating only every 20 or 25 days causes the model to react more slowly to recent changes in user demand, which nudges the overall rolling error back up toward 12.37% and 12.07%.

Window Size (Days)	Test (Days)	Rolling WMAPE (%)	Classic WMAPE (%)	WMAPE Delta	Rolling MASE	Classic MASE	MASE Delta
5	5–76	12.35	12.30	+0.05	0.9948	0.9910	+0.0038
10	10–76	11.86	12.24	-0.38	0.9568	0.9877	-0.0309
14	14–76	11.78	12.79	-1.01	0.9122	0.9903	-0.0781
20	20–76	12.37	13.24	-0.87	0.9420	1.0082	-0.0663
25	25–76	12.07	13.12	-1.05	0.9538	1.0367	-0.0829

Table 7: Forecasting performance of the Rolling window versus the Classic training strategy across varying window sizes.

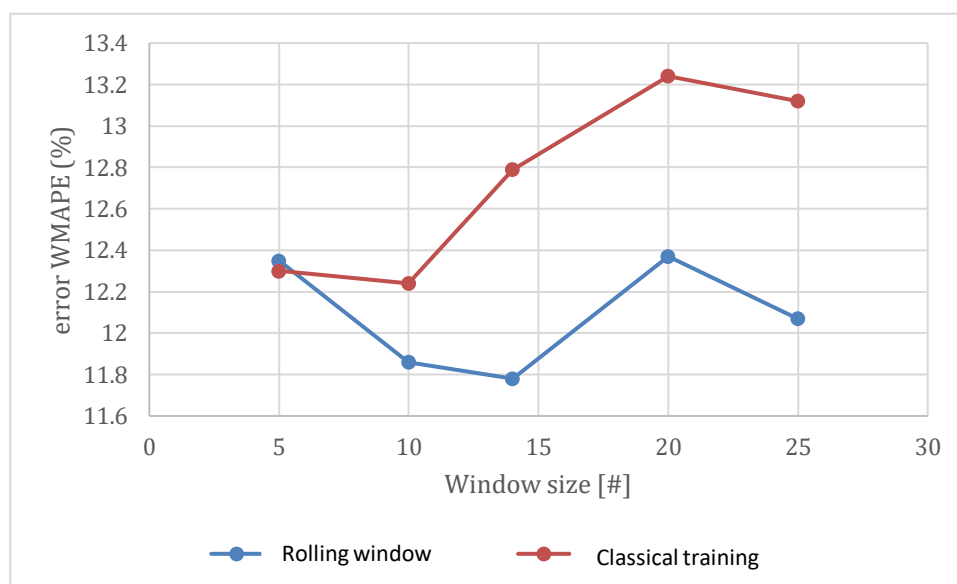


Figure 24: window size here also refers to the zero-shot testing window size of the forecast

Figure 24 visually brings together the overall results from Table 7, highlighting how both approaches behave as the evaluation window expands. At first glance, the classic model’s error (the red line) appears to worsen as the window size increases, rising from 12.24% at 10 days to 13.24% at 20 days. However, this upward trend is not caused by any change in the model itself—since the classic model remains completely frozen—but by the shifting test period. Because testing only begins after the initial training window ends, larger window configurations systematically exclude the earlier days of the dataset. For example, the 5-day setup includes the calm, low-error traffic observed between Days 5 and 24, which pulls down the classic model's overall average. In contrast, the 20-day and 25-day setups discard these easier early days entirely, forcing the classic model to be evaluated only on the later, more volatile traffic periods where forecasting is naturally more challenging.

Even as the testing ground becomes progressively harder, the rolling window strategy (the blue line) consistently maintains superior accuracy, staying well below the classic baseline across nearly every configuration. Because the rolling model continuously updates its weights with recent network data rather than relying on a static snapshot, it actively tracks ongoing trends and shifts in user behavior. The visual curve underscores why a 14-day window is the optimal operating point: while very short windows (5 days) overreact to brief anomalies and very long windows (20 to 25 days) adapt too slowly, the 14-day update strikes the right balance between learning fresh patterns and filtering out random daily noise, achieving the lowest overall error at 11.78% WMAPE and widening the performance gap over the classic benchmark to more than a full percentage point.

4.4.Experiment 3: Resource consumption

This further experiment will take into consideration the computational complexity of the used forecasting techniques. An algorithm’s accuracy is only half the conquest; inference speed and compute cost dictate its feasibility or if it is just a theoretical solution, by tracking the memory consumption of RAM, training time, and inference latency (e.g., how many milliseconds it takes to generate a forecast). While DeepAR might beat SARIMA in accuracy, we still need to quantify the cost. Moreover, documenting exactly how these forecasting algorithms perform on the machine provides baseline metrics for what type of compute would be required to deploy this solution.

The fourth experiment will use SARIMA and DeepAR models both trained and tested on the NetMob dataset, SARIMA is evaluated on the 77 days Lyon city only, split into 70 training days and last 7 days are testing days ($7 \times 96 = 672$ steps at 15-minute granularity) exclusively for evaluation. DeepAR's training set contains 19 city series with Lyon first 70 days. Both models after being trained will execute a rolling 24-hour forecast to be evaluated ($H = 96$ steps) repeated across 7-day window. It’s worth noticing that SARIMA can't share knowledge for this reason in this specific experiment SARIMA model will be trained on a single city dataset (Lyon).

Operating System	Microsoft Windows 11 Home
Processor (CPU)	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
RAM	16 GB DDR4 (15.8 GB usable)
GPU	NVIDIA GeForce GTX 1650 (4 GB)

Table 8: the resources that have been used to conduct the excrement below

Model	Train Time (s)	RAM (MB)	Infer Latency (7 Days, s)	Latency (ms)	Latency / 24h Window (ms)	WMAPE %	MASE	RMSE (GB)
SARIMA (Lyon)	30.5	95.24	1.834	2.73	262.02	29.46	2.54	3.3
DeepAR (Global - Lyon)	132.10	741.11	0.333	0.5	47.59	21.22	1.83	2.5

Table 9: Experiment 4 comparison results between Global-DeepAR and SARIMA

Model	Train Time (s)	RAM (MB)	Infer Latency (7 Days, s)	Latency (ms)	Latency / 24h Window (ms)	WMAPE %	MASE	RMSE (GB)
SARIMA (Paris)	14.30	95.64	1.975	2.94	282.11	26.87	1.866	17.91
DeepAR (Global - Paris)	135.41	736.38	0.41	0.61	58.59	19.87	1.380	14.86

Table 10: comparison results between Global-DeepAR and SARIMA tested on last 7 days of Paris city

Model	Train Time (s)	RAM (MB)	Infer Latency (7 Days, s)	Latency (ms)	Latency / 24h Window (ms)	WMAPE %	MASE	RMSE (GB)
SARIMA (Paris)	14.01	94.92	1.816	2.70	259.48	26.87	1.866	17.91
DeepAR (Local - Paris)	129.12	736.77	0.412	0.61	58.92	14.04	0.975	11.25

Table 11: comparison results between Loca-DeepAR and SARIMA tested on last 7 days of Paris city

The experimental results evaluate the practical trade-off between a classical *SARIMA* model and *DeepAR* model for 24-hour cellular traffic forecasting across Lyon and Paris. The evaluation focuses on two critical dimensions, forecasting accuracy and efficiency.

Across all benchmarks, DeepAR consistently outperforms SARIMA in predictive precision, Lyon Benchmark (Table 6) DeepAR reduces forecast error significantly compared to SARIMA, cutting the error rate (WMAPE) from 29.46% down to 21.22%, while decreasing the Root Mean Squared Error (RMSE) from 3.3 GB to 2.5 GB. Paris Global Transfer (Table 7), DeepAR is trained on a multi-city pool and adapted to Paris, it continues to lead, achieving a WMAPE of 19.87% versus SARIMA’s 26.87%. DeepAR in Paris-Local (Table 8), delivers its best overall result when trained purely on local Paris history. It pushes the error rate down to **14.04%** and achieves a Mean Absolute Scaled Error (MASE) of **0.975**. Because an MASE below 1.0 indicates performance superior to a standard seasonal baseline, DeepAR proves uniquely capable of learning intricate daily routines that classical models miss.

Computational metrics highlight a clear separation between offline training demands and live operational deployment. SARIMA begins very quickly, completing its training phase in 14 to 30 seconds with a minimal memory footprint of approximately 95 MB RAM. In contrast, DeepAR requires ~129 to 135 seconds and around 736 to 741 MB RAM due to the structural overhead of neural network frameworks and gradient optimization. Once deployed for production, the advantage flips. Generating a full 24-hour forecast takes DeepAR only 47 to 59 milliseconds, making it 4x to 5x faster than SARIMA, which requires 260 to 282 milliseconds. DeepAR computes parallel forward passes through its network, whereas SARIMA relies on sequential step-by-step Kalman updates that accumulate delays.

While SARIMA offers a lightweight and fast setup for constrained environments, its forecasting accuracy is noticeably lower, and its live inference is slower. DeepAR requires a higher upfront investment in training time and memory, but it delivers substantially superior accuracy and ~60ms inference speeds, making it the better choice for proactive, automated network orchestration.

It is also worth noting that SARIMA cannot share knowledge, it must be trained again for each different dataset, while DeepAR can be trained once as a Global model and still can be able to forecast and achieve better results than SARIMA. To prove these allegations, an experiment has been conducted putting a single DeepAR model that has been trained once on high frequent data 15 minutes time interval in comparison with multiple SARIMA models to compare the costs and the accuracy. The benchmark evaluates 24-hour cellular network traffic forecasting using real-world data from 20 French metropolitan areas sampled at 15-minute intervals ($m = 96$ time slots per day). For every city, the earliest 63 days (6,048 observations) are allocated for training, followed by 14 consecutive rolling 24-hour evaluation days (1,344 observations per city, yielding 26,880 forecast data points in total). The evaluated Models are Non-seasonal ARIMA(1,1,1), Seasonal- ARIMA SARIMA(1,1,1) $\times$ (1,0,0)₉₆, DeepAR evaluated across 5, 15, and 30 training epochs (with 3 randomized seed repetitions each).

A fundamental theoretical difference between the two model families lies in how they handle multiple time series, Classical ARIMA/SARIMA models are inherently univariate. Their underlying autoregressive state-space equations assume a single continuous temporal trajectory and cannot jointly model multiple independent time series simultaneously. To benchmark ARIMA/SARIMA across all 20 cities, an independent per-city local fitting strategy was employed: 20 separate model instances were instantiated and trained sequentially one dedicated model per city. During testing, each city's state was dynamically rolled forward without refitting parameter. DeepAR, in contrast, is a global model. It ingests all 20-time series simultaneously into a shared recurrent neural network (LSTM). The network learns generalized temporal patterns (such as nationwide diurnal and weekday/weekend patterns) across the entire dataset while conditioning on individual series histories.

The experiments below were executed on a dedicated local Linux workstation. Table 12 reports the hardware specifications of the server used for testing.

Operating System	CachyOS Linux (Kernel 7.2.5-1-cachyos, 64-bit)
Processor (CPU)	AMD Ryzen 7 5800X (8 Cores / 16 Threads @ 3.8–4.7 GHz)
RAM	32 GiB DDR4 (31.3 GiB usable)
GPU	NVIDIA GeForce RTX 5060 Ti (Discrete)
Motherboard	Micro-Star International (MSI) MS-7C84

Table 12: The experiment resources specifications

Results in the table below Table 13 state the computation and accuracy across all runs.

Model Configuration	Training Strategy	Fit Time (s)	Inference Time (s)	Peak RAM (MB)	Peak VRAM (MB)	WMAPE (%)	MASE
ARIMA	20 Local Models	11.7 s	0.45 s	712 MB	0 MB (CPU)	80.19%	5.56
SARIMA	20 Local Models	936.4 s (~15.6 min)	3.01 s (+41.5 s updates)	767 MB	0 MB (CPU)	29.16%	2.28
DeepAR (5 Epochs)	Global Model	13.6 s	1.85 s	2,646 MB	140 MB	22.50%	1.73
DeepAR (15 Epochs)	1 Global Model	38.3 s	1.82 s	2,649 MB	140 MB	21.11%	1.63
DeepAR (30 Epochs)	1 Global Model	75.3 s	1.84 s	2,645 MB	140 MB	20.83%	1.70

Table 13: Models comparison results

Metropolitan Area	SARIMA WMAPE (%)	DeepAR (15 Epochs) WMAPE %	DeepAR (30 Epochs) WMAPE %	Relative Error Reduction
Paris	25.10%	17.22%	15.86%	−36.8%
Lyon	28.66%	18.54%	18.47%	−35.5%
Nice	23.32%	12.05%	14.80%	−48.3%
Lille	27.31%	18.24%	18.02%	−34.0%
Marseille	21.93%	18.74%	20.57%	−14.5%
Toulouse	49.30%	49.24%	48.78%	−1.1%

Table 14: Detailed error experiment results over few cities

Note: WMAPE and MASE reflect the macro-average across all 20 cities (error average across 20 cities) and 14 test windows. DeepAR results reflect the mean across 3 distinct seeds.

The Critical Role of Seasonality, Non-seasonal ARIMA(1,1,1) fails completely on cellular traffic, yielding an error rate of 80.19% (MASE = 5.56). Because mobile traffic is governed by cyclical human behavior, a model without seasonal memory lags simply drifts toward a flat mean over a 24-hour horizon. Incorporating daily seasonality ($m = 96$) in SARIMA brings the WMAPE down to 29.16%. While seasonal SARIMA achieves better accuracy, scaling it across 20 cities incurs a prohibitive computational penalty. Calculating the high-order 96-step lag polynomials across 20 independent state-space models requires over 15.6 minutes (936s) of CPU training time and an additional 41.5 seconds just to update the Kalman filters across the test horizon. While DeepAR eliminates the per-city training bottleneck. For the training Speed Even at 30 epochs, DeepAR trains in just 75.3 seconds on the GPU— $12 \times$ faster than SARIMA. At 5 epochs, it takes merely 13.6 seconds. DeepAR improves accuracy by over 8.3 percentage points compared to SARIMA (dropping WMAPE from 29.16% down to 20.83%). During inference, DeepAR generates all 26,880 forecast values in 1.84 seconds ($\sim 14,600$ predictions/second), compared to the cumulative 44.5 seconds required by SARIMA's sequential updates.

In summary, Classical SARIMA models cannot scale efficiently to multi-city network topologies because they require training independent state-space instances for each city, resulting in excessive training latency (~ 15.6 minutes) and limited accuracy (29.16% WMAPE). By leveraging global cross-city representation learning, DeepAR trains over twelve times faster (75.3 seconds), reduces forecasting error to 20.83% WMAPE, and generates thousands of real-time predictions in under two seconds.

4.5 Experiment 4: Live deployment test

This experiment simulates the functionality of real-time telecom network system (3GPP NWDAF) forecasting 24-hour cellular traffic demands across all 20 French metropolitan areas simultaneously at a 15-minute granularity ($m = 96$ time steps per day). The experimental pipeline is split into three distinct sequential phases:

- Phase 1: Offline Global Pre-Training (Days 1–56 / 8 Weeks). A base global DeepAR model is trained from scratch across all 20 cities for 15 epochs. The network learns the broad, nationwide diurnal rhythms, weekday-to-weekend curves, and cross-city traffic dynamics. Its learned parameters (weights and biases) are extracted and saved as an active checkpoint.
- Phase 2: Online Warm-Start Adaptation (Days 57–70 / 2 Weeks). To bridge the gap toward the live operational window, the model is instantiated with the Phase 1 weights rather than

random initialization. It undergoes a short adaptation phase (5 epochs), absorbing intermediate seasonal shifts without forgetting its foundational patterns.

- **Phase 3: Emulated Live Streaming & Micro-Updates** (Days 71–77 / 1 Week). The script emulates a live production network over 7 continuous days. At the start of each day, the model ingests recent history and produces a 24-hour forecast for all 20 cities simultaneously, plus a combined "France National Total". As real traffic data arrives, daily forecast errors (WMAPE) are tracked per city and nationally. At the end of each day, the model performs a 1-epoch micro-update ($lr = 5 \times 10^{-5}$) using the newly completed day. The updated weights are immediately inherited for the next day's forecast.

The entire streaming history and error telemetry are compiled into a 4-panel interactive dashboard as shown in Figure 25.



Figure 25: experement evaluation results

One of the main advantages that we can experience while implementing DeepAR is the updating feature. In this experiment, we prove that the model is able to learn and update its weights while it is running and the live streaming plot reveals four significant insights across the 7-day evaluation window (May 25–31, 2019). At the National aggregate level, the continual DeepAR model achieves an overall Cumulative WMAPE of 10.30% and an MASE of 0.815. Achieving an MASE well below 1.0 confirms that the continual adaptation framework substantially outperforms standard seasonal benchmarks across the entire nationwide topology.

On normal operational days, the model delivers exceptional accuracy, national daily error hovers between 2.96% and 10.60% (reaching its sharpest accuracy of 2.96% on Sunday, May 26). For individual metropolitan areas like Paris, the error remains tightly constrained between 4.22% and 9.76% across standard days, proving that the model accurately tracks local peak hours and nocturnal lulls. On May 30, a noticeable error spike occurs, national WMAPE jumps to 32.82% (and 29.77% in Paris).

In France, May 30, 2019, was Ascension Day (Jeudi de l'Ascension), a nationwide public holiday. Because telecom traffic abruptly plummeted to weekend-like levels on what would normally be a busy Thursday, the model initially over-forecasted daytime demand. A crucial feature, the continual micro-update mechanism, demonstrates immediate self-correction. In the very next 24-hour cycle (May 31), the model absorbs the unexpected shift and brings the national error immediately back down to 9.54% (and 9.76% in Paris), preventing persistent model drift.

Comparing Panel 2 (France) and Panel 4 (Paris) demonstrates the power of spatial pooling. Paris exhibited an error of 12.46% aggregated across 20 cities, producing a highly reliable forecast for local and national demand.

By combining global pre-training with lightweight daily micro-updates, the continual DeepAR model achieves a cumulative 10.30% WMAPE and an MASE of 0.8152 across all 20 French cities. The live streaming telemetry proves that warm-start adaptation maintains sub-10% daily precision under normal operations and immediately self-corrects after unforeseen demand shocks like public holidays, providing a robust, deployable solution for automated 5G network orchestration.

In summary, I evaluated whether deep learning could practically replace classical statistical forecasting within a 5G NWDAF architecture. After compressing the 2.5 TB NetMob23 dataset down to 120 MB by deleting the geolocation information, then repairing anomalies caused by Daylight Saving Time and network outages, I benchmarked SARIMA against DeepAR across 20 French cities at a 15-minute resolution. While SARIMA struggled to track high-frequency patterns and required over 15 minutes to fit separate per-city models, a single global DeepAR model trained in just 75 seconds, cut average forecasting error down to 20.83% WMAPE, and generated full multi-city forecasts in under two seconds. Testing different continual learning setups under zero-shot conditions showed that updating the model every 14 days hit the ideal balance between learning fresh traffic trends and filtering out short-term noise, outperforming both static batch training and shorter update cycles. Lastly, an emulated live deployment across all 20 metropolitan areas confirmed the model's operational reliability, achieving a cumulative 10.30% WMAPE,

maintaining sub-10% error on normal days, and autonomously self-correcting within 24 hours after an unexpected demand drop during the Ascension Day holiday. These findings demonstrate that a global DeepAR model using lightweight continual updates delivers the precision, low latency, and resilience needed for proactive network automation.

5 Conclusions and Future Works

As cellular networks transition toward 5G and beyond, the adoption of Network Functions Virtualization (NFV) and Service-Based Architectures (SBA) has shifted network operations from rigid hardware deployments to highly dynamic, cloud-native platforms. However, managing virtualized resources purely through reactive mechanisms inevitably leads to resource over-provisioning or delayed troubleshooting during sudden traffic surges. To guarantee strict Service Level Agreements (SLAs) and high Quality of Service (QoS), mobile networks must migrate toward proactive, closed-loop automated assurance. Standardized by 3GPP, the Network Data Analytics Function (NWDAF) serves as the core intelligence engine designed to collect operational telemetry and provide predictive insights for dynamic orchestration.

The primary objective of this thesis was to design, evaluate, and prototype valid practical traffic forecasting schemes within NWDAF architecture to enable real-time proactive network management. By benchmarking traditional statistical modeling against modern deep learning architectures across real-world, high-frequency cellular telemetry from the NetMob2023 dataset, this work investigated the operational trade-offs between predictive accuracy, inference latency, and computational overhead. Furthermore, this research established an optimal continual learning update strategy under zero-shot conditions and demonstrated practical deployment through a modular, based on Go language to design the plugin prototype implementation equipped with an automated accuracy safety gate.

In this thesis different types of forecasting algorithms have been compared to evaluate their precision and performance in Section 4. DeepAR delivers much better accuracy and saves computational costs overall, but it is a greedy algorithm that requires a lot of data. Then we have compared few training strategies to understand their effects and to know what constraints we have to take into consideration. In the beginning of Section 4 I had access to Netmob2023 dataset which was too big to be used in anything so I had to compress it then analyze and fix the anomalies in order to train the models on clean data, then I have experimented with different types of forecasting algorithms and evaluated their precision and performance in the main vision was to understand if DeepAR actually can beat ARIMA models and even if it did I had to understand if the appliance is feasible in real-life environments, I have noticed that DeepAR not only delivers much better accuracy but also saves computational costs overall, but it is a greedy algorithm that requires a lot of data for the training stage to have a decent forecasting model. I have tested the DeepAR on multiple cities to put on test and highlight the fact that a general model can accurately generate a local forecasts with competitive accuracy, proving by that section the superiority that DeepAR have over ARIMA models, and later on I started comparing few training strategies to understand their effects on the algorithm and to know what constraints I have to take into consideration. As a conclusion of the methods that have been evaluated, between the rolling window and classic training. The motivation behind looking for multiple strategies emerged when I noticed that for a plugin to work in the NWDAF live environment needs a good training strategy that can adapt to the new data without forgetting the old patterns. What helped me to arrive at a good model was understanding the effect of rolling window size on the accuracy. I did many experiments to

understand and find the best parameters to have and optimal accuracy. A small window size made the accuracy drops because the model was trying too hard to adapt to the data on the other hand a very large one made the model very slow to adapt to the changes or following the trend, by this the optimal window size was between 14 to 15 days of training data. At the end in Experiment 4: Live deployment test which presented the model that I believe is the best fit for a 5G system. The model was waiting for the first batch of training data to arrive then started performing the forecast while waiting for the new batch to arrive to retrain itself on the past and new data to have a better general knowledge and analyses overall. This code was not just evaluated theoretically but implemented as a prototype plugin in the NWDAF prototype that we presented earlier in section 3. And explained in detail the structure and the algorithm in section 5 proving by that the possibility of applying these innovative approaches in a real system.

Future works can improve some aspects that have not been investigated in depth in this thesis. For instance, the Netmob2023 dataset was 2.5 TB and under heavy compression it reached 100 MB, however this compression made us lose the geolocation of the cells. An improvement can be exploiting a new compression algorithm that maintains the geo location of the cells, to improve the accuracy. Another idea can be to split the data into three traffic areas (city, urban, rural) to emphasize their specific cyclic patterns. Furthermore, anomaly prediction could be improved through an event mapping strategy, which exploits the knowledge of past events that generated anomaly to forecast future ones. Finally, the evaluation of additional algorithms, such as Temporal Fusion Transformer (TFT) could also be worth investigating.

References

- [1] 5G system architecture
https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.03.00_60/ts_123501v150300p.pdf
- [2] <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>
- [3] M. Chiosi et al., “Network Functions Virtualization: An Introduction, Benefits, Enablers, Challenges and Call For Action,” in Proc. SDN and OpenFlow World Congress, Darmstadt, Germany. ETSI White Paper. URL:
https://portal.etsi.org/nfv/nfv_white_paper.pdf.
- [4] https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf
- [5] The NIST Definition of Cloud Computing. <https://csrc.nist.gov/pubs/sp/800/145/final>
- [6] <https://www.redhat.com/en/topics/5g-networks/evolution-to-a-5g-core>
- [7] <https://www.3gpp.org/technologies/nae-5gs-ct3>
- [8] James Durbin and Siem Jan Koopman. Time Series Analysis by State Space Methods: Second Edition. Oxford University Press, 2012. isbn: 978-0-19-964117-8.
- [9] <https://docs.oracle.com/en/industries/communications/network-data-analytics/22.0.0/nwdaf-user-guide/nwdaf-analytics1.html#GUID-BA3EA65F-0770-46F3-80B7-4EB0BACAA3A7>
- [10] <https://www.ibm.com/docs/it/aix/7.3.0?topic=concepts-remote-procedure-call>
- [11] <https://www.ibm.com/think/topics/arima-model>
- [12] <https://www.ibm.com/think/topics/cloud-computing>
- [13] R. Bolla, P. Bono, R. Bruschi, C. Lombardo, N. S. Martinelli and B. Siccardi, “An Open-Source Prototype of Network Data Analytics Function for Next-Generation 5/6G Environments,” 2023 IEEE Globecom Workshops (GC Wkshps), Kuala Lumpur, Malaysia, 2023, pp. 720-725, doi: 10.1109/GCWkshps58843.2023.10465014
- [14] <https://www.ericsson.com/en/core-network/5g-core/network-data-analytics-function>
- [15] NWDAF Prototype implementation source code: <https://github.com/s2n-cnlt/NWDAF>
- [16] <https://www.ibm.com/think/topics/forecasting>
- [17] Netmob2023 Dataset
<https://arxiv.org/abs/2305.06933>
- [18] <https://www.keysight.com/us/en/product/P8900S/loadcore-core-network-solutions.html>
- [19] <https://free5gc.org/blog/20240110/20240110/#overview>

- [20] <https://otexts.com>
- [21] <https://otexts.com/fpp3/simple-methods.html>
- [22] [https://www.researchgate.net/publication/316098318 DeepAR Probabilistic Forecasting with Autoregressive Recurrent Networks](https://www.researchgate.net/publication/316098318_DeepAR_Probabilistic_Forecasting_with_Autoregressive_Recurrent_Networks)
- [23] LSTM explanation
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>