UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Mitigating concept drifts in proactive mobile core scaling via online learning models

Abrahao Leite Ferreira

DM 21/2025

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2025

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Abrahao Leite Ferreira

Mitigating concept drifts in proactive mobile core scaling via online learning models

Dissertação/Tese submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para obtenção do Grau de Mestre/Doutor em Engenharia Elétrica na Área de Computação Aplicada.

Orientador: Profª. Dr. Glauco Estácio Gonçalves

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

Leite Ferreira, Abrahão.
Mitigating concept drifts in proactive mobile core scaling via online learning models / Abrahão Leite Ferreira. — 2025.
138 f. : il. color.

Orientador(a): Prof. Dr. Glauco Estácio Gonçalves
Coorientador(a): Prof. Dr. Cristiano Bonato Both
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2025.

1. Online learning. 2. Mobile networks. 3. Scalability of network core functions. 4. Concept drift. 5. Traffic forecasting. I. Título.

CDD 006.3

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Mitigating concept drifts in proactive mobile core scaling via online learning models

AUTOR: **ABRAHAO LEITE FERREIRA**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA APROVADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM: 28/07/2025

BANCA EXAMINADORA:

Profª. Dr. Glauco Estácio Gonçalves
(Orientador - PPGEE / UFPA)

Prof. Dr. Cristiano Bonato Both
(Coorientador - Unisinos)

Prof. Dr. Diego Lisboa Cardoso
(Avaliador Interno - PPGEE / UFPA)

Prof. Dr. Kleber Vieira Cardoso
(Avaliador Externo - COPPE / UFRJ)

VISTO:

Prof. Dr. Diego Lisboa Cardoso
(Coodernador do PPGEE/ITEC/UFPA)

Dedico este trabalho a todos que me ajudaram a chegar até aqui, com apoio, paciência e incentivo

Acknowledgements

A conclusão deste trabalho não seria possível sem o apoio, a paciência e o incentivo de muitas pessoas, às quais sou profundamente grato.

Agradeço, em primeiro lugar a Deus, fonte de toda sabedoria e consolo. Sua presença discreta, mas constante, foi fonte de força em meio às dificuldades, e consolo durante momentos de atribuições.

Aos meus professores e orientadores, que com empenho e paciência me guiaram ao longo deste percurso.

Aosaamigos que fiz durante essa jornada pelo suporte constante, pelas palavras de ânimo nos momentos difíceis e pela presença nos momentos certos. A amizade de vocês foi essencial para me manter firme durante essa jornada.

Ao LASSE (Núcleo de P&D em Telecomunicações, Automação e Eletrônica), minha gratidão pela oportunidade de desenvolver este trabalho em um ambiente fértil para o crescimento técnico e pessoal. A vivência no laboratório me proporcionou , além deconhecimentos, experiências valiosas que levarei para toda a vida.

E por fim, mas não menos importante, a minha família por todo o suporte e encorajamento a seguir a longa jornada que me trouxe a essa grande realização

"Cumpre o pequeno dever de cada momento; faz o que deves e está no que fazes."
Josemaria Escrivá

Resumo

As redes móveis passam por constantes mudanças e, a cada geração, novos casos de uso surgem para desafiar a tecnologia atual. Os casos de uso atuais do 5G e do futuro 6G exigem que a rede móvel ofereça maior confiabilidade e baixa latência, desafiando assim toda a rede e, em particular, o núcleo da rede móvel. Para atender a esses requisitos rigorosos, a tecnologia de rede central atual incorpora técnicas de escalabilidade baseadas em modelos de aprendizado de máquina, que permitem antecipar as demandas de tráfego e ajustar proativamente os recursos da rede antes que ocorra qualquer degradação na qualidade do serviço.

No entanto, mudanças nos hábitos dos usuários, aplicativos e protocolos de rede podem levar a alterações nos padrões estatísticos do tráfego móvel, um fenômeno conhecido como desvios de conceito. Consequentemente, modelos convencionais de aprendizado de máquina, treinados em dados antigos, tendem a perder precisão ao longo do tempo, o que pode comprometer a adoção de casos de uso com requisitos rigorosos de desempenho. Para superar essa limitação, este estudo avalia o uso de modelos de aprendizado online para o escalabilidade proativa de funções de rede no núcleo de redes móveis. Essa abordagem busca empregar modelos que possam se adaptar continuamente às mudanças na distribuição de tráfego, mantendo previsões consistentes mesmo em cenários sujeitos a desvios de conceito.

Como um estudo de caso, esta dissertação de mestrado discute como modelos de aprendizagem online podem ser empregados para aprimorar a escalabilidade da Função de Gerenciamento de Acesso e Mobilidade (AMF) sob diferentes tipos de desvio de conceito. Utilizando tráfego de dados real de uma operadora de telecomunicações, diversos cenários de desvio de conceito foram delineados e uma simulação abrangente de treze estratégias de aprendizagem online foi realizada em um núcleo móvel simulado. Os resultados mostram como as previsões precisas dos modelos online impactam métricas de serviço, como o número de solicitações perdidas e a taxa de ocupação da função. Além disso, o estudo aponta que nem todos os modelos online mantêm boa precisão preditiva sob desvios de conceito no tráfego, o que confirma a importância de testar diferentes modelos.

Palavras-chave: Redes móveis, Aprendizado online, Núcleo da rede, Escalabilidade de funções do núcleo da rede, Desvio de conceito, Previsão de tráfego

Abstract

Mobile networks are in constant evolution, and at each generation new use cases emerge that challenge the current technology. Current 5G and future 6G use cases require mobile networks to provide high reliability and low latency, thereby stressing the entire infrastructure, and particularly the mobile core. To meet these stringent requirements, current core network technology incorporates scalability techniques based on machine learning models, which allow the anticipation of traffic demands and the proactive adjustment of network resources before any degradation in service quality occurs.

However, changes in user habits, applications, and network protocols can alter the statistical properties of mobile traffic, a phenomenon known as concept drift. As a result, conventional machine learning models, trained on historical data, tend to lose accuracy over time, which may hinder the adoption of use cases with strict performance requirements. To address this limitation, this study evaluates the use of online learning models for proactive scaling of network functions in the mobile core. This approach seeks to employ models that can continuously adapt to changes in traffic distributions, maintaining consistent predictions even in scenarios subject to concept drift.

As a case study, this master's thesis investigates how online learning models can enhance the scalability of the Access and Mobility Management Function (AMF) under different types of concept drift. Using real traffic data from a telecommunications operator, several concept drift scenarios were defined, and a comprehensive simulation of thirteen online learning strategies was carried out in a simulated mobile core. The results show how the accurate predictions of online models impact service metrics such as the number of lost requests and the utilization level of the function. Moreover, the study highlights that not all online models maintain satisfactory predictive accuracy under traffic concept drift, which reinforces the importance of evaluating different models.

Key-words: Mobile networks, Online learning, Network core, Scalability of network core functions, Concept drift, Traffic forecasting.

List of figures

List of tables

List of abbreviations and acronyms

5GC	5G core
AMF	Access and Mobility Management Function
AMFR	Aggregated Mondrian Forests Regressor
ARF	Adaptive Random Forest Regressor
BaR	Bagging Regressor
BLR	Bayesian Linear Regression
CDRs	Call Detail Records
CDF	Cumulative Distribution Function
DIPI	Difference between Ideal and Predicted AMF Instances
DNN	Deep Neural Networks
EWA	Exponentially Weighted Average regressor
FIMT-DD	Fast Incremental Model Tree with Drift Detection
FOC	Full Occupancy Count
GRU	Gated Recurrent Unit
HAT	Hoeffding Adaptive Tree Regressor
HPA	Horizontal Pod Autoscaler
HTR	Hoeffding Tree Regressor
IT	Inference Time
KNN	K-Nearest Neighbors Regressor
LiR	Linear Regression
LOGR	Logistic Regression
LT	Learning Time
LSTM	Long Short-Term Memory

MAE	Mean Absolute Error
NLR	Number of Lost Requests
OHEM	Online Hard Example Mining
OXT	Online Extra Trees Regressor
PAR	Passive-Aggressive Regressor
PCT	Perceptron
PRA	Percentage of Served Requests
QoS	Quality of Service
RMSE	Root Mean Square Error
SGD	Stochastic Gradient Descent
SGT	Stochastic Gradient Tree Regressor
SMR	Softmax Regression
SRP	Streaming Random Patches Regressor
TPE	Tree-structured Parzen Estimator
UE	User Equipment
UFPA	Universidade Federal do Pará

Table of Contents

1 INTRODUCTION

Current mobile networks face the challenge of meeting the demands of emerging applications, such as autonomous vehicles and massive machine-type communications. Although these use cases have distinct requirements in terms of latency, reliability, and bandwidth (LI et al., 2023), they share a common impact on the network: the continuous growth in the number of devices and the heterogeneity of traffic patterns. This growth increases signaling load and control traffic, which may lead to congestion in the control plane and, consequently, to service provisioning failures. Beyond the quantitative increase in demand, mobile networks are also subject to qualitative changes in traffic behavior, driven by evolving user habits, new applications, and updated communication protocols. These changes alter the statistical properties of traffic over time, a phenomenon known as concept drift, which can compromise the predictive accuracy of models used in proactive network management. In this context, the correct dimensioning of network functions at the *5G core* (5GC) becomes not only a matter of capacity planning but also a challenge of adaptability, requiring mechanisms capable of learning and adjusting to dynamic and non-stationary environments (KURANAGE et al., 2023).

Strategies for scaling functions in the current 5G core include reactive and proactive methods. In the reactive method, new instances are created or removed according to current demand. However, the delay between decision making and actual action can cause underprovisioning, service degradation, and oscillations, that is, instances that are reconstructed soon after being removed (AKSHLLEY; CARVALHO; LOPES, 2022). The proactive method, in turn, employs predictive models, usually based on machine learning, to anticipate traffic variations (HOI et al., 2021).

A common practice in proactive methods is to capture data from the network to train the models in an offline way. However, offline learning models tend to lose accuracy over time due to concept drift — that is, changes in the statistical properties of traffic that occur as new applications and network technologies emerge (MANIAS; CHOUMAN; SHAMI, 2023). In this way, offline learning models trained for 5G networks may become less effective as the network evolves towards 6G. In contrast, online learning models are trained with live data and stand out for their ability to continuously adapt to changes in data through the incremental learning process (PÉREZ-SÁNCHEZ; FONTENLA-ROMERO; GUIJARRO-BERDIÑAS, 2018). Due to this characteristic, online learning models have recently been used to solve problems in non-stationary scenarios. An example is the work of (SILVA et al., 2024), which obtained promising results when investigating the use of online learning models for edge computing environments.

The evolution of mobile networks has driven a significant transformation in the way individuals, businesses, and institutions interact with the digital world. With the advent of 5G,

not only a new standard of connectivity emerged but also arose a profoundly redesigned, service-oriented, and highly virtualized network architecture, with promises of high availability, very low latency, and massive support for critical devices and applications.

In this scenario, the 5GC is responsible for enabling several essential functionalities, such as authentication, session management, mobility, network slicing, and *Quality of Service* (QoS) policies. These functions, now implemented in a disaggregated and virtualized manner under the *Service-Based Architecture* (SBA) paradigm, are instantiated as microservices that operate dynamically and scalably in cloud-native environments. This new approach offers benefits such as greater flexibility, interoperability, and faster service deployment, but also presents challenges, including complex orchestration of microservices, efficient scalability under variable loads, increased signaling overhead, and the need to ensure resilience and end-to-end performance.

Control plane functions are key to the efficient and reliable operation of the network. They are constantly subjected to abrupt load variations, infrastructure failures, and stringent latency requirements imposed by critical applications. In this context, it is essential to develop autonomous and adaptive mechanisms capable of proactively scaling these functions, anticipating demands, and responding quickly and efficiently to changes in the operational environment

Among these functions, the *Access and Mobility Management Function* (AMF) is regarded by the industry as the most critical component of the 5GC, as it constitutes the primary entry point for user equipment (UE) and the radio access network (RAN). Due to this central role, the AMF is particularly exposed to traffic overload and must therefore be prioritized in mechanisms for detecting and mitigating such conditions, while ensuring the integrity of registration, authentication, mobility, and session establishment procedures (DAHLMAN; PARKVALL; SKÖLD, 2024)

Failures in these functions, especially under overload or failover conditions, can compromise service continuity, directly affecting the end-user experience. As highlighted by network equipment vendors (Ericsson, 2023), even when failover mechanisms are successfully activated, complete traffic restoration can take considerable time, especially if a large number of UEs are impacted. In critical scenarios, this recovery latency is unacceptable, which reinforces the need for scalability mechanisms capable of dynamically adjusting network resources to maintain operability under large demand fluctuations.

In this context, the scalability of network functions emerges as a central strategy adopted by the industry to ensure operational robustness in the face of load variations and failures. Companies have been investing in modular, microservices-based architectures (Cisco Systems, 2024), in which scalability is promoted even within the 5GC functions. In this case, the AMF is implemented as several independent services (protocol endpoints, authentication, storage, load balancer etc) interconnected through a common network bus, which can scale independently for improving response time and availability.

1.1 Objectives

1.1.1 General Objective

The primary objective of this work is to investigate and evaluate how online learning techniques can be applied to improve the proactive scalability of 5G network core functions, with a focus on the AMF function. The proposal aims to analyze the effectiveness of various machine learning models in handling concept drifts in data traffic.

Through experimentation with real data and simulation of a scalability scenario, the aim is to identify strategies that reconcile predictive accuracy, computational efficiency, and real-time adaptation capacity. This way, this dissertation wants to contribute to the adoption of more robust scalability solutions in telecommunication networks, promoting greater stability and performance in the face of the dynamic nature of these systems.

1.1.2 Specific Objectives

- Analyze the challenges of proactively scaling functions in the 5G network core from the perspective of non-stationary traffic conditions.
- Evaluate the predictive performance of different online learning models, comparing them in terms of their ability to adapt to changes in traffic patterns, and their effect on the scalability decisions of the AMF function, highlighting those with the best balance between accuracy, computational efficiency, and operational stability.
- Simulate a scalability environment based on real traffic data in order to measure the impact of predictions on the provisioning of AMF instances, using high-level operational indicators.
- Evaluate the benefits of adopting online learning models compared to offline approaches, highlighting gains in adaptability, failure reduction, and efficiency in dynamic provisioning of the AMF function.

1.2 Outcomes

During his master's degree, the author conducted research focused on the proactive scalability of network functions in the core of mobile networks, with an emphasis on applying online learning models to non-stationary scenarios. A significant portion of the results obtained were published at national scientific conferences, which served as the basis and framework for this dissertation.

1.2.1 Contributions Related to the Dissertation Theme

The technical content of this dissertation is the result of the consolidation and deepening of two main publications:

- "Online Approach for Proactive Scaling of Mobile Core Network Functions" – presented at the Workshop de Redes 6G (W6G), held as part of the Congresso da Sociedade Brasileira de Computação (CSBC) 2025, this paper proposed an online learning-based approach for proactively scaling network functions. The evaluation was conducted using the AMF as a use case, comparing the performance of online learning models with offline approaches in the task of dynamically scaling network instances. The experiments relied directly on real traffic data, where controlled statistical changes were introduced to simulate diverse concept drift scenarios. The results demonstrated that online learning models exhibit greater adaptability and stability compared to traditional methods in telecommunications scenarios, suggesting their potential for real-world applications subject to frequent changes. This dissertation builds upon that study by incorporating simulated environments in addition to real data, expanding the set of models under evaluation, and adopting new performance metrics, which allows for a deeper and more systematic analysis of proactive scaling under concept drift.
- "Experimentation of Online Models for Scalability of 5G Network Functions" – Accepted in the XLIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT) 2025, this work involved a preliminary exploratory investigation into the use of thirteen online learning models for traffic forecasting in 5G network core functions. The models were evaluated based on metrics such as accuracy, inference time, and computational cost, with all experiments conducted in representative simulated environments. This dissertation builds upon this study, incorporating additional stages of analysis and experimentation, which results in a more comprehensive and in-depth evaluation of the proposal.

1.2.2 Other Contributions During the Master's Degree

The author also participated as a co-author of the following interdisciplinary work:

- "Network Digital Twin for Route Optimization in 5G/B5G Transport Slicing with What-If Analysis," presented at the WS22 IEEE ICC 2025 – 2nd Workshop on Data-Driven and AI-Enabled Digital Twin Networks and Applications (TwinNetApp). The paper proposes a Digital Twin (NDT)-based platform using Graph Neural Networks (GNNs) to predict the impact of routing decisions in 5G/B5G networks. The solution integrates what-if analysis and proactive decision-making, and has been validated across various topologies and failure scenarios, yielding promising results in terms of accuracy and practical applicability.

1.3 Document structure

This dissertation is organized into six chapters, presenting the work carried out in a clear and systematic manner.

- Chapter 2 addresses the fundamental concepts necessary to understand the topic, including the architecture of the 5G network core, the principles of machine learning in continuous data flow (online learning), and issues related to the phenomenon of concept drift, which directly impacts the performance of predictive models in non-stationary environments. In addition to the main concepts related to scalability in a cloud environment
- Chapter 3 reviews the literature on the main approaches already proposed for proactive scaling of 5G network functions, with a special focus on machine learning-based solutions. The limitations of existing work are also highlighted, as well as how the present research seeks to address the identified gaps.
- Chapter 4 describes the set of methodological steps adopted in the evaluations made in this dissertation, including the experimental environment, the data preparation process, the selection and evaluation of online learning models, and the development of the simulation environment used to validate the proposal.
- Chapter 5 presents and analyzes the results of the experiments. Aspects related to the predictive accuracy of the models, the computational costs involved, and the operational impact on the AMF function scheduling decisions are discussed.
- Finally, Chapter 6 summarizes the main contributions of this work, presents the limitations found, and suggests possible directions for future research.

2 BACKGROUND

This chapter presents the theoretical framework necessary to understand the contributions presented throughout this work. Initially, in Section 2.1, the main characteristics of the 5GC architecture, including its modularization into network functions and scalability, are discussed.

Next, in Section 2.3, the fundamentals of machine learning in data streaming learning (online learning models) are addressed, including their advantages, limitations, and specific characteristics when applied to environments with non-stationary characteristics. Within this context, special attention is given to the phenomenon of concept drift, its typologies, and the impacts it can have on the degradation of the model's predictive performance.

Then, in Section 2.2, the main concepts related to scalability systems in telecommunication networks are presented, with an emphasis on proactive approaches to resource allocation.

Finally, in Sections 2.4 to 2.16, this chapter introduces the different online learning models evaluated in this dissertation, highlighting their diversity and the original works that proposed them. This overview serves as a quick reference guide for the approaches considered in the study.

2.1 5G Core

Unlike its predecessor, 5G networks have abandoned monolithic systems, in which all their components were integrated into a single rigid and interdependent structure, and have adopted a more flexible and modular approach. This approach has led to the decoupling of the radio access network (RAN) from the network core, enabling the implementation of virtualized, scalable, and cost-effective solutions (GUEMDANI; PHUNG; SECCI, 2024).

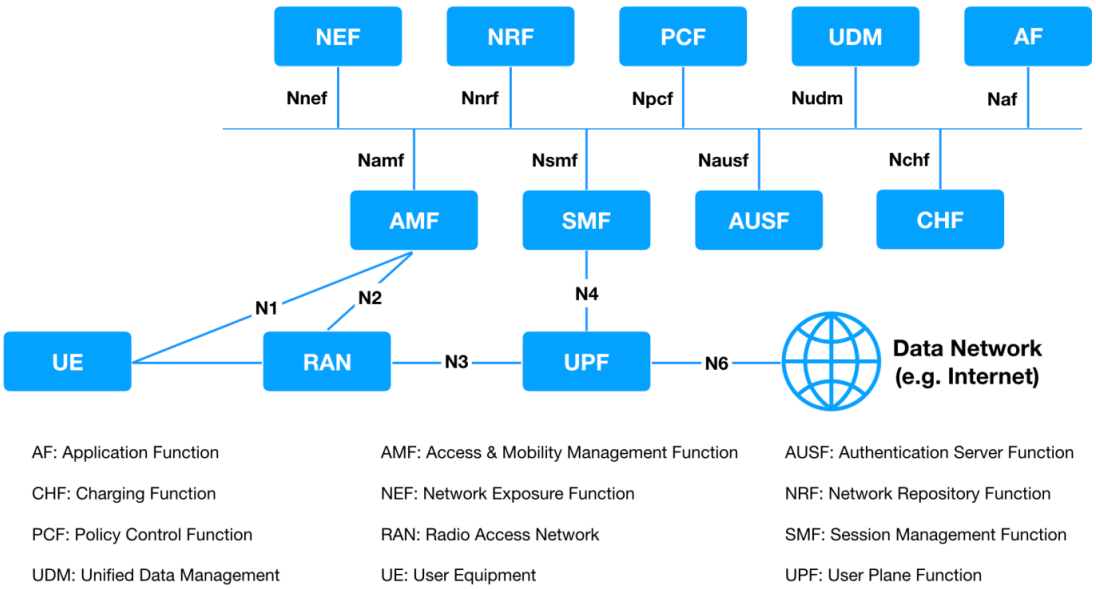
Furthermore, with the introduction of 5G, the concept of Software-Defined Networking (SDN) (NADEAU; GRAY, 2013) begins to play a fundamental role in promoting the decoupling of the control plane from the user plane (AKSHATHA; JHA; KARANDIKAR, 2018). The control plane is responsible for signaling, authentication, mobility management, and application of session policies, whereas the user plane is focused on transporting user data. This division enabled network intelligence to be implemented in an independent, centralized, and programmable manner while packet forwarding was optimized separately (BARAKABITZE et al., 2020).

With the introduction of this new architecture, the network core's functionalities began to be disaggregated into independent components, called Network Functions (NFs), which perform specific and well-defined tasks. These components can be implemented, scaled, and updated independently, which gives the network greater flexibility and agility. To enable communication between these functions, standardized Service-Based Interfaces (SBI) were specified, adopting

API-based approaches that promote interoperability and modularity (3rd Generation Partnership Project (3GPP), 2024d; 3rd Generation Partnership Project (3GPP), 2024c; 3rd Generation Partnership Project (3GPP), 2024b).

This new approach, which replaces the old point-to-point interface-based models with a service-oriented model, became known as Service-Based Architecture (SBA). Figure 1 illustrates this new logical organization.

Figure 1 – The 5G core service-based architecture



Source: (GANGWAL; GRAY, 2023)

The definitions, responsibilities, and interactions of these functions are described in the Technical Specification 3GPP TS 23.501 (3rd Generation Partnership Project (3GPP), 2024a), which establishes the reference architecture for the 5G network core. The following describes the main functions that make up the 5G Core (5GC), along with their respective assignments:

- **AF (Aplication Function):** It is the entity responsible for representing applications that want to interact with the network in order to influence service policies, such as QoS control, collection rules, and traffic routing. Among their functions are the provision of traffic descriptors to identify flows associated with specific applications, the request for differentiated service quality, the receipt of notifications about network events, and participation in the negotiation of background data transfers, thereby optimizing the use of network resources during periods of lower demand.
- **AMF (Access and Mobility Management Function):** It is responsible for managing connections and mobility of devices on the network, as well as performing the record-keeping and authentication of devices. It ensures user equipment mobility, manages the handover (connection transfer) between different cells and networks, and ensures

the continuity of the connection to the user. Additionally, AMF performs the paging process, which involves locating and activating the device as needed, for instance, for call notifications or messages. In addition it interacts with other functions, such as the Session Management Function (SMF) and the Policy Control Function (PCF), to manage the establishment and continuity of data sessions and quality of service (QoS) policies.

- **AUSF (Authentication Server Function):** Performs the device authentication process by validating the credentials provided by the UDM function. The AUSF function acts in conjunction with the Security Anchor Function (SEAF) present in the AMF to ensure that only authorized devices access network services.

- **CHF (Charging Function):** Performs the process of charging the network to the subscriber. It is also responsible for interacting with the PCF function to apply credit limits or modify policies based on user consumption, ensuring real-time financial control over network resources.

- **NEF (Network Exposure Function):** It is responsible for informing external applications about the network's capacity, allowing these applications to consult and interact with internal network features, such as mobility events, data usage, QoS policies, and session control. In addition, NEF ensures that these external interactions occur with authentication, authorization, and traceability through controlled interfaces.

- **NRF (NF Repository Function):** It is responsible for discovering and making available services offered by network functions, acting as a central point for information about active functions. It stores data such as network address, capacity, and support services, allowing any network function to dynamically find other functions with which it needs to interact.

- **PCF (Policy Control Function):** Decides which QoS, access, and billing rules should be applied to a given data flow. Its decisions are based on subscriber data, real-time information, and rules defined by the network operator. The PCF communicates these rules to the SMF and UPF, ensuring that flows are handled appropriately.

- **SMF (Session Management Function):** It handles managing user data sessions. It oversees the creation, modification, and release, as well as the allocation of IP addresses and the selection and control of user plan elements. This function also applies quality of service (QoS) policies and charging rules, interacting directly with functions such as PCF (Policy Control Function) and CHF (Charging Function).

- **UDM (Unified Data Management):** It is the function responsible for managing users' signature and profile data on the network, centralizing essential information such as authentication credentials, access policies, mobility profiles, and service permissions. It acts as an intermediary between the data stored in the UDR (Unified Data Repository) and other core functions that require this information. Among its primary responsibilities are

managing AUSF's authentication parameters, authorizing network access based on the rights of each subscriber, determining the appropriate AMF function to serve a particular device, and managing the registration and mobility of a UE.

- **UPF (User Plane Function):** It governs forwarding the data packages between the device and the external network. In addition, UPF applies QoS rules and collection policies defined by SMF and PCF and may perform deep package inspection, applying priority markings and redirections. In scenarios where devices are in inactive mode, the UPF can also temporarily store data until the UE becomes accessible again, triggering an AMF paging process.

The adoption of this modular architecture also made it possible to implement these components in containerized and cloud environments, known as Cloud-Native Network Functions (CNFs). When executed in containerized environments and managed by orchestration platforms such as Kubernetes, these functions can be scaled horizontally or vertically in an automated manner according to traffic demand and performance requirements. This ability to dynamically scale the computing resources used by NFs represents a fundamental aspect of operational efficiency. It ensures the quality of service in the 5G network by providing a resilient core to the ever-increasing network traffic.

2.2 Scalability Systems

Scalability systems are mechanisms responsible for dynamically adapting the capacity of computing resources allocated to a given application or service according to the variation in its demand. In the context of 5G networks, particularly in the network core (5GC), such systems play a crucial role in ensuring QoS and complying with Service Level Agreements (SLAs).

Scalability can be achieved in two ways: horizontal scalability (scale out/in), which consists of adding or removing instances of a service (e.g., multiple replicas of a network function), and vertical scalability (scale up/down), which involves adjusting the resources of an existing instance, such as memory or processing capacity (LORIDO-BOTRáN; MIGUEL-ALONSO; LOZANO, 2014). In most production environments, and especially in commercial cloud solutions, horizontal scalability is the most widely adopted due to its lower implementation complexity.

In cloud-native platforms such as Kubernetes, these scalability mechanisms are concretely implemented by different controllers. The most common is the Horizontal Pod Autoscaler (HPA), which automatically adjusts the number of pod replicas in a deployment based on observed resource utilization, such as CPU or memory consumption. Complementarily, the Vertical Pod Autoscaler (VPA) adapts the resource requests of containers, reallocating CPU and memory to existing pods according to demand. Other mechanisms extend these approaches: the Cluster

Proportional Autoscaler scales workloads in proportion to the cluster size, while the Kubernetes Event-Driven Autoscaler (KEDA) enables scaling based on external events, such as the number of pending messages in a queue. Additionally, scheduled autoscaling allows scaling actions to follow predefined time-based rules, which can be useful to reduce resource consumption during predictable low-demand periods (The Kubernetes Authors, 2025).

The scheduling process in auto-scaling systems typically follows a control cycle in which the system continuously monitors relevant performance metrics (such as CPU utilization, response time, and request arrival rate), analyzes the collected data to assess the current state and predict load trends, plans the necessary scaling actions, and executes resource allocation or release operations. However, this process is subject to three critical challenges that can negatively impact system performance (SINGH et al., 2019):

- Under-provisioning: The application lacks sufficient resources to handle all requests, which can occur due to flash crowd events or inefficient auto-scaling algorithms. This behavior can lead to Service Level Agreement (SLA) violations and penalties for service providers.
- Over-provisioning: The opposite of under-provisioning, where resources are allocated in excess to meet demand. While this reduces SLA violations and improves reliability, it affects the profitability of the service provider as costs are higher than necessary.
- Oscillation: Refers to the problem of rapid and repetitive scaling of resources without adequate consideration of the impact on application performance. This behavior can be caused by improperly setting static or dynamic threshold values, resulting in the constant addition and removal of resources.

Decision techniques in scalability systems can be grouped into two main paradigms: reactive and proactive approaches. Reactive techniques respond to variations in detected loads at the time of execution, usually based on pre-configured limits. Proactive techniques aim to anticipate changes in demand through predictive models, such as time series or machine learning, enabling more efficient resource allocation with lower latency (ALHARTHI et al., 2024).

In this sense, proactive approaches face additional challenges when traffic patterns are affected by concept drift. Predictive models trained on historical data may become obsolete as the statistical properties of traffic evolve, for example, due to the emergence of new applications, protocols, or usage behaviors. In such cases, the accuracy of forecasts deteriorates over time, which can compromise the efficiency of resource allocation and lead to under- or over-provisioning. Therefore, a key problem in proactive scalability systems is the development of prediction mechanisms capable of continuously adapting to non-stationary environments, where demand varies rapidly and dynamically.

2.3 Online Models

5G networks drive the adoption of increasingly intelligent and automated solutions for managing and orchestrating their infrastructures. In this sense, machine learning techniques have been widely employed, enabling Network Service Providers (NSPs) to anticipate changes in network behavior and respond proactively to failures, performance variations, or changes in demand. However, communication networks have a non-stationary nature, which means that the statistical distributions of data can change over time (GAO, 2022). In these environments, models trained under the assumption of stationarity inevitably become outdated, leading to incorrect assumptions that can even result in critical failures (DITZLER et al., 2015).

To mitigate the challenges posed by data evolution, particularly in continuous streaming environments, new computational paradigms have been developed to enable models to adapt to these changes dynamically. In this context, online learning emerges as a promising approach, as it allows models to be continuously updated as new data becomes available (YANG; GU; WU, 2019). Below, we will explore the main concepts associated with this technique and its role in addressing non-stationarity in modern communication networks.

A data stream is an infinite sequence of instances generated by a process and delivered to a system in a continuous and time-ordered manner. Each instance is accessed individually upon arrival without any prior knowledge of the total amount of data to be processed (READ; ŽLIOBAITė, 2023). In this sense, data such as that from modern telecommunications, including data generated by mobile devices, sensors, and emerging applications, are naturally characterized within the data streaming paradigm. In these systems, each connection, handover, or authentication request must be processed in near real-time to guarantee the rigorous requirements of new use cases in mobile networks.

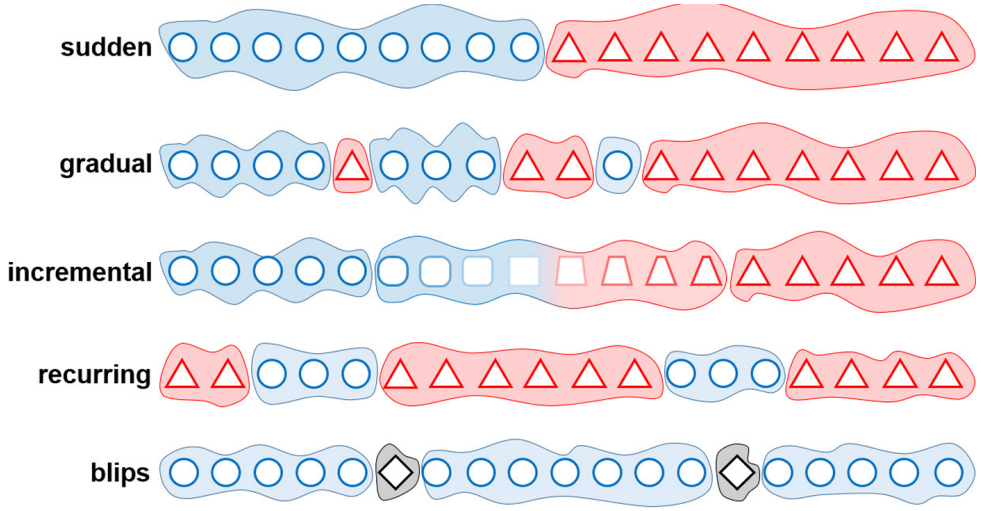
These data are not only massive but also highly dynamic, reflecting the constant evolution of the network in response to new use cases. As new technologies are incorporated into the system, the statistical distribution of the data tends to shift. As a result, the patterns previously learned by machine learning models may become outdated. In such scenarios, traditional methods that rely on training with historical data become impractical—not only due to the challenge of storing large volumes of data but also because older data quickly lose relevance over time (BARTZ; BARTZ-BEIELSTEIN, 2024).

Concept drift is characterized by changes in the previously established relationship between input data and expected results. In statistical terms, it occurs when the joint distribution of data changes over time, i.e., $P_t(X, Y) \neq P_{t+\Delta}(X, Y)$,where X represents the input variables (e.g., features of the data), Y the target variable, t the current time, and Δ a future interval. In other words, concept drift occurs when the statistical relationship between inputs and outputs changes over time. According to (MANIAS; CHOUMAN; SHAMI, 2023), concept drift is one of the biggest challenges in applying machine learning to dynamic networks, especially

in contexts such as 5G networks, where frequent changes in traffic patterns and user behavior are common. These changes not only degrade model performance but also compromise critical decisions, resulting in profound implications for the reliability and quality of service.

This phenomenon can manifest in various ways, such as abrupt, gradual, or recurring drift. According to (KRAWCZYK; CANO, 2018), concept drift can be categorized in terms of speed into different types. Figure 2 illustrates this classification, which includes sudden, gradual, incremental, and recurring drifts. Each of these types will be described in more detail below.

Figure 2 – Types of concept drifts in terms of speed



Source: (KRAWCZYK; CANO, 2018)

- **Sudden:** This phenomenon occurs when there is an abrupt change in the concept or behavior of a system at a specific point in time. In this type of change, no gradual transition between the old and new behavior is observed—the shift is direct and immediate. This form of drift is generally easier to detect compared to others, since its impact on system accuracy is instantaneous, making detection more straightforward and faster.

- **Gradual:** In this type of drift, changes occur gradually over time, with the new and old concepts overlapping. During this interval, both distributions can coexist so that, over time, the new concept becomes stable. This type of drift is more challenging to detect as changes do not cause abrupt falls in model performance and can be confused with noise if not properly monitored.

- **Incremental:** It is a specific type of gradual drift, in which the concept changes in continuous small steps over time, without necessarily having a clear overlap between the previous and new concepts. In this case, changes in distributions occur subtly, being imperceptible at short time intervals but significant along longer intervals.

- **Recurring:** It occurs when a concept that had already been observed previously reappears after some time. This phenomenon can occur cyclically, such as in seasons or periods of the year, or irregularly, as in unpredictable events that recur.
- **Blips:** Although often depicted together with types of concept drift, blips do not represent an actual drift. Instead, they correspond to isolated or short-lived anomalies in the data stream, which may temporarily affect model performance but do not indicate a lasting change in the underlying concept. In other words, blips are closer to outliers than to true drifts, and distinguishing them is important to avoid false alarms in adaptive learning systems.

To address these issues, it is essential to adopt approaches that can learn incrementally and adaptively, allowing models to adjust to changes in the environment in real time, such as online learning. Online learning can be seen as a class of methods within the field of machine learning in which the model is updated sequentially by processing data instances as they are received. This approach aims to continually maximize the accuracy of predictions based on knowledge acquired in previous interactions without the need to reprocess the entire data history. Unlike traditional batch learning, which requires the availability of the entire data set before training, online learning operates in scenarios where data is made available continuously and incrementally. (HOI et al., 2021).

2.4 Aggregated Mondrian Forests Regressor

Aggregated Mondrian Forests Regressor (AMFR) (MOURTADA; GAïFFAS; SCORNET, 2021) is an online supervised learning model that combines the principles of random forests with the hierarchical structure of partitions induced by the Mondrian process. Designed for scenarios where data is received sequentially, AMFR builds an ensemble of decision trees that are dynamically updated as new samples are observed.

The model stands out for being adaptive, utilizing an aggregation mechanism that encompasses all possible tree prunings to make predictions. This aggregation is performed based on an exponential weighting scheme based on the Context Tree Weighting algorithm. The use of the Mondrian process, a stochastic process that defines progressively refined partitions of the input space, enables the model to handle complex and dynamic structures in the data while maintaining statistical consistency over time.

2.4.1 The Mondrian Process

The Mondrian process (ROY; TEH, 2008) is a stochastic process that generates recursive partitions of the input space through orthogonal cuts, which a binary tree can represent. It is an

incremental and random construction, which allows the space to be divided adaptively without the need for complete reconstruction.

Given a budget $\lambda > 0$, which acts as an upper limit for the duration of the process, the procedure follows a sampling mechanism based on exponential distributions. At each step, a time $E \sim \text{Exp}(\cdot)$ is randomly selected; if the accumulated time exceeds λ, the process is terminated in that region. Otherwise, a random cut is performed, and the process continues recursively in each subregion, with residual budget $\lambda' = \lambda - E$.

In the one-dimensional case, the cuts are randomly positioned along the interval $[0, 1]$. In multiple dimensions, the sum of the lengths of the sides of the region is used as the time draw rate, and the choice of the cut dimension is proportional to the size of the region on each axis. This characteristic makes the process more likely to divide large regions, resulting in more balanced partitions.

A fundamental property of the Mondrian process is consistency under constraint: by restricting a partition generated with budget λ to a subregion of space, one obtains, in distribution, a partition that is statistically equivalent to the one that would be produced by applying the process directly to that subregion with the same budget. This characteristic enables local and incremental updates to existing tree structures.

The parameter λ, in turn, controls the complexity of the partition: larger values allow more splits before the process is terminated, leading to deeper trees and finer partitions. In the context of AMFR, λ can be kept fixed over time, regardless of the number of observed samples, which eliminates the need for dynamic adjustments and simplifies its practical application.

2.4.2 Tree Representation and Update in AMF

The Mondrian process can be interpreted as the incremental and stochastic construction of a decision tree. In this structure, each internal node represents an orthogonal cut in the input space, and each leaf corresponds to an indivisible region of the partition. Predictions within each leaf are performed either constantly or through some form of aggregation based on local observations.

The partitions generated by this process have a hierarchical structure naturally compatible with binary tree representations. This representation is fundamental to the functioning of AMFR, which builds and updates a forest of trees based on these partitions.

Initially, each tree in the forest is instantiated with a partition generated by a Mondrian process with budget λ. As new samples (x_t, y_t) are observed, each tree is updated independently, following these steps:

- The leaf containing the point x_t is identified.
- If it is the first sample in that region, only the statistics are updated.

- Otherwise, the Mondrian process is extended, based on the consistency property, to separate x_t from the previous point, resulting in new cuts and the creation of new nodes.
- The statistics associated with the nodes (such as counts or means) are updated along the path from the leaf to the root.

This mechanism enables trees to be expanded locally without requiring a complete rebuild. The consistency of the Mondrian process ensures that the resulting tree is statistically equivalent to the one that would have been obtained if the process had been applied to the complete data set from the beginning.

2.4.3 Prediction Function

For each tree in the forest, the prediction is performed based on aggregation over all possible prunings of the decision tree associated with the partition generated by the Mondrian process. For each pruning (finite subtree), a local prediction $\hat{y}_{\mathcal{T},t}(x)$ is computed, usually represented by the average of the values y observed at the leaves containing the point x.

The total prediction of a tree at time t is a weighted average over these prunings, using an exponential weighting scheme:

$$\hat{f}_t(x) = \frac{\sum_{\mathcal{T} \subseteq \mathcal{T}_\Pi} \pi(\mathcal{T})\, e^{-\eta L_{t-1}(\mathcal{T})}\, \hat{y}_{\mathcal{T},t}(x)}{\sum_{\mathcal{T} \subseteq \mathcal{T}_\Pi} \pi(\mathcal{T})\, e^{-\eta L_{t-1}(\mathcal{T})}},$$

where:

- $\mathcal{T}_\Pi$ is the complete tree generated by the Mondrian partition up to time t,
- $\pi(\mathcal{T})$ is a prior over subtrees (usually proportional to $2^{-|\mathcal{T}|}$, favoring simpler structures),
- $L_{t-1}(\mathcal{T})$ is the cumulative loss of the subtree $\mathcal{T}$ up to $t-1$,
- $\eta > 0$ is the learning rate used in the exponential weights.

The final prediction of the AMFR model is obtained by the simple average of the predictions made by all trees in the forest:

$$\hat{f}_t^{\text{AMF}}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_t^{(m)}(x),$$

where M is the number of trees and $\hat{f}_t^{(m)}(x)$ represents the prediction made by the m-th tree at time t.

2.4.4 Forest Structure and Diversity among Trees

AMFR consists of an ensemble of M decision trees constructed from partitions generated independently by Mondrian processes. Each tree is initialized with a distinct random seed, which ensures structural diversity among them, even when exposed to the same sequence of input data.

The trees do not share information during training; each one maintains its history of samples and their respective local updates. This independence is essential for the ensemble to benefit from variations among the models, which improves the robustness and generalization capacity of AMFR, analogous to what occurs in traditional Random Forests.

Since the final prediction mechanism is an average of the predictions from the trees, the diversity introduced by this independence helps reduce the variance of the model.

2.4.5 Computational Complexity

The AMFR algorithm is developed to operate with a computational complexity that remains controlled as the number of samples observed increases.

With each new sample incorporated into the model, only the root path to the sheet containing this new point is updated in each tree in the set. As the trees generated by the Mondrian process tend to be balanced, the average depth of a tree grows logarithmically with the total number of samples already seen. Thus, the cost of updating an individual tree is proportional to $MathcalO(\ logt)$, where t represents the number of samples processed so far.

The prediction process follows a similar logic: it consists of the efficient aggregation of predictions along different tree pruning, using a collapsed approach inspired by the context tree weighting algorithm This recursive and optimized implementation allows the forecast on each tree to be performed at $MathcalO(\ logt)$.

Considering that the model uses a set with m independent trees, both update and prediction have a total cost $MathcalO(m\ logt)$. This feature makes AMFR especially suitable for large-scale online learning applications where computational efficiency is essential.

2.5 Adaptive Random Forest Regressor

Adaptive Random Forest Regressor (ARF) (GOMES et al., 2018) is an online supervised learning model designed for regression tasks on continuous data streams, in which instances are processed in real-time and cannot be stored for reprocessing. Its architecture consists of an ensemble of incremental regression trees; each trained independently from sequentially observed instances. Diversity among the trees is promoted by two main mechanisms: stochastic sampling of instances, with weights generated by a Poisson distribution, and random selection of attribute subsets at each node split.

To deal with concept drift over time, ARF incorporates change detection mechanisms at two levels: the internal level, which monitors the local statistical stability of each tree and allows selective reinitialization of subtrees, and the external level, which evaluates the global performance of each tree in the ensemble, allowing its replacement when necessary. The final prediction of the model is obtained by taking the arithmetic mean of the individual outputs of the trees, a strategy that helps reduce variance and makes the ensemble more robust to noise and specific data variations.

2.5.1 Model Architecture

ARF consists of a set $T = \{h_1, h_2, \ldots, h_n\}$ of n base regressors, each h_i being an incremental regression tree. These trees are induced according to an adapted version of the *Fast Incremental Model Tree with Drift Detection* (FIMT-DD) algorithm (IKONOMOVSKA; GAMA; DŽEROSKI, 2011), which updates its structure as new instances are observed without the need to store previous data. Each leaf maintains a linear regression model adjusted by least squares, and splits are performed when Hoeffding's inequality indicates sufficient statistical evidence based on variance reduction.

In the context of ARF, each tree receives instances weighted according to a Poisson distribution with parameter $\lambda > 1$, applied independently, which promotes variation between the training sets. Furthermore, for each split, only a random subset of features of size $m \ll M$ is considered, which reduces computational complexity and introduces structural diversity into the ensemble.

Concept change detectors control ensemble updating. Internally, each tree monitors its prediction variance and can reinitialize individual branches in response to local drift signals. Externally, an independent detector monitors the overall performance of the tree. When a change is suspected (*alert* state), training of a replacement tree is started in parallel. If the change is confirmed, the original tree is replaced by the new one, keeping the ensemble up to date without reprocessing historical data.

2.5.2 Prediction Function and Aggregation Strategy

The ARF prediction for an input instance $\mathbf{x}_t \in \mathbb{R}^M$ is obtained by averaging the individual predictions of the n trees that compose the ensemble. Let $h_i(\mathbf{x}_t)$ be the prediction of the i-th tree for the instance $\mathbf{x}_t$, then the aggregate prediction $\hat{y}_t$ is given by:

$$\hat{y}_t = \frac{1}{n} \sum_{i=1}^{n} h_i(\mathbf{x}_t)$$

This simple aggregation approach, based on arithmetic mean, favors the reduction of model variance and presents good robustness against noise. Furthermore, it enables the inference

process to be executed in parallel, as the trees operate independently during the prediction phase.

At runtime, no explicit weights are assigned to individual trees, although variations of the model may consider weightings based on local error metrics, such as the incremental mean squared error. In the standard ARF, however, it is assumed that all trees contribute equally to the final prediction.

2.5.3 Sequential Data Processing and Change Detection

Assuming that the data is received as a continuous stream of instances $(\mathbf{x}_t, y_t)$, where t indicates the order of arrival. The model follows the *test-then-train* strategy, in which each instance is initially used for prediction and is immediately incorporated into training.

To deal with possible changes in the underlying concept of the data stream, ARF incorporates drift detection mechanisms at two distinct levels:

- **Internal:** Each tree in the ensemble is equipped with a local drift detector, responsible for monitoring error (or variance) statistics in the branches of the tree itself. When non-stationary behavior is identified, subtrees can be selectively reinitialized.
- **External:** A drift detector associated with each tree monitors the global performance of the model over time. When this detector enters the alert state, a new tree is initialized in the background (background tree). If a change is confirmed, the original tree is replaced by the trained version.

This hybrid strategy enables ARF to maintain a balance between stability and plasticity, selectively replacing only those ensemble components that are significantly affected by changes in the data stream. The combined use of internal and external detectors eliminates the need for global reinitialization, allowing for localized and continuous adaptation.

2.5.4 Computational Complexity

The computational complexity of the model is directly related to the number of trees n, the average depth of the trees, the number of attributes M, and the size of the attribute subset m considered at each node.

During training, each new instance $(\mathbf{x}_t, y_t)$ is forwarded to each tree h_i, which performs a local update. This update involves locating the corresponding leaf, updating local statistics, and eventually evaluating splitting criteria. The cost per tree is approximately $\mathcal{O}(d \cdot m)$, where d is the average depth of the tree and m is the number of attributes considered for splitting at each node. Thus, the total cost of updating the ensemble is $\mathcal{O}(n \cdot d \cdot m)$ per instance.

In the prediction step, each tree performs a traversal from the root to a leaf, whose cost is also $\mathcal{O}(d)$, resulting in a total prediction cost per instance of $\mathcal{O}(n \cdot d)$.

Additionally, the use of drift detectors, such as ADWIN or Page-Hinkley, introduces an extra cost per tree. For ADWIN, the space required is $\mathcal{O}(\log W)$, where W is the number of recent observations monitored. Since the detectors operate independently per tree and do not interfere with the main execution flow, this cost is amortized over time and does not compromise the model's scalability.

In terms of space, ARF stores up to $2n$ trees simultaneously — considering both main trees and their counterparts in training in the background — but only a fraction are active in operation at any given time, which allows efficient execution even in memory-constrained environments.

2.6 Bagging Regressor

The Online *Bagging Regressor* (BaR) (OZA; RUSSELL, 2001) is an ensemble model designed to adapt the bagging technique to operate on a continuous data stream. Its main feature is the way it promotes diversity among the base models. Instead of training each model with every example, the algorithm decides, for each new sample, how many times it will be used to update each model. This process, guided by a probabilistic distribution, simulates the effect of resampling with replacement performed in traditional bagging, but incrementally.

This approach allows each model in the ensemble to be updated independently and with different implicit subsets of data, which increases diversity among them and helps reduce the variance of the final predictor. The arithmetic mean of the individual outputs is used to obtain the model's prediction. Thus, the Online Bagging Regressor reproduces the effects of traditional bagging in a single run through the data, without the need for multiple passes or storage of the whole training set.

2.6.1 Algorithmic structure

BaR is a model aggregation method that processes data sequentially, updating a set of M base models with each new instance received. Its operation is based on the assignment of random multiplicities to each example, in order to induce diversity among the models in the ensemble.

For each new example $d = (x, y)$ that arrives, the method assigns a number of repetitions k to each model h_m, randomly selected according to:

$$k \sim \text{Poisson}(\lambda = 1).$$

The parameter $\lambda = 1$ ensures that, on average, each example is used once per model, but with random variation that promotes diversity. Thus:

- $k = 0$ implies that the example will not be used to update h_m;
- $k > 1$ implies that the example will be used multiple consecutive times to update h_m.

2.6.1.1 Incremental update

Given the random selection of k for a model h_m, it is updated exactly k times using an online learning optimizer L_o, which incorporates the new information without the need to reprocess previous data.

$$\text{for each } m = 1, \ldots, M: \quad k \leftarrow \text{Poisson}(1) \quad \text{repeat } k \text{ times: } h_m \leftarrow L_o(h_m, d)$$

2.6.2 Prediction Function

In BaR, each model h_m generates its prediction $\hat{y}_m$, according to the learning algorithm that composes it. The final decision of the ensemble is obtained by aggregating the numerical outputs, The most common being the arithmetic mean:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} \hat{y}_m,$$

Where M is the total number of models in the ensemble. This procedure seeks to smooth out individual fluctuations and reduce the variance of the estimates.

2.6.3 Computational complexity

In BaR, the computational cost is determined by the number of models in the ensemble (M), the base model update cost (C_{update}), and the prediction cost (C_{pred}).

In the update, for each new example that arrives, the algorithm iterates through all M models, performing:

1. the drawing of $k \sim \text{Poisson}(1)$;
2. up to k updates of the base model with the example.

Since the expected value of k is 1, the expected cost per example is:

$$O\Big(M \cdot C_{\text{update}}\Big).$$

This cost grows linearly with M, but does not depend on the total size of the dataset, since each example is processed only once.

Prediction requires all M models to produce an output. Thus, the cost is:

$$O\Big(M \cdot C_{\text{pred}}\Big),$$

which is also linear in the number of models.

The total size of the ensemble in BaR is proportional to the number of models (M) and the individual size of each base model. Denoting by S_{model} the size (measured in memory) of a single base model, we have:

$$S_{\text{total}} \approx M \cdot S_{\text{model}}.$$

2.7 Bayesian Linear Regression

Bayesian Linear Regression (BLR) (BISHOP, 2006) is a probabilistic model that defines a linear relationship between input variables and a continuous output variable, incorporating uncertainty about the parameters through the use of probability distributions. The model coefficients are treated as random variables with a prior distribution, which is updated based on the observed data using Bayes' rule.

The model assumes that the targets are generated by a linear combination of basis functions, subject to a Gaussian noise term of known variance. The choice of conjugate distributions, such as the normal distribution for the parameters and the noise, allows for the analytical inference of the posterior distribution of the coefficients, as well as the predictive distribution for new examples.

The Bayesian formulation provides, at each step, the distribution of the parameters conditioned on the accumulated data. This probabilistic representation can be updated recursively as new data are observed, without the need to store or reevaluate the complete set of previous observations.

2.7.1 Algorithmic structure

BLR models the scalar output t as a linear combination of basis functions applied to the input $\mathbf{x}$, plus a Gaussian noise term. It is assumed that each output t_n, associated with an input $\mathbf{x}_n \in \mathbb{R}^D$, is generated according to the model:

$$t_n = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n) + \varepsilon_n, \tag{2.1}$$

Where $\boldsymbol{\phi}(\mathbf{x}_n) \in \mathbb{R}^M$ is the vector of basis functions applied to the input, $\mathbf{w} \in \mathbb{R}^M$ is the vector of parameters, and $\varepsilon_n \sim \mathcal{N}(0, \beta^{-1})$ represents Gaussian noise with β precision.

The parameter vector $\mathbf{w}$ is treated as a random variable with a Gaussian prior distribution:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I}), \tag{2.2}$$

Where α is the precision of the prior distribution. Given N observed pairs $(\mathbf{x}_n, t_n)$, the design matrix $\mathbf{\Phi} \in \mathbb{R}^{N \times M}$ is defined, whose rows are given by $\boldsymbol{\phi}(\mathbf{x}_n)^\top$. The likelihood of the output vector $\mathbf{t} = [t_1, \ldots, t_N]^\top$ is then modeled as:

$$p(\mathbf{t} \mid \mathbf{w}) = \mathcal{N}(\mathbf{t} \mid \mathbf{\Phi}\mathbf{w}, \beta^{-1}\mathbf{I}). \tag{2.3}$$

The posterior distribution of the parameters $\mathbf{w}$, obtained by applying Bayes' rule, is also Gaussian and has the form:

$$p(\mathbf{w} \mid \mathbf{t}) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_N, \mathbf{S}_N), \tag{2.4}$$

With:

$$\mathbf{S}_N = \left(\alpha\mathbf{I} + \beta\mathbf{\Phi}^\top\mathbf{\Phi}\right)^{-1}, \tag{2.5}$$

$$\mathbf{m}_N = \beta\mathbf{S}_N\mathbf{\Phi}^\top\mathbf{t}. \tag{2.6}$$

The matrix $\mathbf{S}_N$ represents the posterior covariance of the coefficients, while $\mathbf{m}_N$ is the posterior mean. These terms are updated as new data are observed, allowing for progressive refinement of the model. In dataflow contexts, updating can be performed sequentially using recursive matrix techniques.

2.7.2 Prediction Function

After training on a set of observations, Bayesian linear regression provides a posterior distribution for the parameters $\mathbf{w}$. The prediction for a new input $\mathbf{x}_*$ is obtained by marginalizing $\mathbf{w}$ into the joint distribution of the model, resulting in a predictive distribution for the target variable t_*. This distribution accounts for both the uncertainty in the parameters and the noise in the data.

Let $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$ be the vector of basis functions applied to the new input. The predictive distribution for t_* is a univariate normal distribution, given by:

$$p(t_* \mid \mathbf{x}_*, \mathbf{t}) = \mathcal{N}(t_* \mid \mu_*, \sigma_*^2), \tag{2.7}$$

Where:

$$\mu_* = \boldsymbol{\phi}_*^\top\mathbf{m}_N, \tag{2.8}$$

$$\sigma_*^2 = \boldsymbol{\phi}_*^\top\mathbf{S}_N\boldsymbol{\phi}_* + \beta^{-1}. \tag{2.9}$$

The mean μ_* represents the point prediction of the model, while the variance σ_*^2 expresses the uncertainty associated with the prediction. The first term of the variance reflects the uncertainty in estimating the parameters. In contrast, the second term corresponds to the variance of the noise in the data-generating process.

2.7.3 Computational complexity

The computational complexity of Bayesian linear regression is associated with the matrix operations involved in updating the posterior distribution and making predictions.

Updating the posterior requires calculating the mean $\mathbf{m}_N$ and the covariance matrix $\mathbf{S}_N$, which involve matrix products and the inversion of a matrix of dimension $M \times M$, where M is the number of model parameters. Direct inversion has a complexity of $\mathcal{O}(M^3)$. However, in sequential update contexts, incremental update techniques such as the Woodbury identity can be employed, reducing the cost to, for example, $\mathcal{O}(M^2)$.

Prediction for a new input involves calculating the mean and variance of the predictive distribution. The mean μ_* is obtained via the scalar product between vectors of dimension M, with computational cost $\mathcal{O}(M)$. The variance σ_*^2 requires the calculation of $\boldsymbol{\phi}_*^\top \mathbf{S}_N \boldsymbol{\phi}_*$, with cost $\mathcal{O}(M^2)$. In terms of storage, the model maintains the covariance matrix $\mathbf{S}_N$, with cost $\mathcal{O}(M^2)$, and the mean vector $\mathbf{m}_N$, with cost $\mathcal{O}(M)$.

2.8 Exponentially Weighted Average Regressor

Exponentially Weighted Average regressor (EWA) (KIVINEN; WARMUTH, 1997) is an online learning model that uses a weighted combination of predictors to make predictions on a continuous data stream. In this model, each predictor receives weights that evolve dynamically as new data is seen. These weights represent the relative confidence in each model and are adjusted exponentially in response to the error made in each prediction. With this, the model values, in its aggregation, the predictors that demonstrate better performance throughout the execution.

The update of the weights follows a multiplicative rule. This characteristic allows EWA to adjust itself in a sensitive but stable way to variations in the relevance of each predictor. The resulting aggregation process maintains an importance distribution that continuously reflects the individual performance history, without requiring reprocessing of previous data or structural changes in the set of models.

2.8.1 Model Architecture

EWA is a model based on an adaptive combination of predictors, in which a fixed set of M base models $\{f_1, f_2, \ldots, f_M\}$ is maintained over time. Each model f_i is a regressor that operates independently of the others and generates an individual prediction from an input instance

$x_t \in \mathbb{R}^d$, such that $\hat{y}_t^{(i)} = f_i(x_t)$. The final prediction of the model at time t is calculated as the weighted average of the individual predictions, according to the expression:

$$\hat{y}_t = \sum_{i=1}^{M} w_t^{(i)} \cdot \hat{y}_t^{(i)}, \tag{2.10}$$

where $w_t^{(i)}$ represents the weight assigned to the model f_i at time t, with the constraint $\sum_{i=1}^{M} w_t^{(i)} = 1$. The weights indicate the relative confidence in the performance of each model, based on the history of accumulated losses up to the current time.

After observing the actual value $y_t \in \mathbb{R}$, the weights are updated based on the individual loss of each model. If $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ is the chosen loss function, the EWA Regressor uses, by default, the quadratic loss defined by $\ell(y, \hat{y}) = (y - \hat{y})^2$. This penalty mechanism is implemented through a multiplicative update rule, which defines the new weight distribution as follows:

$$w_{t+1}^{(i)} = \frac{w_t^{(i)} \cdot \exp\left(-\eta \cdot \ell(y_t, \hat{y}_t^{(i)})\right)}{\sum_{j=1}^{M} w_t^{(j)} \cdot \exp\left(-\eta \cdot \ell(y_t, \hat{y}_t^{(j)})\right)}, \tag{2.11}$$

where $\eta > 0$ is the learning rate, responsible for regulating the sensitivity of the penalty applied to the models as a function of their predictive errors. In this way, the exponential form of the penalty ensures that models with higher losses have their influence reduced in proportion to the severity of their error.

This normalization, applied to the denominator of the equation, ensures that the weights are maintained in a convex distribution at each iteration, keeping the weights within the interval $(0, 1)$ and adding up to exactly 1. In this sense, this structure allows EWA to update its decision function continuously and efficiently, without the need to reprocess the data history or reconfigure the ensemble structure.

2.8.2 Ensemble Structure

EWA is composed of a fixed set of previously defined base models. These models can be instances of different algorithms or variations of the same algorithm with different configurations. There are no requirements for homogeneity among the predictors; they can be the same or completely different, as long as they produce numerical outputs compatible with the regression task.

One of the main characteristics of EWA is that its adaptation to new concepts does not occur through updating the base models, which remain unchanged over time, regardless of variations in the data. Adaptation occurs through the adjustment of the weights associated with each model, which are modified to increase the influence of the most effective predictors while reducing the influence of the least accurate ones. These weights are updated based on the relative performance of each component, using a multiplicative update rule that penalizes models with the highest predictive error more severely in each observed instance.

Finally, the final prediction made by EWA is obtained through a convex combination of the outputs of the base models, in which the weights associated with each component are non-negative and add up to exactly one. This aggregation strategy aims to promote numerical stability, avoid unwanted amplifications in the prediction, and ensure that the result remains within the range of the individual outputs. Additionally, it enables a direct interpretation of the relative contribution of each predictor, facilitating the monitoring of the ensemble over time and identifying potential changes in the data flow dynamics.

2.8.3 Computational Complexity

The computational complexity analysis of EWA can be divided into two main stages: the prediction stage and the weight update stage. Both stages present a linear cost in terms of the number of models that make up the ensemble, denoted by M, which provides the model with controlled scalability in applications involving multiple predictors.

During the prediction stage, each base model f_i is evaluated individually on the input instance $x_t \in \mathbb{R}^d$, producing an estimate $\hat{y}_t^{(i)} = f_i(x_t)$. These estimates are then combined using a weighted average, with weights in effect at time t. The computational cost of this stage is therefore proportional to $M \cdot C_{\text{pred}}$, where C_{pred} represents the cost of evaluating a base model. In the case of linear models, for example, this cost is typically $\mathcal{O}(d)$, resulting in a total prediction complexity of the order of $\mathcal{O}(M \cdot d)$ per instance.

The update phase occurs after the observation of the real value y_t. It involves calculating the individual losses $\ell(y_t, \hat{y}_t^{(i)})$ for each predictor, followed by reweighting the weights based on an exponential loss function. The subsequent normalization ensures that the sum of the weights is unity. This entire process is performed with a linear cost in M, since each term is computed independently and without the need for interactions between the models. Thus, the complexity of the update step is $\mathcal{O}(M)$ per iteration.

From a storage point of view, EWA only requires the maintenance of the current weights and the internal parameters of each base model. Since the method does not store previous instances or perform backpropagation, memory usage remains constant throughout execution, regardless of the number of instances processed. This feature, combined with the independence between predictors, makes EWA compatible with parallel implementations and applications in environments with memory or response time constraints.

2.9 Hoeffding Tree Regressor

Hoeffding Tree Regressor (HTR) is a tree model designed to operate on data streams, utilizing the principle of incremental learning. Its development is based on the extension of the VFDT (Very Fast Decision Tree) algorithm proposed by (DOMINGOS; HULTEN, 2000), initially intended for classification tasks. For regression, the HTR adapts the central mechanism

of the VFD, which uses the Hoeffding limit to perform divisions in the internal nodes of the tree, to the context of continuous variables by replacing entropy criteria with measures of variance or mean squared error.

Its structure consists of a binary decision tree whose internal nodes perform divisions based on predictor attributes. At the same time, the leaves maintain only the statistics necessary to estimate the quality of possible candidate divisions. These divisions are continuously evaluated based on incoming data, but are only implemented when sufficient statistical evidence is reached, as determined by the Hoeffding limit. Due to this approach, the model does not require storing examples, allowing the tree to grow in a selective and controlled manner.

2.9.1 Hoeffding bound

The decision to perform splits on internal nodes of the HTR is based on a statistical result known as the Hoeffding bound (HOEFFDING; ROBBINS, 1994). This bound establishes a maximum value for the difference between the empirical mean of a function of independent random variables and its real expectation, with a previously specified confidence level. This property enables the model to make decisions about the tree structure based on partial samples, without requiring the reprocessing of previously observed data.

In the context of HTR, the quality of a partition is measured by criteria appropriate to regression, such as the reduction in the variance of the target variable's values after the partition. For each attribute considered as a candidate for partition, the statistic G is calculated, which quantifies the expected gain when partitioning the set of examples. Assuming that G_1 and G_2 are the most significant observed values among the candidates, the model proceeds with the division if:

$$G_1 - G_2 > \epsilon$$

Where ϵ is the Hoeffding limit, given by:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}},$$

where R is the maximum possible interval of the function G, δ is the confidence parameter (usually small), and n is the number of examples observed at the node.

This criterion ensures that, with probability at least $1 - \delta$, the choice of attribute G_1 as divisor results in the best available division based on the accumulated information. In this way, the model maintains its incremental nature, performing local and controlled updates.

2.9.2 Model Architecture

The Hoeffding Tree Regressor (HTR) works by incrementally building its structure through sequential processing of individual examples, a characteristic of continuous flow learning

environments. With each new instance received, the model starts routing from the root of the tree, successively evaluating the instance's attributes according to the tests defined in the internal nodes. This process is repeated until the example reaches a leaf.

The leaves, in turn, store sufficient statistics to evaluate future divisions, without the need to keep the examples in memory. In the context of regression, these statistics typically include the number of observed instances, the sums of the values of the target variable, and, when necessary, the sums of the squares of these values. From this data, it is possible to estimate metrics such as the variance of the dependent variable for each possible division induced by candidate attributes.

As data progressively accumulates in a leaf, the model continually reevaluates the statistical gain associated with each predictor attribute as a potential divisor. When the condition established by the Hoeffding bound is satisfied, the leaf is converted into an internal node. This node represents a new division in the tree, and new leaves are allocated for the resulting partitions. Each leaf is initialized with independent statistical storage structures and operates autonomously in the processing of subsequent instances.

2.9.3 Prediction function

Prediction is performed based on statistics accumulated in the leaves of the tree. At each new instance, the model routes the sample from the root to a terminal leaf, according to the tests defined in the internal nodes. Upon reaching the corresponding leaf, the estimate of the target variable is calculated, by default, using the arithmetic mean of the values previously observed in that leaf.

This prediction strategy assumes that, in locally homogeneous regions of the input space, the empirical mean of the target variable values provides a reasonable approximation of the central tendency. The simplicity of this approach contributes to the computational efficiency of the model in continuous flow scenarios, where it is necessary to respond to instances in real-time without incurring computational overhead.

Alternatively, more expressive local predictive models can be used, such as incremental linear regressions trained with the examples accumulated in the leaf. This type of modeling allows capturing linear relationships between the attributes and the target variable in contexts where the simple mean is not an adequate approximation. Empirical criteria usually determine the decision to use local predictors and may depend on the number of examples accumulated in the leaf or the desired computational complexity.

The stability of predictions is directly related to the volume of data observed in each leaf. Leaves that have accumulated a few instances tend to produce estimates with greater variability. In contrast, those that receive a sufficient number of examples tend to generate more stable predictions and are representative of the local conditions of the data stream.

2.9.4 Computational complexity

The update time per instance is $O(d)$, where d represents the depth of the tree at the time the example arrives. This cost arises from routing the instance from the root to the corresponding leaf. At the end of the routing, updates are made to the local statistics of the leaf, the cost of which is $O(a)$, with a being the number of predictor attributes. In addition, for every n_{min} examples observed by a leaf, the model evaluates the gains of splitting for the candidate attributes, with a cost proportional to $O(a)$ per attribute, depending on the nature of the attributes, which can be discrete or continuous, and the granularity of the partitioning considered.

The inference time per instance is also $O(d)$, since it consists solely of routing to a leaf and applying a prediction function based on local statistics, such as the arithmetic mean of the previously observed values.

Memory consumption is directly related to the number of leaves L and the statistics stored per leaf. For each attribute evaluated in a leaf, counters and summations associated with each possible value are maintained, in the case of discrete attributes, or with partitions in the case of continuous attributes. Assuming that s statistics are stored per attribute, the memory complexity per leaf is $O(a \cdot s)$, resulting in a total complexity of $O(L \cdot a \cdot s)$. The model does not store individual examples, which limits memory growth to the structural size of the tree and the number of monitored attributes.

The depth of the tree, the number of attributes, and the frequency of split evaluation determine the total computational cost over time. These factors vary dynamically according to the data distribution and the structural evolution of the model.

2.10 Hoeffding Adaptive Tree Regressor

The *Hoeffding Adaptive Tree Regressor* (HAT) (BIFET; GAVALDA, 2009) is a decision tree-based model that exploits statistical properties of the Hoeffding bound to perform splits between nodes. The model maintains, at each node, estimators that record statistics for calculating the split criterion, allowing the tree structure to be expanded progressively. The decision to split a node occurs only when the difference between the split gains of the candidate attributes exceeds a threshold, allowing the model to evolve gradually as it observes new examples.

A distinctive feature of the model is the use of local concept detectors, incorporated directly into the nodes, to monitor changes in the statistical distributions that guide the splits. Such detectors allow for the identification of variations in the input patterns, triggering the creation of alternative subtrees that compete with the current structure. This mechanism enables the selective replacement of regions of the tree with more recent versions, thereby maintaining the statistical coherence of the model even in the face of variations in the data flow. In this way, the model promotes localized adaptation guided by empirical evidence, without the need for

global parameters or fixed time windows.

2.10.1 Algorithmic structure

The tree is constructed incrementally. Each new instance $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$, consisting of a vector of input attributes $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and a real target variable y, is routed from the root to a leaf of the tree. Upon reaching a leaf l, the model updates local statistics that will be used to evaluate the convenience of splitting the node.

The split decision is based on the comparison between different candidate attributes. For each attribute A_j, a *measure of merit* $G(A_j)$ is calculated, which quantifies the benefit of splitting the data based on A_j. In the context of regression, this measure corresponds to the expected reduction in the variance of the variable y after splitting. Given a set S of examples arriving at the node, the measure of merit can be expressed as:

$$G(A_j) = \sigma^2(S) - \sum_k \frac{|S_k|}{|S|}\sigma^2(S_k),$$

Where:

- $\sigma^2(S)$ is the variance of y in the total set S;
- S_k are the subsets generated by splitting S based on a value or range of the attribute A_j;
- $|S_k|$ is the number of examples in the subset S_k;
- $|S|$ is the total number of examples in the current node.

This equation represents the weighted reduction of the variance resulting from the division. The larger $G(A_j)$, the greater the homogeneity, in terms of y, in the partitions generated by A_j, which indicates a better division.

The tree maintains, in each leaf, the estimates of $G(A_j)$ for all attributes. Denoting by $A_{(1)}$ and $A_{(2)}$ the two attributes with the highest and second highest merit, respectively, the difference between them is defined as:

$$\Delta G = G(A_{(1)}) - G(A_{(2)}).$$

Node splitting is performed if ΔG exceeds a threshold ε, calculated based on the Hoeffding bound:

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

In this expression:

- R is the maximum possible range of the measure of merit G; in the case of variance, R can be bounded by the known range of the variable y;
- δ is a confidence parameter (e.g., 10^{-5}), which defines the desired degree of statistical certainty;
- n is the number of observed instances in the leaf l.

If the condition $\Delta G > \varepsilon$ is satisfied, the node is split according to the attribute $A_{(1)}$, and two new leaves are created for the split branches, with initialized statistics.

To allow adaptation to changes in the data distribution over time, the model incorporates concept drift detectors at each node. These detectors monitor statistics associated with the attributes or the response variable. When they detect a significant change, an alternative subtree is started from that node. This new subtree is built parallel to the main structure, given the same examples, and evolves based on recent data.

Let T be the current tree and T_{alt} be the alternative subtree. Both accumulate estimates of the prediction error over time. The mean performance difference is defined as:

$$\Delta E = \hat{L}(T) - \hat{L}(T_{\text{alt}}),$$

Where $\hat{L}(T)$ and $\hat{L}(T_{\text{alt}})$ are the mean error estimates over a recent window of observations, such as root mean square error (RMSE) or mean absolute error (MAE). When ΔE is significantly positive, the alternative subtree is promoted and replaces the corresponding structure in the main tree. This local replacement process preserves the rest of the tree, allowing selective updating as needed.

Each node in the tree maintains its statistics through incremental estimators. In the case of the HAT-ADWIN version, the ADWIN algorithm is employed, which maintains an adaptive window of recent data and updates the estimate $\hat{\mu}_t$ of the local mean with each new example. ADWIN automatically adjusts the window size and detects changes based on internal statistical tests, controlling sensitivity to drift in each region of the tree independently.

2.10.2 Prediction function

The prediction function in the HAT consists of estimating the value of the target variable $\hat{y} \in \mathbb{R}$ for a new input instance $\mathbf{x} \in \mathbb{R}^d$, based on the current structure of the tree. The prediction process is deterministic and follows the path of the tree from the root to a leaf, successively applying the splitting conditions at each internal node. The path taken depends exclusively on the values of $\mathbf{x}$ and the partitions established by the tree.

Upon reaching a leaf l, the predicted value $\hat{y}$ is calculated based on the local statistics of the response variable y accumulated at that leaf. In its simplest form, the prediction corresponds to the average of the values y observed so far at the leaf:

$$\hat{y} = \frac{1}{n_l} \sum_{i=1}^{n_l} y_i,$$

where n_l is the number of instances that have passed through leaf l so far, and y_i are the target values associated with those instances. This average can be updated incrementally with each new example, without the need to store the individual values y_i explicitly.

In variants of the model, such as those using Naïve Bayes classifiers at the leaves, the prediction may incorporate a conditional model of the output y based on $\mathbf{x}$. Nevertheless, the final value remains a deterministic and local function of the statistics stored at the terminal node reached.

The stability of the prediction depends on the persistence of the tree structure and the statistics at the leaves. However, the presence of change detectors allows the tree to reorganize itself in regions where significant variations are detected. In such situations, alternative subtrees can be promoted, replacing previously stable leaves. Thus, the model preserves predictive accuracy even in dynamic contexts, by balancing stability in stationary regions and local adaptation in regions subject to drift.

2.10.3 Computational complexity

The computational cost of HAT per instance is determined by the number of operations involved in updating the tree and calculating the prediction. Since the model operates entirely online, processing each instance uniquely and incrementally, its complexity is more related to the depth and number of nodes in the tree than to the total volume of observed data.

When receiving a new instance $(\mathbf{x}, y)$, the model performs the following steps:

1. **Routing** of the instance from the root to a leaf: this path involves comparisons at each internal node and has a cost proportional to the depth of the tree, denoted by d_T. On average, $d_T = \mathcal{O}(\log N)$, where N is the number of examples already observed.

2. **Statistics update** at the reached leaf: for each attribute A_j, statistics such as mean and variance are updated incrementally, with a cost of $\mathcal{O}(1)$ per attribute. In the case of using detectors such as ADWIN, updating each estimator has an amortized cost of $\mathcal{O}(\log W)$, where W is the adequate size of the window maintained by the detector.

3. **Split condition check**: after a certain minimum number of examples, the node checks whether the difference between the merits of the attributes satisfies the criterion based

on the Hoeffding bound. Calculating the merit for all attributes incurs a linear cost in the number of attributes and the possible values per attribute.

Thus, the update time per instance is given by:

$$\mathcal{O}\left(d_T \cdot (d \cdot \log W + C)\right),$$

Where:

- d_T is the depth of the tree;
- d is the number of attributes;
- W is the effective size of the ADWIN window;
- C is the fixed cost for maintaining the structure and controlling divisions.

The prediction for a new instance involves traversing the tree from the root to a leaf and calculating the average of the values stored in the leaf. Since this process does not involve the calculation of complex statistics or aggregation between models, the prediction time is:

$$\mathcal{O}(d_T),$$

Where d_T is again the depth of the tree, since this depth increases slowly with the number of instances, the prediction remains efficient even with large volumes of data.

The memory usage of the model is proportional to the number of nodes in the tree and the auxiliary structures maintained in each node. Let n_{nodes} be the total number of nodes (internal and leaves), and s be the space required to maintain the statistics and detectors per attribute. The total memory consumption is given by:

$$\mathcal{O}(n_{\text{nodes}} \cdot d \cdot s),$$

Where:

- d is the number of attributes;
- $s = \mathcal{O}(\log W)$ in the case of using ADWIN (each estimator requires memory proportional to the logarithm of the window);
- Active alternative subtrees add factor, proportional to the number of detectors that have recently signaled a change.

In practice, the growth of the tree and the number of alternative subtrees can be controlled by parameters such as the minimum number of instances per node, the depth limit, and the criteria for eliminating unpromising subtrees.

2.11 K-Nearest Neighbors Regressor

The *K-Nearest Neighbors Regressor* (KNN) model is a non-parametric supervised learning technique focused on regression tasks. Unlike methods based on explicit functions or parametric updates, KNN operates under the locality principle: the prediction of an example is inferred from the closest examples previously observed in the input space.

In an online context, the KNN is adapted to process data increasingly, maintaining a limited buffer from previous observations. Each new instance observed is incorporated into the model's memory, while older instances are discarded according to a substitution strategy, commonly a fixed sliding window. This approach gives the model the capacity for continuous adaptation, essential for non-stationary contexts and subject to concept changes.

Instead of building an explicit model, the KNN represents data representations in their original form. Thus, the predictive capacity of the model is directly influenced by the local distribution of examples stored in the characteristics space, as well as the choice of similarity metric adopted.

2.11.1 Algorithmic structure

The KNN Regressor model, in its online formulation, performs instance-based learning. Instead of adjusting parameters or estimating explicit functions, the model maintains a subset of the most recently observed examples, which serve as a reference for the prediction stage.

To this end, consider a supervised data stream described by a temporal sequence of labeled instances $\{(x_t, y_t)\}_{t=1}^{\infty}$, where each $x_t \in \mathbb{R}^d$ represents a feature vector of dimension d, and $y_t \in \mathbb{R}$ is the corresponding target value observed at time t.

Since KNN operates exclusively on stored examples, a buffer is defined to store the most recently observed instances. This structure is represented by the set $\mathcal{B}_t \subseteq \{(x_i, y_i)\}_{i=1}^{t}$, which contains, at most, $n \in \mathbb{N}$ elements. The value of n imposes an explicit restriction on the model's memory capacity and defines the time horizon of the information considered relevant for predictive purposes.

The buffer is updated incrementally. With each new example (x_{t+1}, y_{t+1}), the model incorporates this information into its memory, discarding the oldest instance if the storage limit has already been reached. The following rule can describe this process:

$$\mathcal{B}_{t+1} = \begin{cases} \mathcal{B}_t \cup \{(x_{t+1}, y_{t+1})\}, & \text{if } |\mathcal{B}_t| < n \\ (\mathcal{B}_t \setminus \{(x_{t-n+1}, y_{t-n+1})\}) \cup \{(x_{t+1}, y_{t+1})\}, & \text{otherwise} \end{cases}$$

That is, the model preserves the n most recent instances observed up to time $t + 1$, automatically discarding older observations as necessary. This mechanism ensures that the model is always up-to-date with the most recent data history.

It is important to note that, unlike parametric models, the KNN Regressor does not perform any functional optimization on the stored data. Learning occurs exclusively through the retention and replacement of instances, which then form the basis for comparison during the prediction process.

2.11.2 Prediction Function

To perform the prediction at a given time t, the model uses its current memory $\mathcal{B}_t = \{(x_i, y_i)\}_{i=1}^{n}$, composed of the last n stored instances. The objective is to estimate the target value $\hat{y} \in \mathbb{R}$ corresponding to a new input instance $x \in \mathbb{R}^d$.

The prediction is based on the search for the K nearest neighbors of x within $\mathcal{B}_t$, according to a distance metric $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$. Let:

$$\mathcal{N}_K(x) = \{(x_{(i)}, y_{(i)})\}_{i=1}^{K} \subset \mathcal{B}_t$$

the set of K pairs (x_i, y_i) with the shortest distance from x, ordered in ascending order by $d(x, x_{(i)})$.

The predicted output $\hat{y}$ is obtained through an aggregation function applied to the target values of the found neighbors. The most common forms are:

(a) Simple Average:

$$\hat{y} = \frac{1}{K} \sum_{i=1}^{K} y_{(i)}$$

(b) Median:

$$\hat{y} = \text{median}\{y_{(1)}, y_{(2)}, \ldots, y_{(K)}\}$$

(c) Inverse Distance Weighted Average:

$$\hat{y} = \frac{\sum_{i=1}^{K} \frac{1}{d(x, x_{(i)})} \cdot y_{(i)}}{\sum_{i=1}^{K} \frac{1}{d(x, x_{(i)})}}$$

To avoid numerical instability, the convention is that if there exists $x_{(i)} \in \mathcal{B}_t$ such that $d(x, x_{(i)}) = 0$, then $\hat{y} = y_{(i)}$, assuming identity between the input instance and a stored instance.

The prediction function in the KNN is therefore defined in terms of the local neighborhood in the input space, without explicitly estimating a global function. This characteristic classifies it as a nonparametric method, dependent exclusively on the instances stored at the current time.

2.11.3 Computational Complexity

Analyzing the computational complexity of the online KNN Regressor model involves two primary operations: updating the model's memory with new instances (learning) and calculating predictions based on nearest neighbors (inference). Both depend directly on the buffer size n, the dimensionality of the feature space d, and the number of neighbors K used in the regression.

The learning step consists of adding a new instance to the buffer $\mathcal{B}_t$, possibly accompanied by the removal of the oldest instance if the storage limit has already been reached. This operation does not involve any calculations on the data; it only involves insertion and deletion operations.

- Time cost: $\mathcal{O}(1)$, assuming a data structure with direct access (e.g., circular queue).
- Space cost: $\mathcal{O}(nd)$, where n is the buffer size and d is the dimensionality of the vectors x_t.

Prediction requires identifying the K nearest neighbors of an input instance $x \in \mathbb{R}^d$, based on a distance metric (e.g., the Euclidean distance). It means that calculating the distance between x and each of the n stored instances, followed by selecting the K shortest distances.

- Distance calculation: $\mathcal{O}(nd)$, since it involves n vectors of dimension d.
- Selection of the K smallest values: $\mathcal{O}(n)$, using partial selection algorithms (such as QuickSelect); or $\mathcal{O}(n \log K)$ if kept in a sorted heap.
- Aggregation (mean, median, weight): $\mathcal{O}(K)$.

Thus, the total inference cost is dominated by the neighbor search step, resulting in a complexity of:

$$\mathcal{O}(nd) \quad \text{(time)} \quad \text{e} \quad \mathcal{O}(nd) \quad \text{(space)}$$

Finally, the model size is determined exclusively by the contents of the buffer $\mathcal{B}_t$, which stores up to n instances, each consisting of a feature vector of dimension d and a scalar target value. Thus, the total space required to store the model is proportional to:

$$\mathcal{O}(nd)$$

This value represents the amount of memory required to maintain the observation history that underlies future predictions. Since the model does not perform compression, abstraction, or transformation of the stored data, the space occupied grows linearly with the buffer size and the data dimensionality.

2.12 Linear Regression

The *Linear Regression* (LiR) (STARBUCK, 2023) is a model that represents the output as a linear combination of the input attributes, with adjustable weights associated with each variable. The equation used to generate the predictions is predefined and remains fixed over time; the learning process consists exclusively of continuously updating the coefficients. With each new observed instance, the model calculates the error between the predicted value and the actual value and modifies the coefficients based on the direction of the loss function's gradient.

The loss function used is typically the quadratic error, which penalizes discrepancies between the prediction and the observed value. The update is performed by adjusting the coefficients in the opposite direction of the gradient, with intensity controlled by a learning rate. Thus, the model operates based on this predefined structure, updating the parameters incrementally with each iteration.

2.12.1 Algorithmic structure

LiR models the relationship between an attribute vector $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ and a continuous response variable $y \in \mathbb{R}$ through a parameterized linear function of the form:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} = \sum_{j=1}^{d} w_j x_j$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$ represents the vector of adjustable coefficients. At each instant t, the model receives a new instance $(\mathbf{x}^{(t)}, y^{(t)})$, makes a prediction $\hat{y}^{(t)} = \mathbf{w}^{(t)\top}\mathbf{x}^{(t)}$, and calculates the corresponding error.

The parameter update is based on the minimization of a loss function, such as the quadratic error function:

$$\ell^{(t)}(\mathbf{w}) = \frac{1}{2}(y^{(t)} - \hat{y}^{(t)})^2 = \frac{1}{2}(y^{(t)} - \mathbf{w}^\top \mathbf{x}^{(t)})^2$$

To minimize this function incrementally, the Stochastic Gradient Descent (SGD) algorithm is used, whose coefficient update rule is given by:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta\,(y^{(t)} - \hat{y}^{(t)})\,\mathbf{x}^{(t)}$$

Where $\eta > 0$ is the learning rate, which controls the magnitude of the correction applied. The expression $(y^{(t)} - \hat{y}^{(t)})\,\mathbf{x}^{(t)}$ corresponds to the gradient of the loss function concerning the coefficients.

This process is repeated iteratively for each new instance received, allowing the parameters to be continuously adjusted in response to the observed data.

2.12.2 Prediction Function

The prediction function is defined as a linear combination of the input features and the adjusted coefficients over time. Given an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ and a coefficient vector $\mathbf{w} = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$, the prediction is computed as follows:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} = \sum_{j=1}^{d} w_j x_j$$

This expression corresponds to the inner product between the coefficient vector and the feature vector. The resulting value, $\hat{y} \in \mathbb{R}$, represents the model's estimate for the response variable associated with the input $\mathbf{x}$.

The prediction calculation assumes that the coefficients $\mathbf{w}$ have already been adjusted by previous iterations of the learning algorithm. The required operation consists only of vector multiplication, whose computational complexity is linear with the number of attributes.

2.12.3 Computational Complexity

The computational cost per instance of online linear regression is determined by the dimensionality of the feature vector $\mathbf{x}_t \in \mathbb{R}^d$, since all relevant operations—prediction and update—occur directly on this vector.

Prediction is performed through the inner product between the input vector and the model coefficient vector, calculated as $\hat{y}_t = \mathbf{w}_t^\top \mathbf{x}_t$. This operation involves d multiplications and $d - 1$ additions, resulting in inference complexity linear in the number of features:

$$\mathcal{O}_{\text{inf}} = \mathcal{O}(d)$$

After prediction, the model performs the coefficient update step based on the observed error. The Stochastic Gradient algorithm updates the weight vector according to the rule $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta(y_t - \hat{y}_t)\mathbf{x}_t$, which again involves a dot product, followed by a scalar-vector multiplication and a vector sum, each with a cost proportional to d. Thus, the update time per instance is also given by:

$$\mathcal{O}_{\text{upt}} = \mathcal{O}(d)$$

The model maintains only the coefficient vector $\mathbf{w}_t \in \mathbb{R}^d$ in memory. There are no auxiliary structures or storage of previous instances, and there is no structural growth over time. Therefore, memory usage remains constant and proportional to the input dimensionality:

$$\mathcal{O}_{\text{mem}} = \mathcal{O}(d)$$

This computational profile makes the model sensitive to the choice of input attributes, since the time and space cost grow linearly with d. However, there is no direct dependence on the total number of instances processed up to time t.

2.13 Online Extra Trees Regressor

The *Online Extra Trees Regressor* (OXT) (MASTELINI et al., 2023) is an ensemble model composed of decision trees, inspired by the batch Extra Trees (XT) algorithm. Its main structural feature is the generation of splits in the tree nodes in a completely random manner, without resorting to optimization heuristics. In each split attempt, a random subset of attributes is selected and, for each attribute, a cut point is chosen stochastically, without merit evaluation. This approach aims to create trees with varied structures and reduced susceptibility to overfitting, thereby seeking to favor diversity within the ensemble.

Additionally, the model can employ subsampling without replacement (subbagging), assigning each instance of the flow a fixed probability of updating each tree. This strategy aims to reduce computational cost and promote independence among the ensemble members. The final prediction is obtained by simply aggregating the tree outputs, typically via averaging.

2.13.1 Algorithmic structure

OXT consists of a set of N_t decision trees that evolve independently from the data stream. Each tree is updated incrementally, processing instances as they are received. Induction occurs based on random splits: at each split attempt at a leaf, a random subset of attributes is selected and, for each attribute, a cut point φ is sampled uniformly within an estimated range from an initial buffer of examples. The algorithm does not use traditional merit metrics, such as global variance reduction, to compare multiple split candidates; it only checks whether the proposed split meets a minimum criterion of statistical significance, defined based on a relaxed version of the Hoeffding bound.

To perform these splits, each leaf node maintains an attribute observer responsible for estimating the attribute range based on a limited buffer of s instances. After filling the buffer, the observer sets a random cut point $\varphi \sim \mathcal{U}(x_f^{\min}, x_f^{\max})$ and starts to incrementally update conditional variance statistics on the two branches defined by the split $x_f \leq \varphi$ and $x_f > \varphi$, based on estimators derived from Welford's algorithm.

The quality of the split is estimated using variance reduction:

$$\Delta\sigma^2 = \sigma^2_{\text{parent}} - \left(\frac{n_{\leq}}{n}\sigma^2_{\leq} + \frac{n_{>}}{n}\sigma^2_{>}\right),$$

Where σ^2_{parent} is the variance of the target value in the current leaf, n is the total number of instances, and $\sigma^2_{\leq}$, $\sigma^2_{>}$ are the variances of the output values in the corresponding subsets to the left and right of the cut point. If the estimate of $\Delta\sigma^2$ is high enough, the split is applied and two new leaf nodes are created, each with its observers. The process repeats as new instances are received, promoting incremental growth of the tree.

2.13.2 Prediction function

Prediction in OXT is performed by aggregating the individual outputs of each tree in the ensemble. For an input instance $\mathbf{x}$, each tree routes $\mathbf{x}$ to one of its leaves, where a local model is used to generate an estimate $\hat{y}_i$. The arithmetic mean of the individual predictions obtains the final ensemble prediction:

$$\hat{y} = \frac{1}{N_t}\sum_{i=1}^{N_t}\hat{y}_i,$$

Where N_t is the total number of trees in the ensemble.

The predictive model present at each leaf can take different forms. The basic version uses the mean of the target values observed at that leaf, updated incrementally with each new instance. Alternatively, adaptive linear models trained online with error-based update mechanisms can be used, which tend to offer greater predictive expressivity in regions where the behavior is linear. The choice between the approaches depends on the allowable computational cost and the expected complexity of the target function.

2.13.3 Computational complexity

The computational complexity of the model is mainly determined by three factors: the number of trees in the ensemble, the average depth of the trees, and the randomization mechanisms adopted during induction. Since each instance of the stream is processed only once, the update cost per instance is proportional to the average number of trees that receive the sample, that is, $p \cdot N_t$, where p is the subsampling rate and N_t is the total number of trees.

In each tree, the update cost is primarily determined by routing the instance to a leaf and subsequently updating the local predictive model. Since there is no exhaustive evaluation of split candidates, and since only one cut point per attribute is maintained, the cost per leaf node is significantly lower than in algorithms that compare multiple points or attributes. The update of the variance statistics is done incrementally, with a constant cost per instance.

During inference, the computational cost is proportional to the number of trees, since each instance is routed to a leaf in each tree. The prediction is then obtained by aggregation,

typically an average operation with a linear cost in N_t. Thus, the total inference cost per instance is $\mathcal{O}(N_t \cdot h)$, where h represents the average depth of the trees.

As for memory consumption, each tree stores only the active nodes and their respective attribute observers, the number of which depends on the tree's depth and width. Assuming d input attributes and h levels per tree, the total memory required can be estimated as $\mathcal{O}(N_t \cdot d \cdot h)$, disregarding the cost of predictive models at the leaves, which can vary depending on the adoption of simple averages or linear models. Overall, OXT presents a light computational profile, with moderate memory usage and good scaling with the number of trees and data complexity.

2.14 Passive-Aggressive Regressor

The *Passive-Aggressive Regressor* (PAR) (CRAMMER et al., 2006) model is a linear regression model whose update is formulated as a convex optimization problem with constraints. The main characteristic of PAR is that its parameters are updated only when the prediction made on an instance presents an error greater than a tolerance threshold, defined by an insensitivity parameter ε. Otherwise, the model remains unchanged.

The PAR update rule seeks to minimize the distance between the new weight vector and the previous vector, under the constraint that the loss on the current instance is zero or less than the value of ε. This approach aims to maintain stability regarding irrelevant variations in the data and makes punctual adjustments when the predictive performance is considered unsatisfactory. The formulation admits variants with explicit regularization, allowing the magnitude of the updates to be controlled based on an aggressiveness parameter.

2.14.1 Intuition of passive-aggressive operation

The operation of the PAR model is based on a principle of balance between two opposing strategies: keeping the parameters unchanged when the prediction is considered acceptable (passive behavior) and performing an immediate and direct correction when the prediction violates a loss condition (aggressive behavior).

This duality is operationalized through a conditional update rule. When the prediction error falls within a tolerance margin ε, the model assumes that no adjustment is necessary and preserves its current parameters. On the other hand, when the error exceeds ε, the model performs an update that forces the prediction to satisfy the zero loss condition (or the ε-insensitive loss), ensuring immediate correction of the violation.

The update is designed to be minimal in the sense of Euclidean distance from the previous parameter vector; that is, the model modifies its weights just enough to correct the current error, without compromising the knowledge accumulated in previous iterations. This strategy promotes robustness against noise and punctual variations, while ensuring local adaptation when the

prediction becomes inadequate.

2.14.2 Algorithmic structure

PAR is a linear regression model whose updates are defined by a constrained convex optimization problem. At each iteration t, the model receives an input instance $\mathbf{x}_t \in \mathbb{R}^d$, where d is the dimensionality of the feature space, and observes an associated real value $y_t \in \mathbb{R}$.

The model maintains a parameter vector $\mathbf{w}_t \in \mathbb{R}^d$, which defines the linear prediction function. The prediction performed at time t is given by:

$$\hat{y}_t = \mathbf{w}_t^\top \mathbf{x}_t$$

where $\hat{y}_t \in \mathbb{R}$ is the model output.

To quantify the prediction error, the model uses the ε-insensitive loss. This function disregards minor deviations and only penalizes predictions whose absolute error exceeds a threshold $\varepsilon > 0$. It is defined as:

$$\ell^\varepsilon(\mathbf{w}_t; (\mathbf{x}_t, y_t)) = \begin{cases} 0, & \text{if } |\hat{y}_t - y_t| \leq \varepsilon \\ |\hat{y}_t - y_t| - \varepsilon, & \text{otherwise} \end{cases}$$

Therefore, if the prediction $\hat{y}_t$ is within the tolerance band around the actual value y_t, no update is performed. Otherwise, the model is adjusted to correct the violation.

When a loss occurs, the weight vector is updated using the following optimization:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 \quad \text{subject to} \quad \ell^\varepsilon(\mathbf{w}; (\mathbf{x}_t, y_t)) = 0$$

This problem seeks a new weight vector $\mathbf{w}$ that corrects the prediction error while remaining as close as possible to $\mathbf{w}_t$, thereby preserving the knowledge accumulated up to time t.

The solution to this problem leads to a closed update of the form:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t \cdot \text{sign}(y_t - \hat{y}_t)\mathbf{x}_t$$

The term $\text{sign}(y_t - \hat{y}_t)$ indicates the direction of the correction (positive or negative), and the scalar τ_t determines the magnitude of the update. This coefficient is calculated by:

$$\tau_t = \frac{\ell^\varepsilon(\mathbf{w}_t; (\mathbf{x}_t, y_t))}{\|\mathbf{x}_t\|^2}$$

The value of τ_t is proportional to the excess error and inversely proportional to the length of the input instance, which avoids excessive updates on high-norm inputs.

In addition to the basic version of the algorithm, the model supports variants with regularization that introduce an aggressiveness parameter $C > 0$, which limits the impact of the update:

- **PA**: applies the exact update necessary to nullify the loss, without any limitation.
- **PA-I**: imposes a linear constraint on the magnitude of the update:

$$\tau_t = \min \left\{ C, \frac{\ell^{\varepsilon}(\mathbf{w}_t; (\mathbf{x}_t, y_t))}{\|\mathbf{x}_t\|^2} \right\}$$

- **PA-II**: introduces a quadratic penalty on the slack:

$$\tau_t = \frac{\ell^{\varepsilon}(\mathbf{w}_t; (\mathbf{x}_t, y_t))}{\|\mathbf{x}_t\|^2 + \frac{1}{2C}}$$

- **PA-III**: combines the previous approaches.

Data processing in PAR is strictly sequential. At each new instance, the model makes a prediction, computes the loss, and determines whether to update the parameters. The update occurs exclusively when the error exceeds the value of ε, which avoids unnecessary modifications to the weight vector.

Finally, the parameter ε plays a crucial role in controlling the model's sensitivity. Higher values of ε result in a more conservative model, which only reacts to significant discrepancies. Smaller values make the model more responsive to slight variations.

2.14.3 Prediction function

The prediction function of the PAR model is linear and based on the parameter vector $\mathbf{w}_t$ updated at each iteration. For an input instance $\mathbf{x}_t \in \mathbb{R}^d$, the prediction is obtained by the inner product:

$$\hat{y}_t = \mathbf{w}_t^{\top} \mathbf{x}_t$$

where $\hat{y}_t \in \mathbb{R}$ is the predicted value at time t, and $\mathbf{w}_t \in \mathbb{R}^d$ is the current weight vector.

Although the model is defined in primal space, it can be extended to dual space by using kernel functions. In this formulation, $\mathbf{w}_t$ is represented as a linear combination of previous instances, and the dot product $\mathbf{x}_t^{\top} \mathbf{x}_i$ is replaced by a kernel function $K(\mathbf{x}_t, \mathbf{x}_i)$. This extension enables the modeling of nonlinear relationships between input attributes while preserving the general update structure of the algorithm.

2.14.4 Computational complexity

The computational complexity of the PAR model is linear concerning the dimensionality of the input d, both in the update stage and in the inference stage. At each iteration, the model calculates the inner product $\mathbf{w}_t^\top \mathbf{x}_t$, evaluates the ε-insensitive loss function, and, if the loss is positive, updates the weight vector. Each of these operations involves vectors of dimension d and, therefore, has a computational cost $\mathcal{O}(d)$ per instance.

Inference consists of applying the linear prediction function:

$$\hat{y}_t = \mathbf{w}_t^\top \mathbf{x}_t$$

This calculation corresponds to a scalar product between two vectors of dimension d, with a computational cost also equal to $\mathcal{O}(d)$. This cost remains constant over time, since the model does not depend on previous instances to make predictions.

Memory consumption in the primal formulation is limited to storing the weight vector $\mathbf{w}_t \in \mathbb{R}^d$, resulting in memory complexity $\mathcal{O}(d)$. In kernelized versions, the weight vector can be implicitly represented as a linear combination of past instances, which implies an increase in memory consumption proportional to the number of updates. However, this effect is not present in the linear version of the model.

2.15 Stochastic Gradient Tree Regressor

The *Stochastic Gradient Tree Regressor* (SGT) model (GOUK; PFAHRINGER; FRANK, 2019) is a gradient-driven decision tree induction algorithm designed to operate in the context of supervised online learning on data streams. Its goal is to perform continuous value predictions in environments where data is received sequentially and the model must be updated incrementally without recurrent access to previous data.

SGT builds a single decision tree whose structure is dynamically adjusted based on stochastic gradient information extracted from the loss function used for the regression task. Each observed instance can result in the local update of a leaf by adjusting the predicted value, or in the splitting of a node if there is sufficient statistical evidence that the subdivision will lead to an expected loss improvement. The use of first- and second-order gradients enables the algorithm to select the loss function, as well as the split points and predicted values at the leaves. The flexibility of the method stems from its general formulation: the choice of the loss function determines the model's behavior, eliminating the need to redefine specific heuristics for induction or evaluation of splits.

2.15.1 Algorithmic structure

The SGT model maintains a binary or multiway tree structure, composed of internal nodes responsible for partitioning the input space and leaf nodes that store predictive scalar

values. At each time point t, upon receiving a new labeled example (x_t, y_t), the model routes x_t to a leaf of the tree. Then, local updates are performed on this leaf, which may involve updating the predicted value or splitting the node, resulting in the creation of new leaves.

The update criterion is based on minimizing an objective function composed of an approximation of the empirical loss and a regularization term. Let f_t be the prediction function of the tree at time t, and let $u : \mathcal{X} \to \mathbb{R}$ be a local modification in the tree (i.e., a possible split or adjustment in the predictive value of a leaf). The model update seeks to find the modification u that minimizes:

$$\mathcal{L}_t(u) + \Omega(u), \tag{2.12}$$

Where:

$$\mathcal{L}_t(u) = \sum_{i=r+1}^{t} \ell\left(y_i, f_t(x_i) + u(x_i)\right), \tag{2.13}$$

$$\Omega(u) = \gamma|Q_u| + \frac{\lambda}{2} \sum_{j \in Q_u} v_u(j)^2. \tag{2.14}$$

The above terms are defined as follows:

- $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ are the observed examples in the interval $[r+1, t]$;
- $\ell(y, \hat{y})$ is the loss function used (e.g., quadratic error);
- $u(x_i)$ represents the local modification applied to the current prediction $f_t(x_i)$;
- Q_u is the set of identifiers of the new leaves that would be created by u;
- $v_u(j)$ is the predicted value assigned to the new leaf j;
- $\gamma > 0$ penalizes the structural complexity of the tree (number of leaves added);
- $\lambda > 0$ penalizes the magnitude of the predictive values.

To enable real-time optimization, a second-order Taylor expansion of the loss function is used. It is:

$$g_i = \frac{\partial \ell(y_i, f_t(x_i))}{\partial f_t(x_i)}, \tag{2.15}$$

$$h_i = \frac{\partial^2 \ell(y_i, f_t(x_i))}{\partial f_t(x_i)^2}, \tag{2.16}$$

Then the approximate loss is given by:

$$\mathcal{L}_t(u) \approx \sum_{i=r+1}^{t} \left[g_i u(x_i) + \frac{1}{2} h_i u(x_i)^2 \right], \tag{2.17}$$

and the expected reduction in loss when applying u is:

$$\Delta\mathcal{L}_t(u) = \sum_{i=r+1}^{t} \left[g_i u(x_i) + \frac{1}{2} h_i u(x_i)^2 \right]. \tag{2.18}$$

For each candidate split, define $u(x)$ as:

$$u(x) = \begin{cases} v_u(q_u(x)), & \text{if } x \in \text{domain of the leaf being split,} \\ 0, & \text{otherwise,} \end{cases} \tag{2.19}$$

where $q_u(x) \in Q_u$ maps x to the identifier of the corresponding new leaf.

Let $I_j = \{i \mid q_u(x_i) = j\}$ be the set of indices of the instances routed to the new leaf j. The estimated loss reduction becomes:

$$\Delta\mathcal{L}_t(u) = \sum_{j \in Q_u} \left[\left(\sum_{i \in I_j} g_i \right) v_u(j) + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) v_u(j)^2 \right]. \tag{2.20}$$

By adding the regularization term, we obtain the function to be minimized for each new leaf j:

$$\left(\sum_{i \in I_j} g_i \right) v_u(j) + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) v_u(j)^2. \tag{2.21}$$

The derivative of this expression concerning $v_u(j)$, set equal to zero, provides the optimal value for the new leaf:

$$v_u^*(j) = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}. \tag{2.22}$$

This value is used to calculate the expected gain attributed to the new leaf if the split is applied. The algorithm evaluates all possible splits, based on discrete or discretized attributes, and selects the one that maximizes the expected reduction of the regularized loss.

The split is applied only if it is statistically significant, as determined by a hypothesis t test. Otherwise, the tree structure is maintained, and only the value of the current leaf can be updated based on the observed gradients.

2.15.2 Prediction function

The model prediction is performed based on the value stored in the leaf reached by the input instance. Let $f(x)$ be the function represented by the tree at the current instant, and let $x \in \mathbb{R}^d$ be an instance. The instance traverses the tree following tests on internal nodes until it reaches a leaf l, whose predictive value $v_l \in \mathbb{R}$ determines the model output:

$$\hat{y} = f(x) = v_l \quad \text{such that } x \in l. \tag{2.23}$$

When a split is performed on a leaf, that leaf is replaced by an internal node, and two or more new leaves are created. For each new leaf j, the predictive value v_j is defined based on the minimization of the regularized loss associated with the instances that would be routed to that leaf. We assume a second-order Taylor expansion of the loss function $\ell(y, \hat{y})$ around the current prediction $f(x_i)$:

$$\Delta\mathcal{L} \approx \sum_i \left[g_i \cdot u(x_i) + \frac{1}{2} h_i \cdot u(x_i)^2 \right], \tag{2.24}$$

Where:

$$\begin{aligned} g_i &= \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)} \quad \text{(gradient)}, \\ h_i &= \frac{\partial^2 \ell(y_i, f(x_i))}{\partial f(x_i)^2} \quad \text{(Hessian)}, \\ u(x_i) &= v_j, \quad \text{if } x_i\text{is routed to new leaf } j. \end{aligned}$$

For each new leaf j, define I_j as the set of instances that would be routed to it. The optimal value v_j^* that minimizes the local regularized loss is:

$$v_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{2.25}$$

where λ is the regularization parameter. This value is assigned to the new leaf and is used for future predictions.

Even as the tree evolves, the predictions remain locally stable. Because updates are based on accumulated gradients and applied locally, the model maintains consistency with previous estimates, gradually adapting to new regions of the input space as more data is observed.

2.15.3 Computational Complexity

The model update time is proportional to the depth of the tree, since the path to the leaf and the update of the statistics involve only the nodes visited along this path. After a fixed

number of examples observed in a leaf, the evaluation of possible splits is performed, the cost of which depends on the number of attributes and the candidate partitions for each attribute. Since this verification is performed periodically, the average cost per instance remains constant under the steady-state regime of the tree.

The inference time is also proportional to the depth of the tree, since it consists only of routing the input instance to a leaf, followed by reading a stored scalar value. This procedure does not involve additional calculations or recursive calls.

Memory usage is controlled by the number of leaves and the statistics stored in each one. Each leaf maintains sums of gradients, Hessians, counters, and, eventually, estimates of variance and covariance, whose requirements are constant per attribute. Thus, the total memory consumption grows linearly with the number of leaves in the tree and with the dimensionality of the data. Because the structure is built incrementally and supervised by splitting criteria based on statistical significance, tree growth tends to be regulated, even under extensive flows.

2.16 Streaming Random Patches Regressor

The *Streaming Random Patches Regressor* (SRP) model (GOMES; READ; BIFET, 2019; GOMES et al., 2020) is an ensemble-based incremental machine learning technique that explores diversity induction through the combination of random subsets of instances and attributes, referred to as random patches. The construction of each base model involves randomly sampling the input stream in both dimensions, which aims to promote structural diversity among the ensemble members. This approach increases the heterogeneity of the models and, consequently, enhances the ensemble's ability to represent multiple regions of the input space.

Each SRP base model is incrementally trained and maintained on an active set of predictors. The composition of this set is dynamic and regulated by replacement mechanisms, which evaluate the individual performance of the models over time. Replacement can be triggered by window-based strategies (fixed, random, or adaptive) or by explicit change detectors, such as ADWIN. When a potential performance decline is identified, a new model is trained in the background using a new data patch and may eventually replace the corresponding active model.

2.16.1 Algorithmic structure

The generation of base models in SRP involves the creation of M independent predictors $\{h_1, h_2, \ldots, h_M\}$, each trained online on a random subset of attributes and a random weighting of the instances observed in the stream. Let $\mathbf{x} \in \mathbb{R}^d$ be an input instance. For each model h_i, a subspace $\mathcal{A}_i \subseteq \{1, 2, \ldots, d\}$ is defined, with $|\mathcal{A}_i| = m < d$, selected randomly and fixed throughout the lifetime of the model. The input $\mathbf{x}_{\mathcal{A}_i}$ represents the projection of $\mathbf{x}$ onto the subspace $\mathcal{A}_i$.

For each new instance $(\mathbf{x}_t, y_t)$ of the stream, a multiplicity value $w_{i,t} \sim \text{Poisson}(\lambda)$ is sampled for each model h_i. This value determines how many times the instance will be used in the incremental update of the model, simulating resampling with replacement. The training of each base model is then performed based on the pairs $(\mathbf{x}_{t,\mathcal{A}_i}, y_t)$, weighted by $w_{i,t}$. Thus, the training process can be formalized as:

$$h_i \leftarrow \text{Update}\left(h_i, \mathbf{x}_{t,\mathcal{A}_i}, y_t, w_{i,t}\right), \quad \forall i \in \{1, \ldots, M\}.$$

Ensemble maintenance involves rules for replacing base models. Each h_i is monitored by a change detector that evaluates its error sequence $\{\ell_{i,t}\}$, where $\ell_{i,t} = \mathcal{L}(h_i(\mathbf{x}_{t,\mathcal{A}_i}), y_t)$, with $\mathcal{L}(\cdot)$ representing the loss function, such as the squared error. Upon alert (e.g., via ADWIN), a background model $\tilde{h}_i$ is trained, with a new subspace $\tilde{\mathcal{A}}_i$ and incrementally updated until a replacement criterion is satisfied when this occurs, $h_i \leftarrow \tilde{h}_i$, keeping the ensemble size constant.

In addition to change detection-based strategies, SRP-Reg can also adopt window-based replacement schemes. Let $W_i = [t_s^{(i)}, t_r^{(i)}]$ be the lifetime window of h_i; the boundaries of this window can be fixed, random, or adaptive. In any case, replacement ensures that the active models reflect, over time, different periods and subspaces of the data stream.

2.16.2 Prediction Function

Let $\mathcal{L} = \{h_1, h_2, \ldots, h_M\}$ be the set of active models, where each model h_i is trained on a specific random patch, that is, a random subset of instances and attributes of the stream. Given an input instance $\mathbf{x} \in \mathbb{R}^d$, each model h_i performs its prediction considering only the projection of $\mathbf{x}$ onto the subspace $\mathcal{A}_i \subseteq \{1, \ldots, d\}$ of attributes relevant to that model. The individually predicted output by h_i is denoted by:

$$\hat{y}_i = h_i(\mathbf{x}_{\mathcal{A}_i}).$$

The final ensemble prediction, $\hat{y}$, is obtained by aggregating the individual predictions. In SRP, two aggregation functions are commonly used:

- **Arithmetic mean:**
$$\hat{y} = \frac{1}{M} \sum_{i=1}^{M} \hat{y}_i.$$
- **Median:**
$$\hat{y} = \text{median}(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M).$$

The choice of the aggregation function Φ affects the sensitivity of the ensemble to divergent predictions. The arithmetic mean is sensitive to extreme values and is suitable when model errors are approximately symmetric and do not contain significant outliers. On the

other hand, the median offers greater statistical robustness against atypical predictions, being particularly useful in situations where newly initialized or outdated models produce extreme values.

The diversity among the active models is a crucial factor in determining the ensemble's performance. Since each h_i is trained on different regions of the feature space and different subsets of instances, individual errors tend to be weakly correlated. This property is essential for aggregation to reduce the total variance of the system, as described in the bias-variance decomposition for ensembles:

$$\text{Var}[\hat{y}] \approx \frac{1}{M}\text{Var}[\hat{y}_i] + \left(1 - \frac{1}{M}\right)\text{Cov}[\hat{y}_i, \hat{y}_j], \quad i \neq j.$$

By minimizing the covariance between models (*via* diversity induced by random patches), the ensemble achieves better generalization by reducing the variance without significantly increasing the bias.

2.16.3 Computational Complexity

The computational complexity analysis of SRP considers the cost per instance during the data flow, both for inference and for updating the base models. Suppose that the ensemble maintains M active models, each operating on a subspace with $m < d$ attributes. Let T be the average processing time of a base model per instance, in its specific subspace.

During the inference phase, each active model makes an individual prediction based on the projection of the input $\mathbf{x}_t \in \mathbb{R}^d$ onto the subspace $\mathcal{A}_i \subseteq \{1, \ldots, d\}$. The total inference cost per instance is proportional to the number of models and the complexity of the individual predictor:

$$\mathcal{O}_{\text{inf}} = M \cdot T_{\text{inf}}(m),$$

where $T_{\text{inf}}(m)$ is the inference cost of a base model with m attributes.

In the update phase, each model h_i receives the instance weighted by a value $w_{i,t} \sim \text{Poisson}(\lambda)$. The expected number of updates is λ per model. Thus, the total update cost per instance is given by:

$$\mathcal{O}_{\text{upd}} = M \cdot \lambda \cdot T_{\text{upd}}(m),$$

Where $T_{\text{upd}}(m)$ is the incremental update cost of a base model with m attributes.

The replacement strategy adopted directly affects the cumulative computational cost. Strategies with fixed windows result in predictable periodic replacements. Strategies based on random windows or change detectors, such as ADWIN, introduce variable overhead associated

with performance monitoring and maintaining the background model. Assuming that R is the average replacement rate per model (number of resets per unit time), and that training a background model involves τ instances, the amortized cost per instance of replacement is:

$$\mathcal{O}_{\text{swap}} = M \cdot R \cdot \tau \cdot T_{\text{upd}}(m)/N,$$

Where N is the total number of instances processed.

Regarding memory usage, each base model stores local statistics about its feature subspace and, depending on the implementation, auxiliary structures for computing splits, in the case of trees, or internal estimates. The total memory usage is therefore:

$$\mathcal{O}_{\text{mem}} = M \cdot S(m),$$

Where $S(m)$ represents the space occupied by a single base model, dependent on m, the average depth of the structure, and the amount of statistics maintained.

2.17 Chapter Summary

Throughout this chapter, the main concepts and foundations that support the approach proposed in this dissertation were presented. Initially, the architecture and operational challenges of the 5G core network (5GC) were discussed, with emphasis on the importance of scalable and adaptive solutions to meet emerging use cases and the increasing demands of mobile traffic.

The chapter also addressed the characteristics of scalability systems, distinguishing between horizontal and vertical scaling mechanisms, and highlighting the challenges posed by under-provisioning, over-provisioning, and oscillation in resource allocation. In particular, proactive scalability approaches were emphasized for their potential to anticipate demand variations. However, it was noted that their efficiency can be compromised by the phenomenon of *concept drift*, which occurs when the statistical properties of traffic evolve over time, reducing the predictive accuracy of conventional machine learning models.

In this context, the chapter introduced the paradigm of online learning as a promising solution to deal with non-stationary environments. Unlike offline approaches, online learning models are able to update their internal parameters continuously, adapting to traffic fluctuations and maintaining predictive accuracy even in the presence of sudden or gradual changes. To contextualize this approach, the main typologies of concept drift were described, including sudden, gradual, and recurrent drifts, as well as their impacts on predictive models.

Finally, the set of online regression models used in this dissertation was presented, together with the adopted hyperparameters. To provide a clearer overview, Table 1 categorizes the models according to their underlying principles. This organization aims to situate each

method in relation to its family of techniques, reinforcing the theoretical basis that underpins the experiments of this work.

Table 1 – Summary of Regression Models by Category

Category	**Models**
Decision Trees	SRP, HTR, SGT
Ensemble Methods	ARF, AMFR, BaR, EWA, OXT, SRP
Linear Models	LiR, BLR, PAR
Neighbor-Based Models	KNN

Source: The author.

Based on this theoretical foundation, the next chapter presents the proposal developed in this dissertation, detailing its architecture, the components involved, and how the concepts explored here will be applied to address the problem of proactive scalability in the context of the 5GC.

3 RELATED WORK

The increasing volume of traffic and the diversity of applications in fifth-generation mobile networks have driven the adoption of intelligent strategies for the dynamic management of network resources. In particular, the proactive scalability of network core functions has become a central theme in the search for solutions that ensure quality of service and operational efficiency.

Several machine learning-based approaches have been explored to predict demand variations and anticipate the allocation of computational resources. However, most existing studies focus on offline learning techniques, which rely on previously collected and processed historical data. Although efficient in stationary scenarios, these methods present limitations when applied in highly dynamic environments, such as 5G networks.

Recently, strategies based on online learning have gained prominence as a promising alternative to deal with the non-stationary nature of modern telecommunications networks. This class of algorithms is capable of continuously updating its parameters based on new data without the need to reprocess large historical volumes.

Given this scenario, this section presents a review of the main works related to the proactive scheduling of network functions in the context of 5G Core (5GC), with a special focus on proactive scalability solutions. The most relevant approaches, their characteristics, limitations, and recent advances in applying online learning techniques to this problem will be discussed.

3.1 Description of Related Works

Among the reactive approaches for scaling the 5G core, (ALAWE et al., 2018a) proposes a method based on control theory. Although the proposed method decreases the number of unfilled requests compared to other reactive methods, fluctuations and delays in decision-making have resulted in significant request rejections in their tests. Similarly, (CHOUMAN; MANIAS; SHAMI, 2023) has a reactive solution that minimizes the number of instances based on service classes. Although it produces great solutions, the proposal may suffer from provisioning delay, as it only reacts to the observed demand. Additionally, the proposal's dependence on parameters to be adjusted compromises its efficiency and scalability.

On the other hand, (ALAWE et al., 2018d) investigated the use of *Deep Neural Networks* (DNN) and *Long Short-Term Memory* (LSTM) models to predict the demand for requests. The task is addressed as a classification problem, subdividing traffic to cargo lanes, where each range is associated with the number of AMF instances required to endure it.

To evaluate the model, the authors made a simulator based on discrete events. To estimate

the number of requests that an AMF instance can bear, they used the OpenAirInterface (OAI). Through the simulation, they determined that an AMF instance can handle up to 20 ATTACH requests per second before latency increases significantly and connections are refused. This value was used as a reference in the simulator to define the processing capacity of each AMF instance in the simulations.

The results demonstrated that the LSTM-based model surpassed both the approach with deep neural networks (DNN) and the traditional fixed-based method. The LSTM achieved approximately 90% accuracy in forecasting the load class, compared to about 80% for the DNN. In simulations, the proactive mechanism with LSTM was able to anticipate increased demand and previously allocate the necessary AMF instances, reducing response time and avoiding system overload. Compared to the Threshold approach, the LSTM model significantly reduced the number of locked requests, as well as eliminating the delays associated with late creation of new instances.

The works of (KURANAGE et al., 2022) and (KURANAGE et al., 2023) investigate the use of *Gated Recurrent Unit* (GRU) for proactive scalability of the AMF function in Kubernetes. The model is used to predict the future use of CPU based on the number of requests, in order to anticipate scalability needs.

To perform the experiments, the AMF function behavior has been simulated through a contained web server, where HTTP requests follow the traffic standards of a telecommunications provider. The primary objective of this approach is to maintain the QoS during scalability events, thereby reducing the response time of services and minimizing replica operating time.

The proposal is compared to the traditional Kubernetes scalability method, the *Horizontal Pod Autoscaler* (HPA), which acts reactively based on fixed thresholds of CPU usage. In experiments, the solution demonstrated higher results, achieving lower response times during load peaks and a reduction in replica operating time when compared to HPA configured with different thresholds.

In the same sense, the work of (PASSAS et al., 2022) investigates the scalability of the AMF function in Kubernetes environments, to maximize the admission rate of users and minimize the waste of computational resources. In fact, unlike previous work, the authors implement a functional instance of the 5G nucleus using the Open5gcore framework, which allows for a non-trading-only collection. Infrastructure, such as CPU usage and memory, as well as network operational indicators, including the number of admitted users and the Drop Rate disposal rate.

For the prediction task, the Wavenet model is used, trained as a binary classifier, whose output indicates whether the system should perform an AMF function scale-out or scale-in operation. The results show that the model achieved 97% accuracy in predicting scaling actions. Resulting in the admission of all 18,000 simulated users, surpassing both the version without

staggering, which admitted to 14,000, and the CPU-based HPA scale that reached 17,500 users. Additionally, the system achieves a 50% reduction in the average number of AMF instances, indicating more efficient resource allocation.

In the paper (ALAWE et al., 2018c), the authors investigate the use of neural networks such as CNN, LSTM, and a combined architecture (CNN-LSTM) for traffic prediction in mobile networks. The study stands out for considering the problem from the perspective of conceptual changes, evaluating the models' ability to handle different load amplitudes, such as scenarios with regular traffic, duplicated traffic, and subsequent return to the original pattern. To address the models' limitations in the face of dynamic network evolution, the authors propose a periodic retraining mechanism. To make this process more efficient, they introduce a technique that combines K-means clustering with Monte Carlo sampling, accelerating the training phase and thus enabling frequent model updates.

3.2 Challenges in Existing Approaches

Threshold-based dimension strategies such as HPA, (ALAWE et al., 2018a), and (CHOUMAN; MANIAS; SHAMI, 2023), are widely used in contained environments such as Kubernetes, especially for stateless services. However, in these approaches, scalability occurs only after the application has signs of overload, which can result in temporary degradation of service as new instances are initiated and become fully operational. In addition, these techniques are subject to the phenomenon of oscillation, whose negative impact on stability and application performance becomes particularly evident in scenarios with short-term load fluctuations. Thus, this type of strategy is inappropriate for new cases of mobile networks, in which they require almost real-time responses, high reliability, low latency, and continuous availability even in the face of highly dynamic and heterogeneous loads.

On the other hand, works such as (KURANAGE et al., 2022) and (KURANAGE et al., 2023) attempt to overcome these limitations by proposing proactive scalability methods. However, by using metrics such as CPU and memory to guide scalability, these methods end up limiting themselves to increasing instances one at a time. In scenarios of abrupt demand growth, this approach can be slow and inefficient, resulting in system response delays and potential degradations in service quality.

The works (ALAWE et al., 2018d), (PASSAS et al., 2022), and (CHOUMAN; MANIAS; SHAMI, 2023) address the use of proactive methods that utilize classes, which associate a given number of requests with a required number of glsAMF instances to serve them. Although this approach allows the activation of multiple instances simultaneously, it has significant limitations. Because it relies on previously known load levels, its effectiveness is reduced in scenarios such as mobile networks, where traffic is constantly growing. Since the number of devices connected to the network continues to increase, these methods require frequent data reprocessing and

model retraining, as the predefined classes no longer adequately represent the actual volume and variability of requests.

The work of (ALAWE et al., 2018c) presents a relevant contribution when investigating traffic prediction models from the perspective of network evolution. The proposal introduces a consistent scalability metric in the face of these changes, as well as a solution to dealing with the temporal evolution of the data. However, the research also has important limitations in addressing contemporary challenges of mobile networks. Although re-training use allows the model to adapt to new network behaviors, this approach can be insufficient in scenarios characterized by frequent, abrupt, or unpredictable changes.

This limitation arises because emerging mobile networks are characterized by increasing densification, device diversity, and the constant emergence of new use cases, which makes traffic behavior much less predictable. This scenario imposes significant pressure on predictive models that merely react to change, requiring constant re-training. It is noteworthy that these scenarios may not occur gradually, but rather suddenly and dynamically, requiring more agile and continuous adaptation mechanisms than those offered by approaches based exclusively on reactive re-reaction. In addition, the computational cost and latency associated with frequent model updates can compromise its practical viability, especially in sensitive applications such as autonomous vehicles, industrial systems, critical infrastructure, or remote surgeries, where decisions need to be made almost in real-time and with a high degree of reliability.

Table 2 compares the works presented to the proposal of this article. In general, the proactive approach is the most common because it has advantages over the reactive approach. The metric used in decision-making varies among articles, and it is common to convert the prediction problem of a continuous variable into a classification problem in terms of traffic levels, which makes it challenging to adapt to growing traffic. However, the main limitation of existing work is its low ability to continually adapt to changes in the network without the use long retraining.

In this sense, this work aims to fill these gaps by proposing an online scalability model capable of adapting to dynamic changes in networks, thereby maintaining the efficiency of predictions even in constantly evolving scenarios. To achieve this, the performance of various online learning approaches is investigated to identify the most promising models for environments where temporal variability and the need for rapid response are prevalent.

Table 2 – Comparison between the reviewed works and the method proposed in this paper.

Article	Method	Scalability metric	Adaptable to new concepts
(ALAWE et al., 2018a)	Reactive	Load on AMF	No
(ALAWE et al., 2018c)	Proactive	Load on AMF	Yes (Offline)
(ALAWE et al., 2018d)	Proactive	Load classes on AMF	No
(KURANAGE et al., 2022)	Proactive	CPU usage	No
(PASSAS et al., 2022)	Proactive	Load classes on AMF	No
(KURANAGE et al., 2023)	Proactive	CPU usage	No
(CHOUMAN; MANIAS; SHAMI, 2023)	Reactive	Load classes on AMF	No
Proposed Method	Proactive	Load on AMF	Yes (Online)

Source: The author.

3.3 Chapter Summary

This chapter presents the main works related to the scalability problem in mobile networks, addressing both reactive and proactive strategies. The analysis has enabled us to identify fundamental aspects that should be considered in developing practical solutions to this problem, such as the importance of a careful choice of scalability parameters and the need for strategies that are resilient to continuous infrastructure transformations and evolving network use patterns.

In addition, the study highlighted significant limitations in existing approaches, particularly with regard to their ability to adapt to non-stationary scenarios. Many of these solutions assume stable conditions or depend on periodic update cycles, which compromises their effectiveness in constantly transforming scenarios, marked by greater densification, a variety of devices, and the evolution of traffic profiles. Such limitations underscore the need for more flexible mechanisms that can continually adapt to changes in network conditions.

In this context, this chapter also introduces the proposal of this work, which seeks to overcome the weaknesses observed through an online learning-based approach capable of self-adaptation to environmental variations and maintaining scalability efficiency in next-generation mobile networks.

4 METHOD

Scalability is a fundamental requirement for ensuring the efficient and reliable operation of mobile networks. Increased traffic, especially in critical control plane functions, can lead to overload, increased latency, and, in extreme cases, request rejection due to a lack of adequately allocated resources (ALAWE et al., 2018b). Effective scalability management allows network resources to be allocated proportionally to demand, ensuring consistent performance even in the face of sudden traffic spikes. This capability prevents QoS degradations such as high delays, packet loss, or connection failures (KURANAGE et al., 2023). At the same time, it reduces the waste of computing resources by avoiding unnecessary oversizing during periods of low utilization (ALAWE et al., 2018c).

Furthermore, the demands imposed by next-generation mobile networks present even more complex challenges for telecommunications infrastructure. To effectively realize promises such as sub-millisecond latency, high reliability, and support for up to 1 million devices per square kilometer, a profound rethink of the core and transport network architecture is necessary. In this context, the scalability of these layers becomes a determining factor in ensuring that network performance remains stable even under extreme load scenarios (Cisco Systems, Inc.; PLDT Inc., 2020).

In critical applications, such as those used in public services, autonomous transportation, e-health, and massive IoT, scalability failures can compromise essential requirements, including low latency, high availability, and reliability. Thus, scalability is not only a technical requirement, but a functional prerequisite for the viability of many of 5G's strategic use cases (Ericsson, 2023).

In this sense, studies such as (6G-PPP Architecture Working Group, 2022) highlight that scalability, along with resilience, will be even more central to future network architectures. To handle traffic volatility and the diversity of emerging service requirements, the network must be organized in an adaptive, automated, and service-oriented manner. However, traditional proactive models assume a stationary environment for their whole operation. The telecommunications environment, in turn, is known as non-stationary (LIU et al., 2023), characterizing a scenario in which traffic patterns and relationships between input and output variables become continuous. In this context, online models become more suitable as they are capable of adapting increasingly to changes in data behavior without the need for retraining (HOVAKIMYAN; BRAVO, 2024).

Because of this characteristic, online models can maintain consistent performance levels over time, even in the face of unpredictable changes in network conditions. This continuous adaptation is essential for ensuring quality of service (QoS), as it enables the system to respond quickly to environmental changes, significantly reducing the latency between the occurrence of a

change and the need for corrective scaling action. Furthermore, online models are specifically designed to handle continuous data streams (streaming), making them highly suitable for systems that require real-time processing (HOI et al., 2021). As a result, they feature low operational latency, meaning they can produce outputs almost instantaneously after the arrival of new data, ensuring the fast and consistent responses required by next-generation mobile networks.

4.1 Experimental Environment

Defining an efficient scalability architecture depends directly on the choice of metrics used to guide resource allocation decisions. Metrics such as CPU and memory usage, although widely adopted in containerized environments, have significant limitations. In systems subject to real traffic, abrupt bursts of requests are common, capable of drastically increasing the load in very short time windows. This behavior is widely recognized in 5G architectures, which face explosive traffic growth and require scaling based on peak load to avoid congestion in the network core. (Cisco Systems, Inc.; PLDT Inc., 2020)

In this sense, because there is no deterministic relationship between the utilization percentages of these resources and the number of additional instances needed to absorb future load, corrective actions based on these metrics tend to occur late or underestimate. As a result, new resource allocation occurs slowly and incrementally, requiring multiple scaling steps until the system achieves stability. This behavior compromises the system's ability to respond quickly and accurately to sudden fluctuations in demand, thereby increasing the risk of service quality degradation during peak periods.

On the other hand, approaches that use predefined load ranges face difficulties in adapting to progressive and continuous increases in demand. Over time, the classes no longer accurately represent the network's reality, necessitating frequent reprocessing and continuous reconfiguration, which in turn makes the system more complex and less adaptable.

Thus, traffic load, expressed as a request rate per unit of time, represents a more accurate alternative that is sensitive to the real dynamics of the environment. Unlike indirect metrics, request volume immediately reflects the impact of user activity on network components, allowing for more accurate inferences about the need to allocate new instances. This approach enables a more agile response to sudden changes in traffic by predicting the exact number of instances needed to serve them. Furthermore, using traffic load as a central metric aligns with the guidelines for scalable 5G architecture, as evidenced in the PLDT and Cisco study, which highlights the need for peak load-based planning to mitigate congestion and ensure service continuity in the network core (Cisco Systems, Inc.; PLDT Inc., 2020).

Similarly, for a scalable system to be effective, its output must be formulated to allow for immediate response, even in rapid growth scenarios that require multiple instances. Formulations based on binary decisions, such as "scale up" or "scale down," are limited in this context, as they

require multiple decision cycles to accommodate substantial load increases, resulting in slow and inefficient reactions.

In contrast, by having as an output a quantitative estimate of demand, such as the expected rate of requests per second, the system becomes more interpretable, flexible, and modular. By decoupling prediction and decision-making, this approach enables, for example, the simultaneous activation of multiple instances in advance, reducing the impact on startup time. Furthermore, it avoids unnecessary fluctuations in the number of instances by allowing the use of more intelligent tolerance margins, rather than relying on binary reactions to momentary variations. It also becomes possible to dynamically adapt the mapping between load and instances based on real-world observations, continually refining the allocation policy. Another significant benefit is the ability to perform hypothetical simulations based on the forecasts, facilitating the evaluation of different strategies without interfering with the production environment. Finally, explicit load forecasting provides greater transparency and auditability, enabling decisions to be justified based on quantitative metrics and facilitating the validation of the system as a whole.

Regarding decision-making, this can adopt vertical or horizontal scalability strategies. Vertical scalability aims to expand the computational resources of an existing instance, such as CPU or memory. Traditionally, this required successive reboots or reconfigurations, thereby increasing operational complexity and response latency. However, recent advances in cloud-native platforms, such as Kubernetes, introduced the In-place Pod Resize mechanism (v1.33, beta by default), which allows CPU and memory adjustments in running containers. Depending on the resize policy, CPU changes can often be applied without restarting the container, while memory changes may still require a restart. This evolution reduces the disruption typically associated with vertical scaling and increases its applicability in latency-sensitive environments. Horizontal scalability, on the other hand, consists of replicating instances in parallel and is more suitable for distributed and dynamic environments. This approach prioritizes parallelism, fault tolerance, and component isolation, thereby reducing the system's overall impact from individual failures. Furthermore, its alignment with microservices architectures and orchestrated platforms, such as Kubernetes, makes it especially effective for cloud-native applications. The reduced instancing complexity and the ability to create multiple replicas simultaneously allow the system to react more quickly to sudden traffic fluctuations, especially when combined with proactive load forecasting mechanisms (DO et al., 2025; The Kubernetes Authors, 2025). This conception is in line with the 5G Core (5GC) architecture, which was designed with native support for horizontal scalability, as highlighted by Ericsson, by emphasizing the use of cloud-native principles to promote elasticity, resilience and separation between the control and user planes, fundamental elements to ensure the adaptability and robustness of the network infrastructure (Ericsson, 2023).

To evaluate the proposal, the Access and Mobility Management Function (AMF) was selected as the subject of study. The AMF was chosen due to its central role in the 5G Core architecture's control plane, which is responsible for critical procedures such as user registration,

mobility management, and authentication. Several factors justify the selection of this function as a case study:

1. The AMF is sensitive to overload, especially in scenarios involving massive simultaneous access, such as large-scale events or high-density population areas;
2. AMF performance directly impacts the latency of control procedures and the success rate of connections, and is therefore crucial for the user experience;
3. In addition, the AMF has intensive signage behavior, which makes it an excellent candidate to assess the effectiveness of real -time -oriented scalability strategies.

The following outlines the application of the discussed architecture. The subsection presents a conceptual view of a scalable system, highlighting the cargo balancing mechanisms, instance replication, and different scaling strategies, reactive, proactive, and online learning-based, that underlie the proposal of this work. This approach illustrates that an adaptive load forecast can be integrated into the decision-making process.

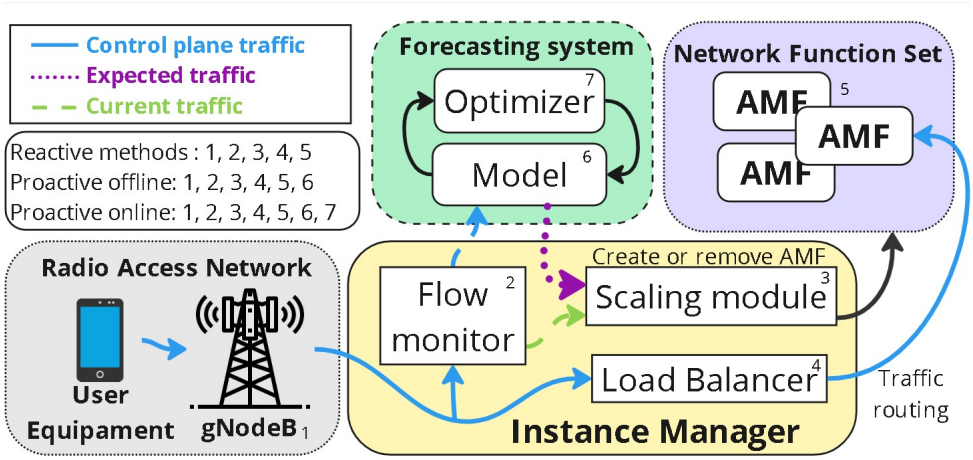
4.2 Architecture and Components of the Core Scalability System

Figure 3 shows a conceptual view of a scalability system for the 5GC. Focusing on the control plane, the figure indicates that control traffic is sent from the *User Equipment* (UE) to the 5GC, through the base station (gNodeB). At its core, the traffic is received by the AMF, which is responsible for establishing UE sessions and serves as the gateway to other 5GC functions. In a scalable 5GC, AMF replicas are executed on a network function set, and the traffic is distributed between replicas by a Load Balancer. Furthermore, the Scaling Module scales instances horizontally according to the traffic demand.

In reactive scaling strategies, the Scaling Module makes scalability decisions based on the current traffic demand directly, employing thresholds for creating and removing instances based on the traffic volume. This strategy is straightforward and easily adaptable to various scenarios. However, it has limitations regarding the response time to new demands (AKSHLLEY; CARVALHO; LOPES, 2022), caused by the gap between the decision-making and the actual creation and/or removal of instances. This delay can result in periods of undersizing, causing service degradation. It can lead to oscillations, i.e., an instance being recreated shortly after being removed (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014), making it difficult to use in highly dynamic scenarios.

Proactive methods, on the other hand, use predictive models to anticipate demand. Thus, the sizing module makes its decisions based on information from a prediction model, usually a

Figure 3 – Conceptual view of a scalable Mobile Core.



Source: The author.

machine learning model that, from current traffic data collected by the monitoring agent, predicts future traffic demands. These methods mitigate the delays of reactive methods by anticipating decision-making and the creation of new instances.

According to the learning strategy they use, proactive methods can be classified into methods online and methods offline. In offline methods, models are trained and later implemented in the environment to perform the predictions (HOI et al., 2021). However, telecommunications networks are non-stationary environments (MANIAS; CHOUMAN; SHAMI, 2023), i.e., traffic presents conceptual deviations as new network applications and technologies are introduced. This behavior leads offiline learning models to experience degradation in the quality of their predictions, becoming less effective as the network evolves.

In contrast, online learning models stand out for their ability to continuously adapt to changes in data through the incremental learning process (PÉREZ-SÁNCHEZ; FONTENLA-ROMERO; GUIJARRO-BERDIÑAS, 2018). In this case, an Optimizer observes the output of the Model and continuously improves it with each new predicted instance. Due to this characteristic, online models have recently been used to solve problems in telecommunications scenarios. An example of this is the research by (SILVA et al., 2024), which obtained promising results when investigating the use of online learning in resource scaling in edge computing environments.

In this way, the online learning model, incorporated into the forecasting system, operates continuously and adaptively within the scalable architecture. After the Flow Monitor monitors the current traffic flow, the data is forwarded to the Model, which predicts future traffic demand. With each new data input from the operational environment, the online Model is immediately updated by the Optimizer, which adjusts its parameters based on incremental learning algorithms, such as stochastic gradient descent.

This Optimizer is the essential component that enables the system to learn in real-time, giving the Model the ability to dynamically adapt to changing traffic patterns. Thus, as the

network evolves, for example, with the introduction of new services, seasonal behaviors, or changes in user density, the online Model follows these variations, adjusting its predictions without the need for complete reprocessing or retraining, as occurs with offline methods.

Thus, the scalability module enables proactive decisions based on forecasts that accurately reflect the current and emerging state of the network, anticipating overload scenarios and preventing undersizing. This approach makes the system more robust, as it not only reduces fluctuations in provisioning but also monitors the network's natural evolution, ensuring more efficient resource allocation given the non-stationary nature of telecommunications networks.

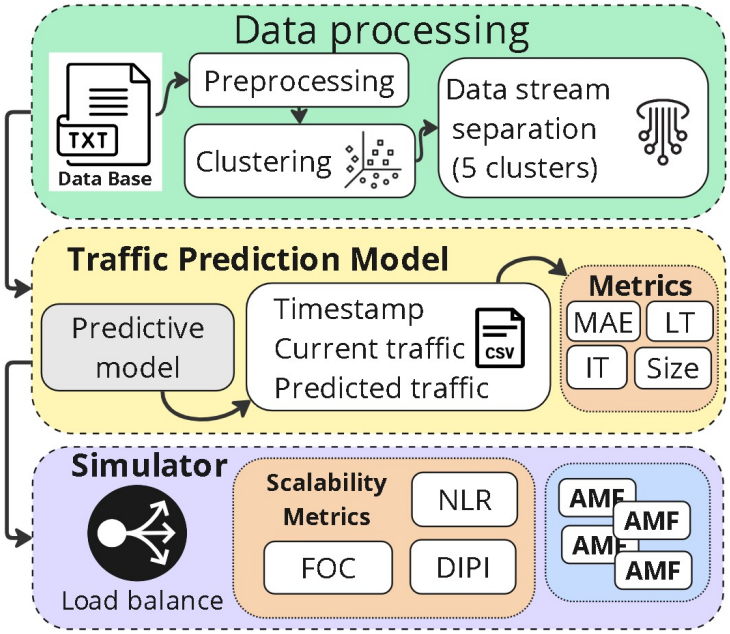
In this sense, this work proposes and evaluates the adoption of online learning models for the scalability of network functions by incorporating them into the forecasting module of a proactive scheduling system. The objective is to enable adaptive forecasting of traffic demand, allowing for more accurate scalability decisions that align with seasonal fluctuations and the continuous evolution of the network. With this, we aim to develop a more resilient and responsive system capable of operating efficiently in dynamic and non-stationary environments, such as those found at the core network.

4.3 Evaluation

Figure 4 illustrates the employed method to evaluate this work. The first step, described in Section 4.4, involves processing a real mobile network traffic dataset, which ensures that the probability distributions learned by the models are similar to those found in practice while, at the same time, due to the data structure, enabling the introduction of concept drifts. The next step (Section 4.5) consists of selecting, training, and evaluating the online learning models for traffic prediction. Finally, Section 4.6 describes the construction of a simulator to quantify the impact of the models' predictions on the scalability of the AMF function through high-level operational indicators. All experiments, code and processed data are publicly available for reproducibility and future research[1].

[1] Available at: <https://github.com/lasseufpa/5g-online-scaling-exp>

Figure 4 – Evaluation method used in this work.



Source: The author.

4.4 Dataset and Processing

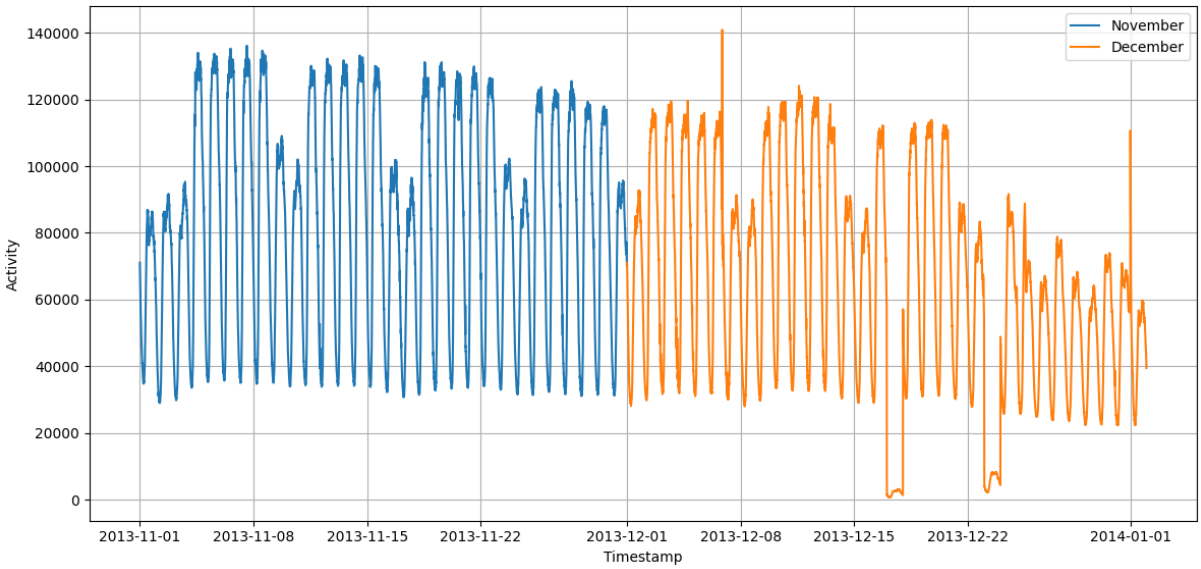
The evaluation utilizes telecommunications activity data from Telecom Italia's mobile network in and around Milan (BARLACCHI et al., 2015), which was collected through the use of *Call Detail Records* (CDRs)[2]. from November 1, 2013, to January 1, 2014[3]. These records are known in the mobile network literature and have been used in similar works (ALAWE et al., 2018c; ZHU; SUN, 2020; DEALMEIDA et al., 2021).

The records are organized by dividing the study region into a square grid of 100 x 100 cells, each with an ID and a position `(x, y)`[4]. Records are obtained at 10-minute intervals and contain the cell ID, the timestamp of the measurement start, and details of SMS, phone call, and internet activity. For security and privacy reasons, the CDR data were resized by a factor known only to the provider, but which preserves the real traffic distribution. Table 3 shows an example of the first 5 data points from a CDRs. The activity data (SMS, calls, and internet) were aggregated into a single measure called Total Activity, as described in (ZHU; SUN, 2020). Figure 5 shows the distribution of control activity during the months of November and December

Given the focus on the AMF function, it is assumed that approximately 10% of the Total Activity corresponds to Control Activity, i.e., the number of control-plane requests sent to the 5G core every 10 minutes. This proportion reflects the notion that interactions occurring in the user

[2] CDRs record user activity over time for billing and network management purposes.
[3] Hosted at <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EGZHFV>
[4] The location of each cell is available at <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/QJWLFU>

Figure 5 – Distribution of control activity for the months of November and December



Source: The author.

plane are generally accompanied by signaling procedures in the control plane, such as session establishment, mobility management, or context updates.

Although the choice of 10% as the share of Control Activity is an approximation, it is consistent with previous studies in the area. Works such as (ALAWE et al., 2018d) adopt this proportion as a representative order of magnitude for signaling traffic in relation to total user activity, since direct measurement of control plane requests is usually unavailable in public datasets. Similar simplifications are also employed in studies focused on proactive scaling of the 5G Core (KURANAGE et al., 2022; CHOUMAN; MANIAS; SHAMI, 2023), where traffic of interest is derived as a fraction of total activity. In our case, the Telecom Italia dataset preserves the statistical distribution of real mobile traffic, including temporal patterns (daily and seasonal peaks) and spatial heterogeneity across the city. Thus, applying the 10% factor uniformly allows the Control Activity to preserve the natural variability of the original dataset, making it a representative approximation for evaluating concept drift and overload in the AMF function.

Table 3 – Example of Communication Activities

ID	Timestamp	Country Code	Internet Activity	Inbound SMS	Outbound SMS	Inbound Call	Outbound Call
1	1383260400000	39	0.141864	0.156787	0.160938	0.052275	11.028366
1	1383261000000	39	0.278452	0.119926	0.188777	0.133637	11.100963
1	1383261600000	39	0.330641	0.170952	0.134176	0.054601	10.892771
1	1383262200000	39	0.681434	0.220815	0.027300	0.053438	8.622425
1	1383262800000	39	0.243378	0.192891	0.053438	0.080738	8.009927

Source: The author.

Although the Telecom Italia dataset was collected in 2013–2014, its use in this work remains appropriate. Over the past decade, mobile network traffic has undergone significant

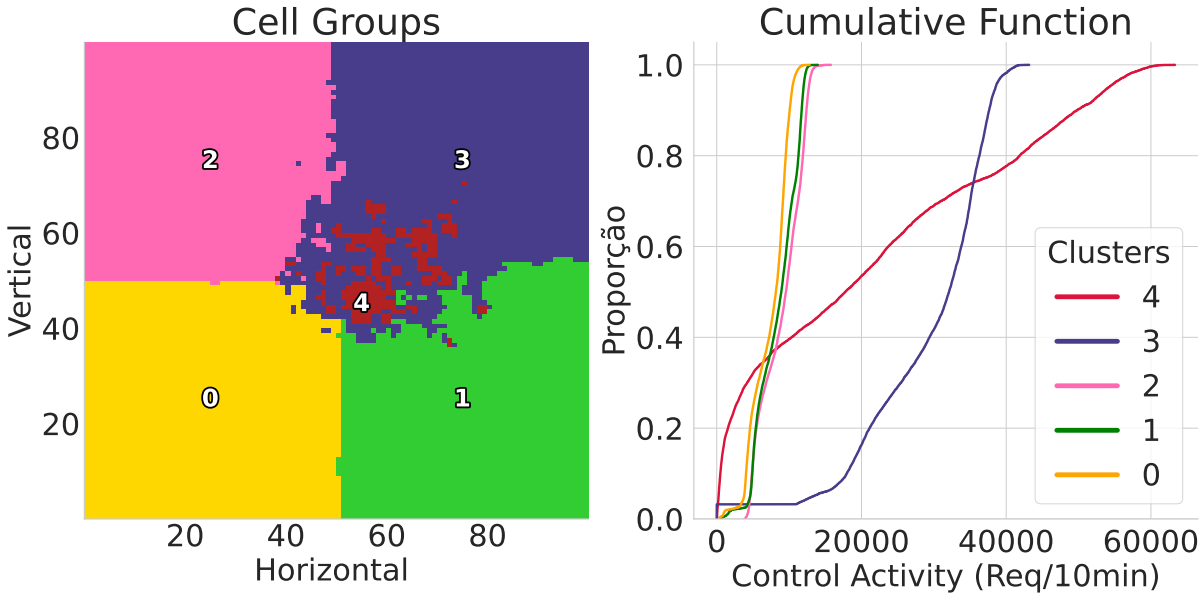
transformations in both volume and diversity of applications, particularly with the consolidation of 5G and the widespread adoption of data-intensive services such as video streaming, cloud gaming, real-time applications, and, more recently, Internet of Things (IoT) devices. Nevertheless, it is important to emphasize that control-plane load is not determined directly by the specific type of application in use, but rather by structural processes such as user authentication and registration, session establishment and release, mobility management, and handover procedures. These mechanisms are primarily driven by patterns of user access and mobility within the network, which tend to preserve stable characteristics across technological generations, even though variations in intensity or frequency may occur as services evolve.

The Telecom Italia dataset captures precisely these features at a metropolitan scale, recording millions of interactions distributed across time and space. This dataset enables the observation of temporal patterns (daily peaks, weekly cycles, seasonal variations), spatial heterogeneity (differences between central and peripheral areas), and urban mobility dynamics, all of which continue to directly influence the control plane in 5G networks. Therefore, although the absolute traffic volumes observed today are higher and applications more diverse, the behavioral and urban structures underlying control-plane load remain relevant, ensuring the suitability of this dataset for the problem under investigation.

Furthermore, the focus of this work is not the estimation of absolute contemporary traffic levels, but rather the evaluation of online learning strategies under concept drift in scenarios of proactive scalability of core network functions. For this purpose, the key aspect is the ability of models to adapt to changes in data distribution — a property that does not depend on the absolute level of traffic. In other words, the results should be interpreted as evidence regarding methods and mechanisms of adaptation applicable to the current context of 5G and future generations, rather than as an exact characterization of present-day traffic volumes.

To assess the impact of conceptual drift, each cell's Control Activity was used to divide the dataset into groups (*clusters*) with different patterns. *Clusters* allow simulating abrupt conceptual drift events, as it is possible to train a model with data from one *cluster* and test it with data from another *Cluster* with a different statistical pattern. Thus, similarly to the work by (DEALMEIDA et al., 2021; ZHU; SUN, 2020), the *K-means* clustering algorithm was applied to separate the cells into different *clusters* according to their Control Activity. Using the same dataset, the works evaluated different values of K, determining the optimal number of 5 *clusters*. Figure 6 shows the result of grouping the 10,000 cells into 5 *clusters*, on the left, and the Cumulative Control Activity Function in each *Cluster* (in requests per second), on the right.

Figure 6 – Left: Grid of cells subdivided into five clusters. Right: Cumulative function of Control Activity in each cluster.



Source: The author.

In the 10,000-cell grid, it is possible to observe that *Clusters* 2 and 3 cover primarily the central region of Milan and, therefore, their distributions are more extensive, being essentially different from each other and from the others. *Clusters* 0, 1, and 4, in turn, present similar spatial coverage and relatively similar distributions, with a smaller amplitude. These results suggest significant differences in the traffic pattern between *clusters* 2 and 4 and the remaining three, while the latter show greater similarity.

Several datasets available in the literature were considered for this work, including that of **EURECOM** (MEKKI; TOUMI; KSENTINI, 2022), which uses the my5G-RANTester (SILVEIRA et al., 2022) traffic generator to simulate loads on the AMF function, implemented with OpenAirInterface. This study collects metrics, including CPU usage, memory usage, and latency, to evaluate the function's performance in containerized environments.

While **EURECOM's** work is valuable, particularly for employing the AMF function in a containerized environment aligned with current network virtualization practices, its approach relies exclusively on infrastructure metrics such as CPU and memory usage. These metrics, while useful for some evaluations, are limited as scalability signals, as they reflect only the internal state of computing resources, without adequately capturing the service's functional behavior under fluctuations in traffic load. In particular, there is no direct visibility into the demand profile that led to these usage conditions, which compromises the ability to anticipate the need for scaling in dynamic or abrupt scenarios. Furthermore, analyzing this dataset from the perspective of conceptual drift would be ineffective, as the data does not reveal the operational context in which the functions were evaluated.

Given these limitations of this approach, which are common to other datasets, we chose to use the dataset provided by Telecom Italia, which represents real mobile traffic patterns across an entire city. Because this dataset is observational and not synthetic, it offers a broader time horizon, greater spatial granularity, and a scale more compatible with production environments. Due to its size and richness, it was possible to identify multiple traffic profiles even within the same region. These characteristics enabled the models to be exposed to different load patterns over time, allowing for the analysis of their ability to adapt to changes in behavior, an essential aspect for robust scalability strategies.

4.5 Control Activity Prediction Models

This section aims to present the regression models used in this research, highlighting their main characteristics and the hyperparameters adopted. In order to maximize the performance of each model, a search for hyperparameters was performed using November control activity data from Cluster 0. This search was conducted based on the *Tree-structured Parzen Estimator* (TPE) algorithm (BERGSTRA et al., 2011), a Bayesian optimization technique implemented by the Hyperopt library. The choice of this method is due to its effectiveness in contexts where the objective function assessment involves sequential processing of temporal windows in data flows. Contrary to the random or exhaustive search, TPE guides exploration to more promising search space regions, using the distribution of previous results to a smarter selection of new sets of hyperparameters.

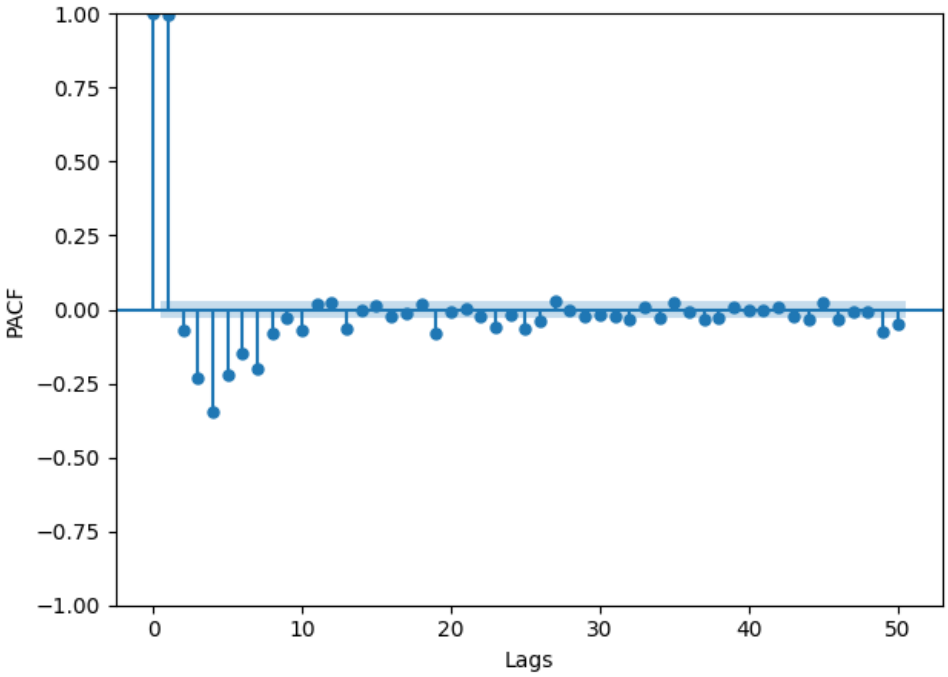
To search for the best hyperparameters, the November data of Cluster 0 was used, divided into 70% of the initial training data and 30% of the following test data. The process used size 20 temporal windows, defined based on the partial autocorrelation function applied to November data in different clusters to capture more relevant temporal dependence, to exemplify, Figure 7 shows the autocorrelation function applied in cluster 0 data for the month of November. Each model was subjected to 40 optimization attempts, in which the TPE algorithm sought to minimize the *Root Mean Square Error* (RMSE) in the test data. The following are the regression models employed in this research, together with the respective hyperparameters adopted.

The model AMFR was configured with 61 predictors. Tree weight update was performed at a learning rate of 98.56%, and the aggregation mechanism between predictors was activated.

For the ARF, 16 trees were used as a base. The selection of attributes in each division was made considering the square root of the total number of available variables. The method of adding predictions was the median.

The trees had a maximum depth of 20 levels and divisions after a waiting period of 29 instances. Data reamination was conducted with a Lambda value equal to 1. The leaf forecast was adaptive, allowing alternation between averages and internal models, with the choice based on quadratic error and 71.76% exponential decay softening. The sensitivity to conceptual change

Figure 7 – Partial Autocorrelation Function (PACF) for Cluster 0 - November



Source: The author.

was configured with a delta value of 6.19%, and the criterion for resolution of divisions was set with a threshold of 0.088%. The weighted vote between the trees has been deactivated. The model did not apply pre-dispute based on merit.

BaR was composed of 29 linear regression predictors, each using a 1.06%learning rate for intercept.

In the BLR was defined with Bayesian coefficients where the alpha parameter, representing the accuracy of the a priori distribution, was equal to 1. The beta parameter, related to the noise of the data, was adjusted to the value 1.2. No additional softening technique has been employed.

In EWA, three linear predictors were used, each with distinct optimizers: Stockhill, Adagrad, and RMSPROP. The combination of predictions was weighted according to the individual performance of each predictor, considering quadratic error as an evaluation metric. The weight update rate was configured at 0.15%. All input data were normalized incrementally

For HAT, the model assesses the possibility of division for each 350 instances, with a maximum depth of 10 levels. Hoeffding's statistical test was conducted with a significance level of 8.18×10^{-5}. The threshold used for divisions was 5.49%. The prediction in the leaves was performed adaptively, alternating between average and linear regression, based on the quadratic error softened by a decay factor of 91.16The model applied subsurvent replacement based on a significance level of 4.93%. Bootstrap sampling was deactivated, and binary divisions were used. The pre-poda was activated based on merit.

The HTR model, with parameters similar to the adaptive model, was followed, except for the replacement of subsidiaries, which was not applied. The model made divisions every 197 instances and considered a maximum depth of 10 levels. The significance level adopted was 0.0726, with a draw resolved from a 3.73%threshold. The prediction in the leaves was adaptive, and the smoothing was applied with a decay factor of 70.12%. Binary and pre-poda divisions were kept.

KNN considered 29 nearest neighbors to make their predictions, with added values by median. Updates were done online, with progressive storage of the observed instances.

LiR was configured with a learning rate for the independent term of 16.32%, and with L2 regularization with intensity of 4.75%.

In the case of OXT, the model consisted of 8 predictors, with selection of attributes based on the logarithmic scale in base two. Reamosmão followed the subscription strategy, with a rate of 80.96%. The concept detection mechanism has been disabled, as has the randomization of the depth of the trees. The maximum permitted depth was 20 levels. The divisions occurred after the accumulation of 73 instances and required at least 14 samples per subdivision. The forecast on the leaves was adaptive, guided by quadratic error with decay of 88.05%. The model considered binary divisions, used a Delta value of 8.13% and a 7.26% tie threshold. The weighted vote was deactivated, but the pre-foddle was kept active.

The model PAR operated in standard mode, performing updates only when the error exceeded a tolerance margin. The margin was defined as 97.2%, and the coefficient of regularization was 2.55. The constant term was included in the model.

The model SGT was configured with a significance level of 4.52×10^{-6} for the division statistical tests. The divisions were evaluated every 30 observed instances, and the depth of the tree was limited to 5 levels. The initial forecast was defined as 82.82. To restrict the magnitude of predictions and control the frequency of divisions, penalties were applied, with regularization coefficients of 4.05% and 6.88%, respectively.

Finally, the model SRP was configured with 20 predictors trained from random subsets of instances and attributes. The proportion of variables considered in each predictor followed the square root of the average of the total attributes. The training was based on the subspace technique combined with instance sampling. A simple average performed the aggregation of predictions. The weighted vote between the predictors was deactivated. No concept detectors or alert mechanisms were used, and attribute selection was entirely random.

Table 4 presents the average quadratic error values (RMSE) obtained by each model at the end of the search, using the best hyperparameters found during the optimization process.

Table 4 – RMSE values of the models achieved in the search for hyperparameters

Model	**RMSE**
AMFRegressor	243,4920
ARFRegressor	175,9335
OXTRegressor	171,2379
BaggingRegressor	194,3798
EWARegressor	204,7488
SRPRegressor	195,1911
HoeffdingAdaptiveTreeRegressor	197,5937
HoeffdingTreeRegressor	199,8919
SGTRegressor	310,2746
BayesianLinearRegression	169,9306
LinearRegression	192,7519
PARegressor	2119,8044
KNNRegressor	261,5358

Source: The author.

In order to compare the obtained results, we used the LSTM model as a representative of the offline learning models. Commonly used for time series forecasting, LSTM was selected due to the results presented in (ALAWE et al., 2018d), which, using the same dataset, showed that the LSTM model was more accurate than other offline models. However, it is important to note that in related works, LSTM was typically applied to classification problems, where the model predicted load classes. In contrast, this dissertation addressed the problem as a regression task, directly forecasting the number of AMF requests over time. This formulation enabled a finer-grained prediction and better integration with the proactive scaling mechanism. Therefore, it was necessary to implement and optimize a new LSTM model specifically tailored to regression rather than reusing the classification-based versions found in previous works.

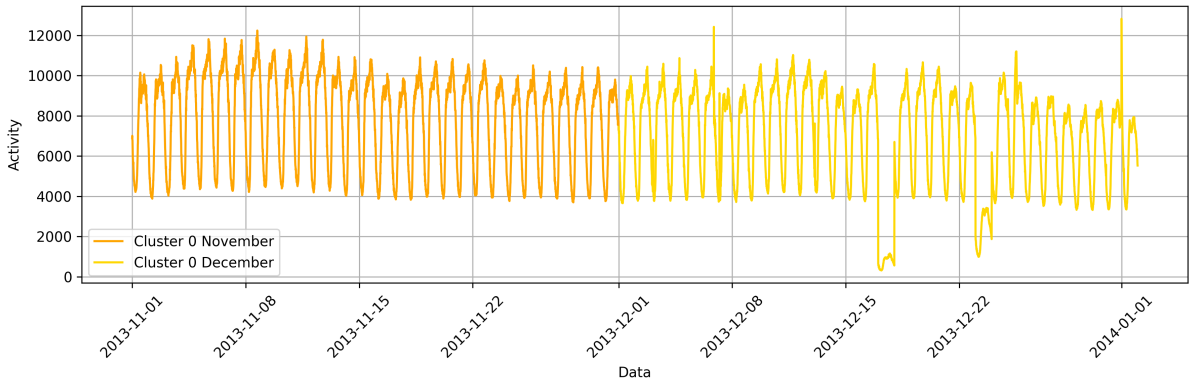
The best LSTM model obtained by the search has an LSTM layer with 240 hidden units, followed by a dense layer with one output unit. The activation function used in the LSTM layer was the hyperbolic tangent, while the dense layer used linear activation. A learning rate of 0.00104 and a batch size of 80 were adopted. At the end of the optimization process, a validation RMSE of 177.5471 was obtained.

To evaluate the models, we consider a cross-training/cross-testing scenario, where the November Control Activity data from a given cluster is used for training, and the December data from another cluster is used for testing. In total, 120 different combinations of training and testing pairs could be performed across the available clusters. However, in this work, we restrict the evaluation to using the November data from cluster 0 as a training base, testing the models on the December data from all other clusters, including cluster 0 itself.

Figure 8 illustrates Scenario 0, which corresponds to the activity pattern of Cluster 0 throughout November and December, training and testing respectively, with no transition

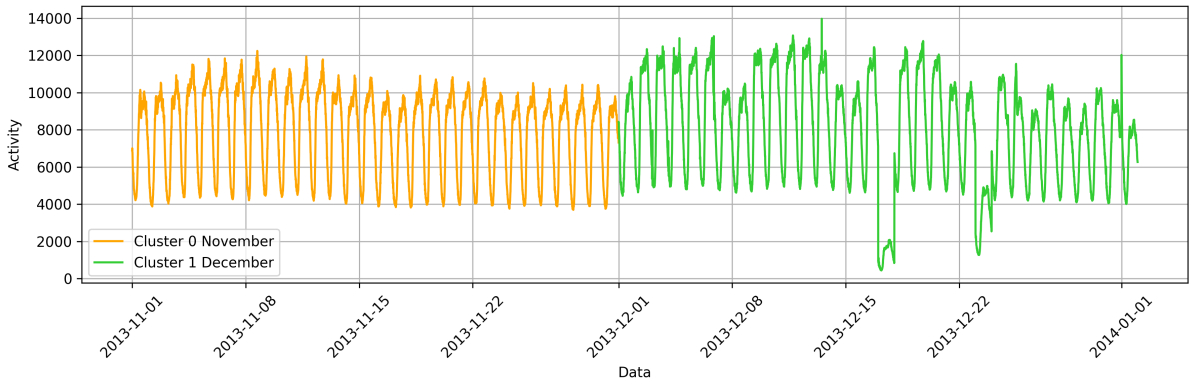
between clusters. Next, Figure 9 illustrates Scenario 1, combining the activity of Cluster 0 (November) with that of Cluster 1 (December). Similarly, Figure 10 shows Scenario 2, based on Cluster 0 (November) and Cluster 2 (December), while Figure 11 corresponds to Scenario 3, combining Cluster 0 (November) and Cluster 3 (December). Finally, Figure 12 presents Scenario 4, which combines Cluster 0 (November) with Cluster 4 (December), used in one of the test cases.

Figure 8 – Scenario 0 (Activity): Cluster 0 (Novembrer) + Cluster 0 (December)



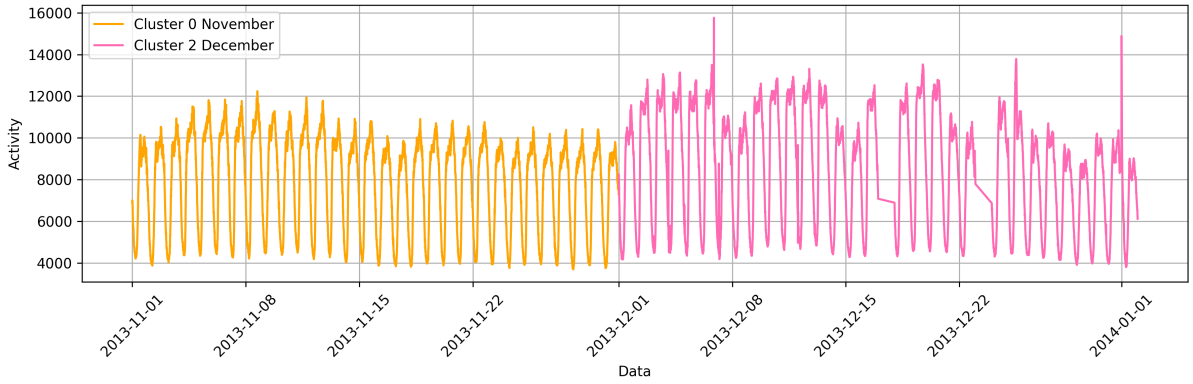
Source: The author.

Figure 9 – Scenario 1 (Activity): Cluster 0 (Novembrer) + Cluster 1 (December)



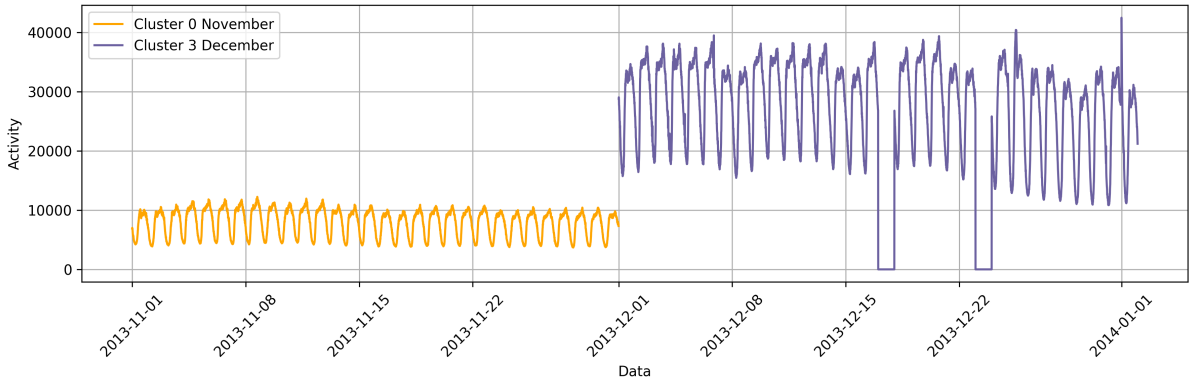
Source: The author.

Figure 10 – Scenario 2 (Activity): Cluster 0 (Novembrer) + Cluster 2 (December)



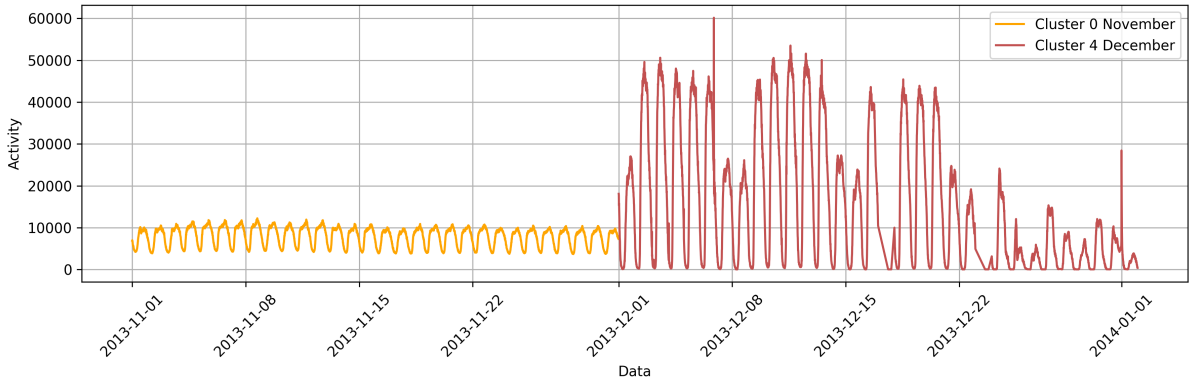
Source: The author.

Figure 11 – Scenario 3 (Activity): Cluster 0 (Novembrer) + Cluster 3 (December)



Source: The author.

Figure 12 – Scenario 4 (Activity): Cluster 0 (Novembrer) + Cluster 4 (December)



Source: The author.

These scenarios evaluate the models' ability to handle concept drift, whether artificial, such as that induced by cluster switching between training and testing, or natural, resulting from

the data's inherent dynamics. An example of this behavior can be observed when analyzing the traffic pattern during the holiday season, such as Christmas and New Year's, when a clearing of the city center is noticeable. As shown in Figure 12, the gradual reduction in activity starting on December 22nd corresponds to a distinct pattern compared to previous periods.

Another behavior that will be analyzed is network failure. In this context, the scenarios in Figures 8, 9, and 11 stand out, as they represent particularly challenging cases for the models. In these scenarios, a sharp drop in activity is observed during December, with values close to zero between the 15th and 23rd. This type of disruption disrupts the regularity of the previous pattern. It poses a significant challenge for the models, which must be able to adapt to sudden and unforeseen changes in data dynamics.

To address these different scenarios, the models were configured to use a window of 20 past occurrences of activity values as a basis for predicting the next value. This window size was defined based on partial autocorrelation analysis of the November data, taking into account the variability between the different clusters.

The evaluation of the models' performance uses the *Mean Absolute Error* (MAE), a measure of the average magnitude of the errors and is expressed as MAE $= \frac{1}{n} \sum i = 1^n |y_i - \hat{y}i|$.

In addition, the computing costs of the models were analyzed according to: *Inference Time* (IT), the mean time required for the model to make a prediction; *Learning Time* (LT), the average time the model takes to update its parameters for each new data sample; and the size of the model in memory, measured in kilobytes (KB) at the end of each experiment.

4.6 Simulation

The last step of the evaluation is a trace-based simulation (JAIN, 1991) that evaluates the different models regarding the provisioning of AMF instances and computes high-level operational indicators. The simulator, developed in Python with the Simpy library[5], receives as input a trace containing the actual Control Activity data and the predicted Control Activity - coming from the model under evaluation - at each instant in time, with a sampling interval of 10 minutes (see Section 4.4).

Models of the Scaling Module, Load Balancer, and AMF replicas were implemented. During each time interval, each AMF replica receives a portion of the control traffic from the Load Balancer, processes the amount supported by their capacity, and rejects any exceeding requests. Based on the measurements reported in (ALAWE et al., 2018d), the capacity of each AMF instance is set to 20 requests per second (i.e., 1,200 requests over 10 minutes).

To minimize the Load Balancer influence on scaling decisions, the traffic distribution follows the round-robin algorithm, with a slight modification when an instance is being removed.

[5] <https://simpy.readthedocs.io/en/latest/>

In a 5GC, one should avoid terminating an AMF instance if there are any UEs associated with it. The behavior of real UEs was emulated using observations from (MENG et al., 2023), which, based on data from an American operator, showed that most UEs remain in service for up to 2 minutes, with only 1% remaining for more than 16 minutes. Therefore, instances are removed gradually: in the first 10-minute interval after the removal decision, the Load Balancer sends traffic to the instance occupying 5% of its capacity; in the next 10 minutes, the share is reduced to 1% before complete removal.

The Scaling Module decides, using Equation 4.1 at each time interval, the number of instances ($N_{\text{predicted}}$) required to meet the predicted demand. The occupancy of the instances is limited to 80% so that they remain in a region of efficient operation, avoiding reaching prohibitive response times (JAIN, 1991).

$$N_{\text{predicted}} = \left\lceil \frac{\text{Predicted Control Activity}}{0.8 \times \text{Capacity of AMF}} \right\rceil . \tag{4.1}$$

Once the $N_{\text{predicted}}$ is defined, the Scaling Module adjusts the number of AMF instances according to the Algorithm 1. New AMF instances are created within the time interval in which the prediction is received (since the time to create an instance is usually less than 10 minutes, in practice). However, to mitigate the effect of oscillations, instances removal is made gradually.

Algorithm 1: Active instance management algorithm

Input : $N_{\text{predicted}}$: Predicted number of required instances;
$\text{Total}_{\text{instances}}$: Total number of active instances;
$\text{Life}_{\text{instance}}(i)$: Current lifetime of instance i;
Output : Updated states and $\text{Life}_{\text{instance}}(i)$;
if $Total_{instances} > 1$ **then**
  Decrease the lifetime of all instances;
end
if $Total_{instances} < N_{predicted}$ **then**
  Increase the lifetime of all instances;
  Increase the number of instances;
else if $Total_{instances} == N_{predicted}$ **then**
  Increase the lifetime of all instances;
else if $Total_{instances} > N_{predicted}$ **then**
  Increase the lifetime of the first $(\text{Total}_{\text{instances}} - N_{\text{predicted}})$ instances;
foreach $instance(i)$ **do**
  if $Life_{instance}(i) == 0$ **then**
    Shut down instance i;
  end
end

Each instance has a "life" indicator that turns the instance off when the indicator reaches 0. At each time interval, the life of all instances is reduced, and three conditions are evaluated. The first case occurs when the number of current AMF instances ($\text{Total}_{\text{instances}}$) is lower than the required number ($N_{\text{predicted}}$). In this case the life of all instances is increased and new instances are created to meet the demand. When $\text{Total}_{\text{instances}}$ and $N_{\text{predicted}}$) are equal, all instances have

their lives restored so that they remain active. In the third case, only some instances have their lives restored. The last loop marks the instance for shutdown. Only in this case will the Load Balancer gradually act to reduce traffic to that instance. In the event of a failure in the forecast process, such as the generation of underestimated or negative values, the system maintains the operation of at least one active instance of AMF, ensuring minimal continuity of the service.

The performance of the models in simulations was evaluated using three metrics: *Full Occupancy Count* (FOC), measures the number of events in which requests exceeded the maximum capacity of the AMF instances; *Number of Lost Requests* (NLR), the total number of requests not served throughout the simulation; and *Difference between Ideal and Predicted AMF Instances* (DIPI), which computes the error between the appropriate number of AMF instances to handle the current demand and the $N_{\text{predicted}}$ value, with negative values indicating undersizing and positive values indicating oversizing.

The choice of using a simulator instead of real 5G core implementations (such as free5GC or Open5GCore) is grounded in three main aspects: (i) causal control and isolation of variables, (ii) reproducibility and experimental scalability, and (iii) cost, time, and comparability across methods.

First, causal control and isolation of variables are essential for analyzing the effectiveness of different online learning strategies under concept drift. Real implementations introduce multiple confounding factors — such as specific hardware characteristics, operating system configurations, scheduling policies, and code-level particularities — that may obscure or distort the actual effect of the algorithms under evaluation. A simulator, in contrast, allows these factors to be fixed or varied in a controlled manner, ensuring that performance differences can be attributed primarily to the adaptation methods being analyzed.

Second, reproducibility and experimental scalability are indispensable in comparative studies. Evaluating robustness under diverse concept drift scenarios requires executing multiple independent replications and systematically varying parameters, which, in real environments, would entail high operational cost and low predictability of results. In a simulator, however, it is possible to standardize conditions, define random seeds, automate runs, and even conduct multiple experiments simultaneously. This enables broader coverage of test scenarios and ensures stronger statistical consistency in the conclusions.

Finally, cost, time, and comparability across methods are significant factors. Simulation substantially reduces operational overhead, allowing the exploration of a wide range of strategies within a reduced timeframe. Moreover, simulators enable the compression of temporal scales, making it possible to observe behaviors that would normally unfold over weeks or months within only a few hours of execution.

To quantify the experimental uncertainty of the simulator, each evaluation scenario was re-executed 30 times: one execution with the original dataset and 29 executions with variations

generated by jittering. This technique consists of adding point-wise Gaussian noise to the input time series, producing synthetic replicas that preserve relevant statistical properties while introducing controlled variability (IGLESIAS et al., 2023).

Let x_t denote the series (or feature vector) at time t. For each repetition $r \in \{2, \ldots, 30\}$, a synthetic replica was created as:

$$x_t^{(r)} = x_t + \varepsilon_t^{(r)}, \quad \varepsilon_t^{(r)} \sim \mathcal{N}(0, \sigma_j^2),$$

where $\sigma_j = \kappa \cdot \hat{\sigma}_x$. Here, $\hat{\sigma}_x$ is the sample standard deviation (per feature) and κ is a scaling factor. For sensitivity analysis, values of $\kappa \in \{0.01,\ 0.02,\ 0.05\}$ were considered. For non-negative variables, truncation was applied $x_t^{(r)} \leftarrow \max(0, x_t^{(r)})$, while categorical or integer variables were not perturbed.

For the 29 jittered repetitions, the following random seeds were used: {2, 49, 85, 111, 127, 184, 211, 222, 244, 263, 324, 383, 413, 455, 462, 511, 536, 538, 578, 620, 657, 670, 695, 781, 826, 858, 869, 936, 988}. The execution without jitter employed the same seed as in the baseline experiment.

Each replica $r = 1, \ldots, 30$ was processed through the complete pipeline, producing a set of performance metrics. For each metric M, the point estimate was reported as the sample mean $\bar{M}$ and the 95% confidence interval was calculated under the normal approximation as:

$$\text{CI}_{95\%}: \quad \bar{M} \pm 1.96 \cdot \frac{s_M}{\sqrt{30}},$$

where s_M is the sample standard deviation of the 30 executions. This confidence interval was reported by model, by scenario, and also in an aggregate form, in the latter case weighted by the number of windows or observation time of each cluster.

On the other hand, for the DIPI, which corresponds to the empirical distribution of accumulated errors for each model under each scenario, a distinct procedure was adopted. Specifically, we aggregated all observation windows from the 30 executions (1 original + 29 with jitter) and computed the empirical frequency of each deviation value with respect to the ideal number of instances. Formally, for each possible deviation k, the proportion is given by

$$\hat{p}_k \ = \ \frac{n_k}{N} \times 100\%, \qquad k \in \{-K_{\max}, \ldots, 0, \ldots, K_{\max}\},$$

where n_k is the number of windows with $\text{DIPI} = k$ and N is the total number of windows considered. Results are reported as percentages for each k, highlighting the mass at $k = 0$ (exact match), negative values (under-provisioning), positive values (over-provisioning), and possible asymmetries. Aggregated results across clusters were obtained by simple pooling ($n_k = \sum_c n_{k,c}$, $N = \sum_c N_c$).

4.7 Chapter Summary

Chapter 4 presents the proposed methodology for evaluating the proactive scalability of network functions based on online learning models. Initially, the architecture of the developed system is described, consisting of modules responsible for data collection, control activity prediction, and decision-making for automatic instance scaling. This architecture was designed to operate in a continuous flow, enabling scalability decisions guided by predictions updated with each new data input.

In the evaluation stage, the learning models used, the dataset adopted, and the simulator developed for experimentation are detailed. Fourteen online learning algorithms, with different adaptation characteristics and computational complexity, were selected and applied to the task of predicting the signaling load of the AMF function, in addition to an offline model used as a comparative reference.

The dataset was constructed from real network records, processed and organized into spatial clusters to represent different regions of the infrastructure. From this set of data, an experimental environment was created that simulates multiple types of concept drift—including abrupt, gradual, incremental, and recurring variations—to test the models' adaptability in dynamic scenarios.

To facilitate the experiments, a simulator was developed capable of integrating model predictions into the automated scaling process. The simulator operates with temporal granularity, allowing for metrics such as the number of lost requests, total occupancy events, and discrepancies between the ideal and predicted number of function instances.

5 RESULTS

This section presents an evaluation of the models, considering three complementary dimensions: prediction quality and computational efficiency, as well as system-level operational performance. The analysis is divided into two parts: the first compares different online learning approaches, evaluating their merits and limitations. The second part compares the best identified online approach with a reference model based on offline learning.

5.1 Online Learning Algorithms: Performance Evaluation

5.1.1 Forecasting and Computing efficiency

Table 5 presents the MAE values for all models in each scenario. The five models with the best average performance were: BLR, OXT, EWA, ARF, and SRP. The BLR model had the lowest overall mean error, with a MAE of 283.86, followed by OXT (385.09), EWA (420.05), ARF (543.45), and SRP (485.22). These models performed relatively consistently in the scenarios with lower activity levels (scenarios 0-2), exhibiting significantly fewer minor errors.

Table 5 – MAE values in each scenario for all models

Model	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Mean
AMFR	246.31	292.52	297.04	773.97	1382.59	598.48
ARF	209.79	198.96	167.82	1076.90	1063.79	543.45
OXT	178.19	187.99	170.81	670.80	717.64	385.09
BaR	5.82E+12	3.80E+12	180.52	6.70E+13	760.19	1.53E+13
EWA	184.21	200.60	200.18	653.77	861.48	420.05
SRP	211.55	232.39	192.71	907.52	881.91	485.22
HAT	1.26E+13	1.10E+10	184.26	1.23E+14	884.23	2.71E+13
HTR	362.68	388.99	184.43	1784.41	880.05	720.11
SGT	417.68	485.08	360.26	1373.22	2787.97	1084.84
BLR	129.34	142.41	148.13	425.73	573.70	283.86
LiR	4.37E+13	4.49E+13	182.47	4.13E+14	810.98	1.00E+14
PAR	2673.54	3119.77	3313.19	10138.35	15957.14	7040.40
KNN	317.45	389.90	314.67	1215.81	1639.70	775.51

Source: The author.

However, an increase in errors was observed in scenarios 3 and 4, regions characterized by higher traffic intensity. Even the best models suffered degradation in these regions. For example, OXT increased from 170.81 (scenario 2) to 670.80 (scenario 3), and BLR from 148.13 to 425.73. This trend indicates that the models have greater difficulty capturing patterns in scenarios with greater variability. Among the models with poorer performance, KNN stands out,

exceeding a MAE of 1200 in scenarios 3 and 4. Models such as SGT and PAR also exhibited high errors, particularly in higher-intensity scenarios.

Furthermore, other interesting behaviors are found when observing the results of the BaR, LiR, and HAT models, in which they presented extremely high errors in scenarios 0, 1, and 3, with MAE values that reach orders of magnitude between 10^{10} and 10^{14}. These anomalous performances significantly influenced their overall averages, positioning them among the models with the worst average performance.

On the other hand, these same models obtained quite competitive results in scenarios 2 and 4. In scenario 2, for example, BaR obtained a MAE of 180.52, a value comparable to that of HTR (184.43) and OXT (170.81). A similar situation is observed in scenario 4, where HAT obtained a MAE of 884.23, close to the result of SRP (881.91) and significantly better than that of KNN (1639.70).

This behavior can be attributed to the fact that, in scenarios 0, 1, and 3, network failures occurred during the period 2014-12-16 to 2014-12-23, resulting in a sharp drop in the number of requests, which reached values close to zero. This atypical condition compromises the statistical pattern of the data, making it more challenging to develop models that effectively handle extreme regimes or abrupt breaks in the time series. As a result, these models exhibit degraded performance in these scenarios, which substantially affects their overall averages.

Figure 13 illustrates this limitation by presenting the results obtained by the LiR model in **scenario 3**. To visualize the predictions and observations, which vary widely in magnitude and include both positive and negative values, the (*signed logarithmic scale*) was adopted, defined by the transformation:

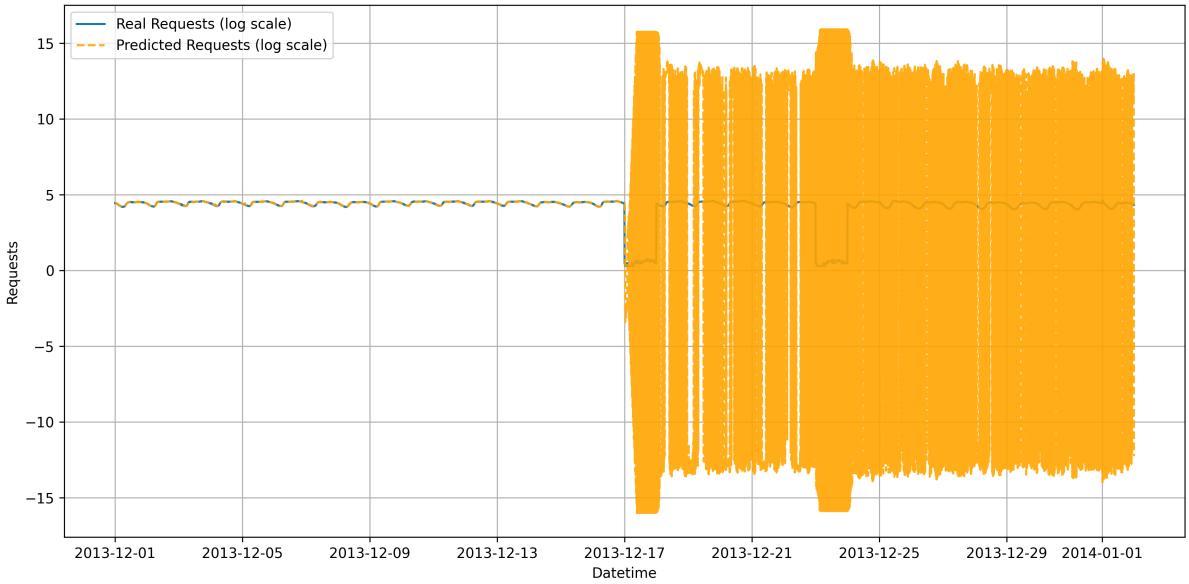
$$y = \text{sign}(x) \cdot \log_{10}(1 + |x|)$$

This scale allows the data to be represented in a single continuous band, reducing the visual impact of outliers while preserving the signal information. It can be observed that, in the initial periods, the model reasonably follows the trend of the data. However, as the interval associated with the network failure approaches, its predictive ability deteriorates substantially.

Figure 14 presents the *Cumulative Distribution Function* (CDF) of the residual error (*Residual Error* (RE)) for the models evaluated in Scenario 0, representing the cumulative frequency of error occurrence on a logarithmic scale. This visualization allows us to identify not only which models are most inaccurate but also how these errors are distributed, revealing properties such as bias, stability, and the presence of extremes.

In general, among the best-performing models, BLR stands out for its curve that is more concentrated around zero, with a steep leftward slope, indicating stable forecasts and low error variability. The OXT, ARF, and SRP models also maintain robust behavior, with a

Figure 13 – Log-Scaled Real vs. Predicted Requests for LiR (Scenario 3)



Source: The author.

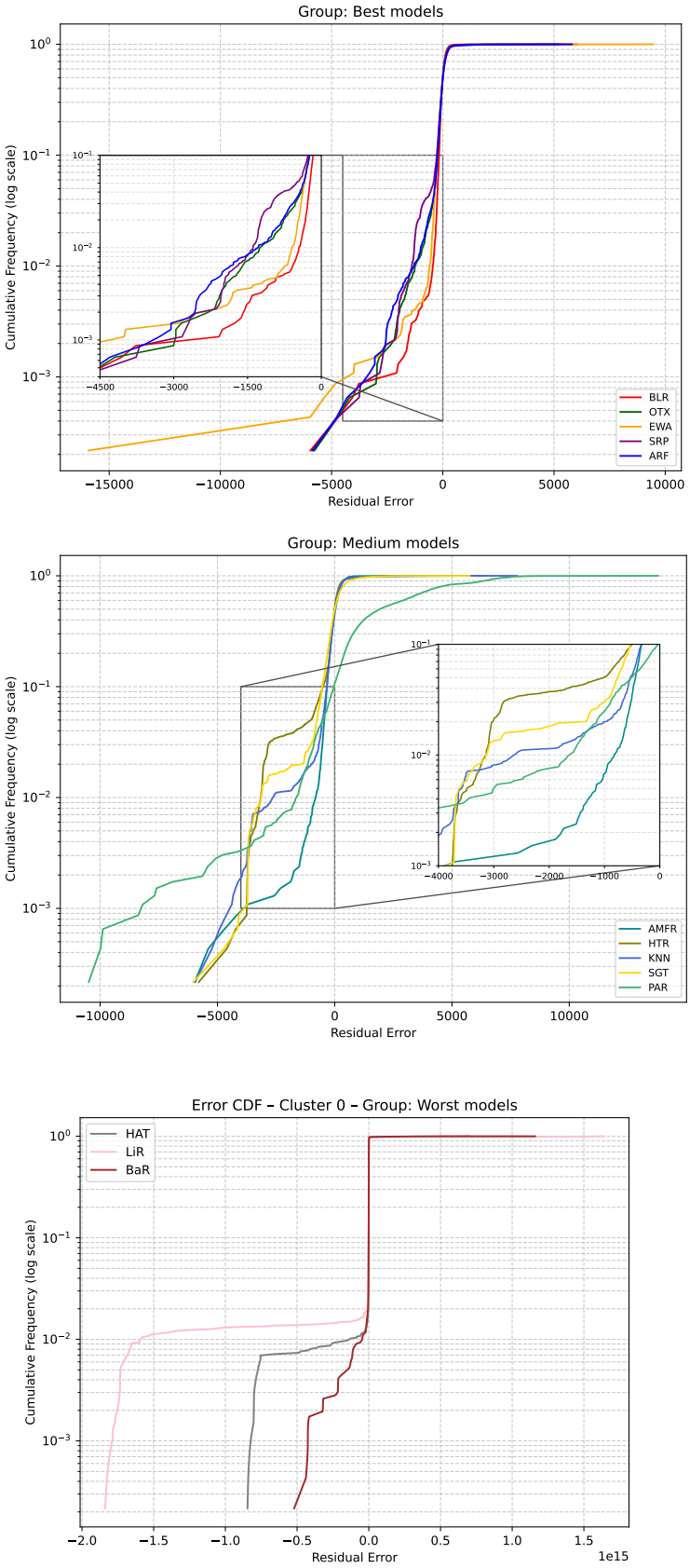
slight tendency toward underestimation. In this group, SRP stands out for presenting a more pronounced concentration of errors around -1000, which, while not representing a critical deviation, suggests a slight but consistent forecast bias below actual demand.

A specific exception, however, can be observed in the behavior of EWA, whose curve exhibits a steep left-sloping tail, with residual errors reaching magnitudes around -15,000. This trait indicates that, despite the model's good overall stability, it is susceptible to unstable behavior during abrupt transitions. As illustrated in Figure 15, EWA responds to the sudden drop in traffic with a sequence of extremely negative predictions, followed by a sharp upward corrective swing—a positive offset spike—before finally realigning with the actual series. This pattern suggests that the model is not only slow to capture the reversal, but also attempts to correct the drift aggressively and unbalancedly, resulting in a short period of instability.

In contrast, the LiR, BaR, and HAT models exhibit widely extended CDF curves, covering a range from -1.5×10^{15} to $+1.5\times10^{15}$, which indicates extreme errors of both signals, both with sudden bursts of overestimation, and varying levels of underestimation. The highly asymmetric and irregular distribution of these curves suggests chronic difficulty in dealing with the broken patterns observed in this scenario, which seriously compromises their applicability.

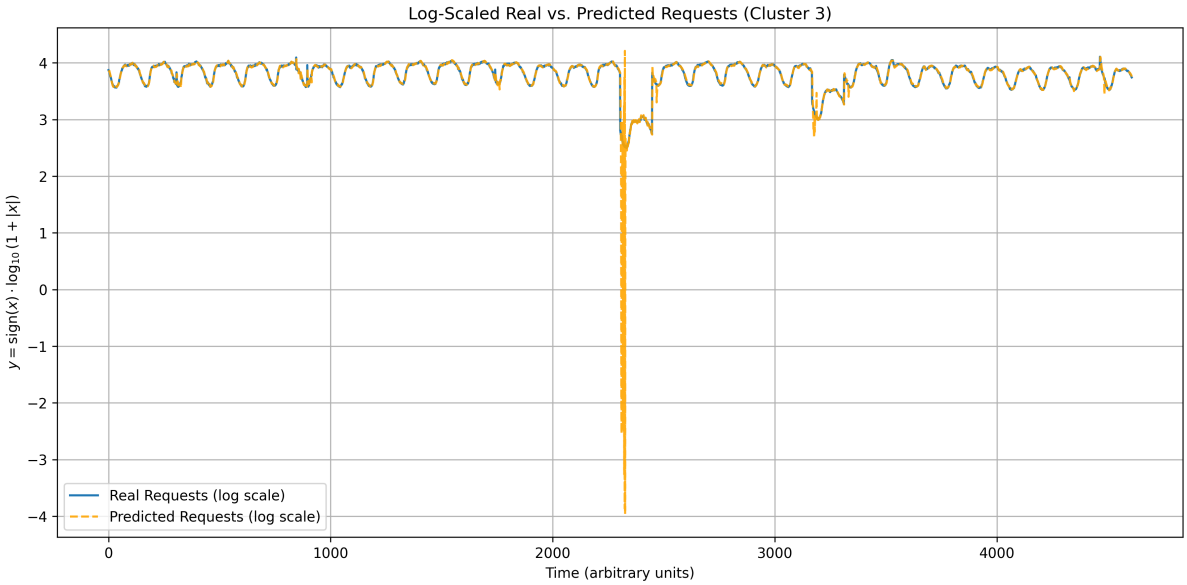
Intermediate models—such as HTR, AMFR, KNN, SGT, and PAR—present varied behaviors. HTR, for example, has a reasonably steep curve, close to the best models, but still with greater dispersion. KNN, on the other hand, draws attention to a clear inflection between RE values from -5000 to -1000, revealing an atypical concentration of negative residual errors. This pattern indicates a clear tendency toward underestimation, which may be related to the model's inability to adequately adjust to abrupt drops in the time series, especially during the

Figure 14 – Residual error distribution in scenario 0 for all model groups



Source: The author.

Figure 15 – Log-Scaled Real vs. Predicted Requests for EWA (Scenario 0)



Source: The author.

critical period of network failure.

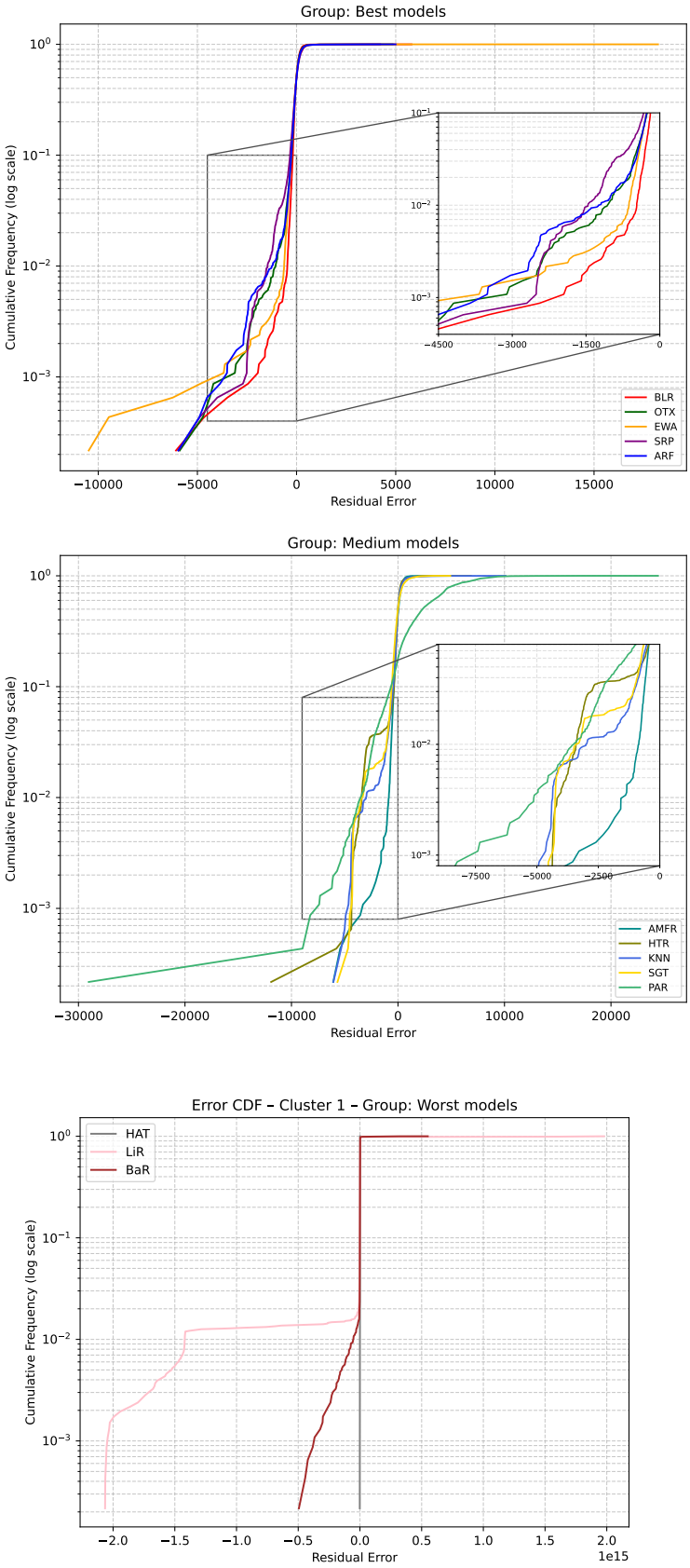
Finally, models such as PAR and SGT present curves that begin with a reasonable slope but extend into tails that reveal the presence of occasionally high errors. This indicates behavior that appears stable on average, but is subject to significant failures in specific situations.

The figure 16 CDF of RE in scenario 1, again with Y-shaft logarithmic scale in order to highlight low frequency and high magnitude events. The general behavior of curves in this scenario resembles that observed in scenario 0, reflecting the impact of network failures also present in this interval. However, some additional particularities stand out in the distribution of errors between the models.

Models classified as best performance BLR, OXT, EWA, SRP, and ARF, presented concentrated curves on the left, with fast growth and good compaction around zero. Among them, BLR and OXT continue to stand out for leaning and symmetrical curves, indicating high accuracy and low error dispersion. The SRP curve has a softer slope, suggesting a slight increase in dispersion, but still within acceptable behavior. The ARF shows a stable pattern, with distribution similar to that previously observed and without visible evidence of degradation. Its curve maintains good compaction and controlled growth, suggesting that the model remains consistent even in the face of network failures.

The EWA model, in turn, maintains a promising initial curve, but presents again, as already observed in scenario 0, a sharp tail to the right, sign of a recurring concentration of overestimation errors. Visual inspection even reveals multiple compensation moments, in which positive errors appear after intense negative oscillations, suggesting a model oscillatory response pattern.

Figure 16 – Residual error distribution in scenario 1 for all model groups



Source: The author.

In the intermediate group AMFR, HTR, KNN, SGT, and PAR, greater variability in error distribution is observed. The AMFR model has long tails and slow growth in the CDF, signaling relevant occurrence of extreme errors in both directions. O KNN once again has an inflection in the negative stretch of the curve, with error accumulation between -10000 and -3000, reinforcing its recurrent underestimation pattern. OHTR and SGT display intermediate curves, without extreme behavior, but with greater dispersion than the models of the first group. The PAR, in turn, shows a compact curve to some extent, but with a prolonged tail, which may represent vulnerability to abrupt variations.

Worse performance models, HAT, LiR, and BaR, follow the pattern already observed in scenario 0, having significantly displaced curves, with shallow growth and tails that extend by various magnitude orders. The visual pattern here reinforces the statistically detected anomalous behavior, with RE that reaches the house of 10^{15}. The shape of these curves suggests that these models are not only inaccurate, but also highly unstable, with recurrent errors of great magnitude, even in a relatively less complex scenario than the later ones.

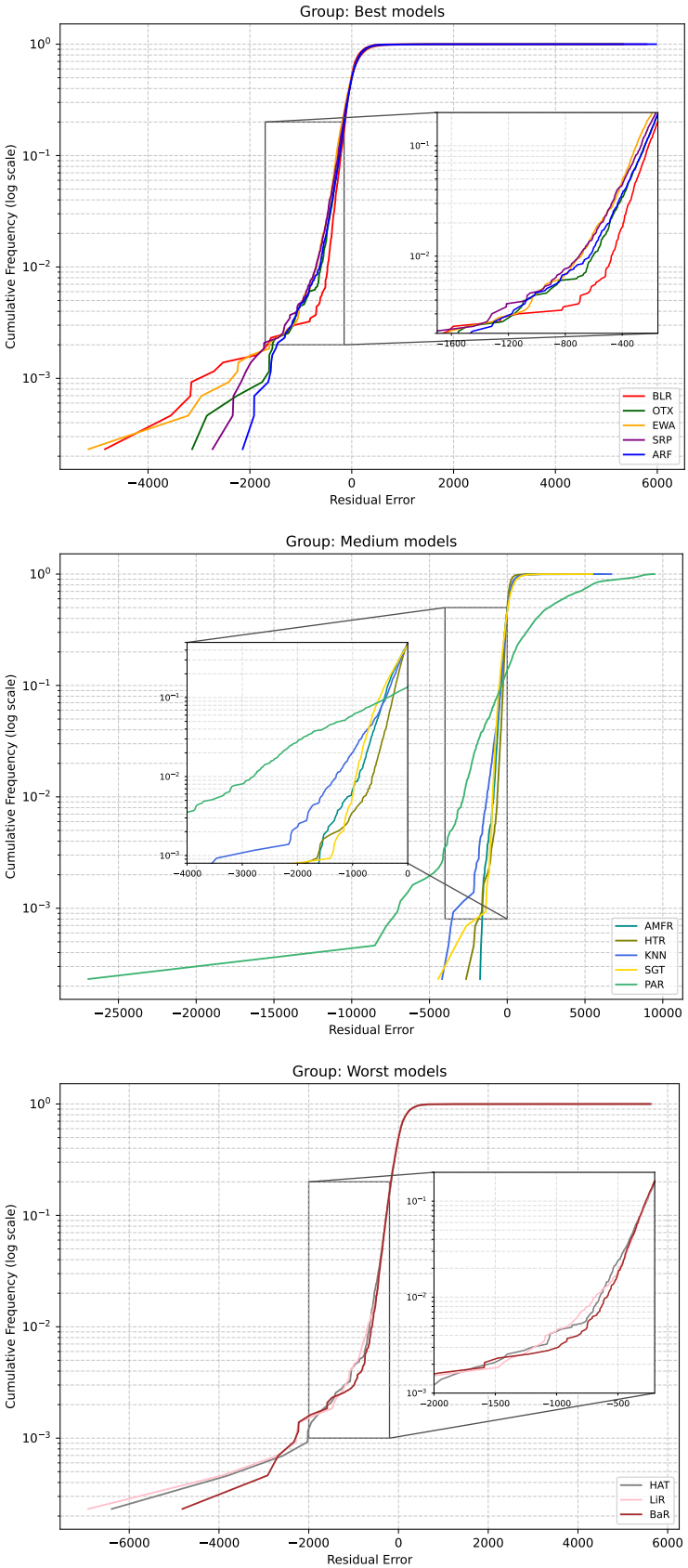
In scenario 2, represented by Figure 17, most models maintained the previously observed performance standard; however, some important deviations warrant highlighting. One of these cases is the BLR model, which, although it continues among the best in terms of overall distribution, now has a stretched tail on the left, with residual errors exceeding -4000. This behavior indicates episodes of more pronounced underestimation than in previous scenarios, which can signal a momentary loss of adherence to the local dynamics of the series.

Another relevant case is the PAR model, whose performance is substantially determined in this scenario. Its curve displays an extremely long negative tail, with errors that reach -25000, by far the most extreme among all models analyzed in this group. This result highlights a critical disability of the model in capturing local traffic standards in this scenario, suggesting structural fragility in the face of specific load fluctuations or possible sudden trend changes. The AMFR model, in contrast, shows a moderately dispersed curve, with no significant outliers. While it does not match the compactness of the best-performing models, its residuals remain within acceptable bounds, indicating a performance that is somewhat unstable but not critically so.

In this scenario, we can analyze the trends of models in scenarios where there are no abrupt failures. Under these stable conditions, the models LiR, BaR, and HAT, which in previous scenarios have significantly degraded performance, demonstrate much more contained error curves without large oscillations or pronounced tails. Its residual errors remain within acceptable limits, and its general behavior approaches that of the intermediate models.

Although they still exhibit greater dispersion and less inclination in the near-zero region, which indicates lower accuracy, these models are functionally stable in this type of scenario. In some cases, such as the PAR, their performances is higher, as this model presented errors accentuated in this scenario, with residual values that exceed $-25,000$.

Figure 17 – Residual error distribution in scenario 2 for all model groups



Source: The author.

In scenario 3, the result was consistent with that observed in scenarios 0 and 1 for virtually all models. That is, both positive and negative performance standards have been kept. However, as the traffic load in this scenario is higher, residual errors, although statistically similar to those of previous scenarios, become numerically more expressive. This increase in magnitude occurs because, even with visibly more controlled error curves, the deviations are now focused on larger volumes of traffic. The figure 18 illustrates this behavior, showing how the change in load distribution directly influences the magnitude of errors, although the structure of waste distribution remains relatively stable.

In Scenario 4, model performance standards are approaching what was observed in Scenario 2. In this case, the scenario has no abrupt failures; however, the distribution of the load is more demanding, which numerically expands residual errors without necessarily indicating structural degradation of the models. Models like BLR, OXT, EWA, SRP, and ARF continue with predictable behaviors - the first two with compact curves near zero, and the others maintaining stability, although subject to more intense deviations at specific points.

The figure 19 highlights this behavior, showing that, despite the greater magnitude of errors, the structure of error curves remains consistent with the previously observed. Lower performance models, such as LiR, BaR, and HAT also repeat the scenario 2 standard, continue with sharp dispersion, but without the extreme errors associated with failure periods, which reinforces that their poor performance is especially critical under unstable conditions.

Continuing the analysis, Table 6 presents the inference times (IT) of each model across the different scenarios. Since this work focuses on applications operating under stringent latency and scalability requirements, this indicator becomes as relevant as predictive accuracy.

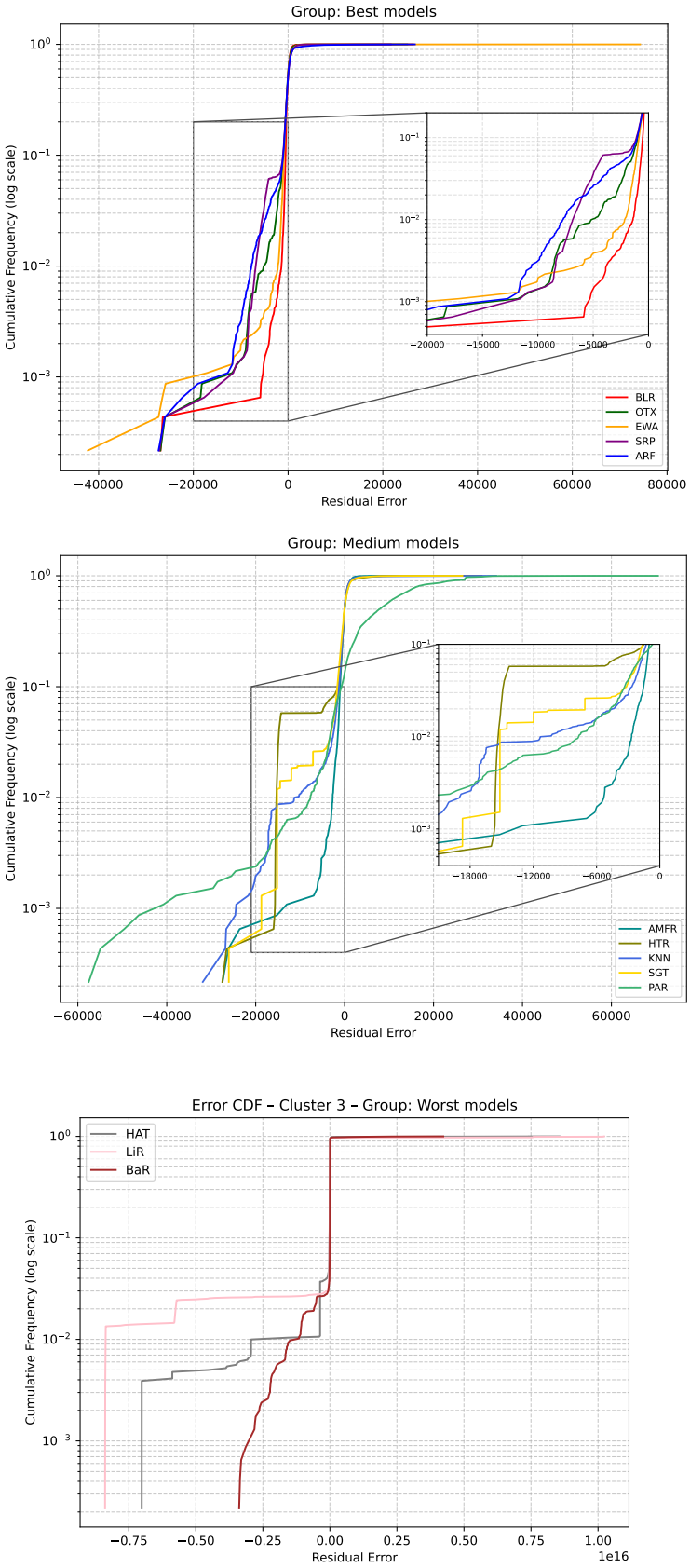
Table 6 – IT (in milliseconds) for each scenario of all models.

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Mean**
AMFR	4.2021	4.5605	2.1149	4.3946	3.3577	3.7260
ARF	0.2042	0.2455	0.2579	0.2234	0.2271	0.2316
OXT	0.0671	0.0667	0.0650	0.0646	0.0708	0.0668
BaR	0.1815	0.1826	0.1809	0.1818	0.1788	0.1811
EWA	0.0292	0.0295	0.0296	0.0294	0.0291	0.0294
SRP	0.2759	0.2001	0.2107	0.3073	0.3172	0.2622
HAT	0.0152	0.0153	0.0127	0.0150	0.0125	0.0141
HTR	0.0100	0.0099	0.0109	0.0099	0.0107	0.0103
SGT	0.0071	0.0060	0.0060	0.0060	0.0060	0.0062
BLR	0.0066	0.0066	0.0067	0.0066	0.0067	0.0066
LiR	0.0039	0.0037	0.0039	0.0038	0.0038	0.0038
PAR	0.0056	0.0054	0.0055	0.0055	0.0056	0.0055
KNN	2.1034	2.0988	2.1160	2.1247	2.0793	2.1044

Source: The author.

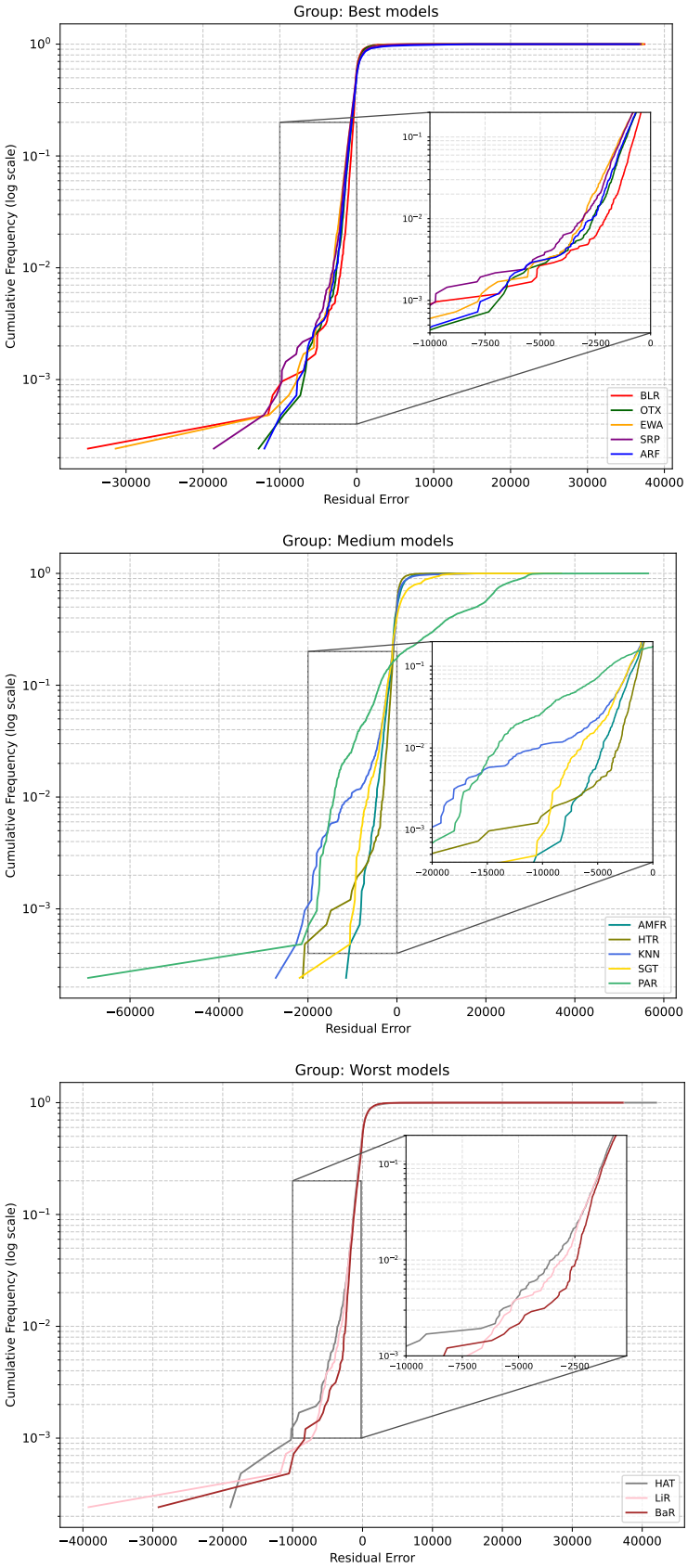
In this scenario, the BLR, OXT, EWA, and ARF models stand out as the most viable

Figure 18 – Residual error distribution in scenario 3 for all model groups



Source: The author.

Figure 19 – Residual error distribution in scenario 4 for all model groups



Source: The author.

alternatives, as they combine good MAE performance with extremely low inference times. BLR, for example, achieved the lowest average error among all models and also had one of the fastest response times (0.0066 ms), demonstrating excellent computational cost-benefit. OXT maintained similar performance, with a competitive mean error and an average inference time of only 0.0668 ms. These results suggest that such models are capable of operating efficiently even under heavy load scenarios, directly contributing to the network's responsiveness and scalability.

The EWA model, although with a slight loss in accuracy compared to the previous models, presented a similarly low inference time (0.0294 ms), which can be advantageous in applications that prioritize absolute response time. ARF, in turn, performed robustly in MAE, especially on lower-activity scenarios, and maintained an average inference time of less than 0.25 ms, positioning it as an operationally reliable solution.

Table 7 – LT (in milliseconds) for each scenario of all models.

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Mean**
AMFR	25.4279	25.2040	22.4248	27.3794	21.0466	24.2965
ARF	2.7045	2.8290	2.6005	2.8847	3.1231	2.8284
OXT	0.7679	0.7650	0.7605	0.7524	0.7725	0.7637
BaR	0.6460	0.6472	0.6484	0.6443	0.6486	0.6469
EWA	0.1811	0.1816	0.1823	0.1805	0.1805	0.1812
SRP	12.0485	10.7630	11.0537	12.3876	13.0197	11.8545
HAT	0.5998	0.5948	0.3800	0.5226	0.3535	0.4901
HTR	0.3777	0.3741	0.3759	0.3593	0.3739	0.3722
SGT	0.8871	0.7165	0.7149	0.6385	0.6985	0.7311
BLR	0.5259	0.5291	0.5291	0.5288	0.5328	0.5291
LiR	0.0360	0.0357	0.0357	0.0362	0.0361	0.0359
PAR	0.0290	0.0290	0.0292	0.0291	0.0291	0.0291
KNN	2.4989	2.4589	2.4275	2.4722	2.3712	2.4457

Source: The author.

Table 8 shows the model sizes (in kilobytes) generated in each scenario. In modern cloud computing-based architectures, model size has a direct impact on system operation, as cloud environments are built on the logic of on-demand instantiation, dynamic service replication, and artifact transfer between availability zones or regions.

In these scenarios, smaller models offer significant operational advantages; they reduce instance startup time and lower storage and transmission costs. Furthermore, because models are often packaged within containers or virtualized functions, their size directly affects the build, publishing, and activation time of these components. Therefore, while storage itself is not a significant technical limitation, model portability and light weighting are essential to ensure agility and efficiency in cloud resource management.

Table 8 – Model size (in KB) for each scenario

Model	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Mean
AMFR	1365801.02	1402583.77	1368195.70	1369088.56	1345056.62	1370145.13
ARF	7033.41	9146.78	15680.95	3875.90	5556.63	8258.73
BaR	45.09	45.09	45.09	45.09	45.09	45.09
BLR	57.45	57.45	57.45	57.45	57.45	57.45
EWA	12.96	12.96	12.96	12.96	12.96	12.96
HAT	1645.89	1478.41	1290.39	1751.04	874.98	1408.14
HTR	998.59	1202.87	1484.12	1284.21	911.80	1176.32
KNN	17.01	17.01	17.01	17.01	17.01	17.01
LiR	2.20	2.20	2.20	2.20	2.20	2.20
OXT	23600.63	24616.80	23084.65	22571.35	22947.42	23364.17
PAR	3.45	3.45	3.45	3.45	3.45	3.45
SGT	41974.62	42100.07	45453.65	43785.49	41914.20	43045.61
SRP	37191.27	40668.19	43013.92	35600.99	32111.88	37717.25

Source: The author.

The results show a wide variation in the size of the generated models. AMFR, for example, is by far the largest, averaging over 1.37 GB. This value is considerably higher than the others. It can negatively impact instance startup time or the time to publish new service versions, especially in automated pipelines with frequent build and deploy steps. Even in commercial clouds with high bandwidth and elastic resources, handling models of this size imposes an operational cost in terms of activation latency and internal network usage.

Other medium-sized models, such as ARF (approximately 8.26 MB), HAT (1.4 MB), and HTR (1.17 MB), have more moderate sizes and are already more suitable for cloud workflows, allowing them to be instantiated and replicated with less overhead.

On the other hand, most of the evaluated models are remarkably compact. EWA, for example, has a fixed size of only 12.96 KB, while BLR remains at 57.45 KB, and OXT, although larger than the previous ones, remains at around 23 MB. In addition to offering good predictive and computational performance, these models stand out for their ease of transport and deployment in distributed environments, enabling frequent updates without noticeable penalties in loading latency.

Models such as SGT (43 MB) and SRP (37 MB), although not the largest, fall within a range that may require additional attention when used in pipelines with a large volume of instancing or a high update frequency. Extremely compact models, such as LiR (2.2 KB) and PAR (3.45 KB), are also among the smallest. However, as discussed previously, their predictive performance disqualifies them for practical use, making their size an irrelevant factor in the decision.

5.1.2 Simulation Results

Tables 9 and 10 present, respectively, the NLR and FOC for each model in the different scenarios. This analysis enables us to evaluate not only the predictive effectiveness of the models in simulated environments, but also, above all, the direct impact of these predictions on the system's scalability. By jointly observing the volume of loss and the frequency of events, it is possible to infer how much each approach contributes to, or compromises, the maintenance of quality of service under different load regimes.

Table 9 – NLR per model in each scenario.

Model	S0	S1	S2	S3	S4
AMFR	2018	24	2886	49876	66016
ARF	2018	24	2886	247984	68154
BaR	N/S	N/S	2886	N/S	43672
BLR	1251	24	0	28584	43308
EWA	1251	24	2886	31541	45758
HAT	N/S	N/S	2886	N/S	44213
HTR	2018	24	2886	12541	44213
KNN	2018	24	5265	228656	151213
LiR	N/S	N/S	0	N/S	43308
OXT	2018	24	2886	56790	45029
PAR	1251	24	0	7632672	2782102
SGT	1251	24	2886	31950	59144
SRP	2018	24	2886	29497	47639

Source: The author.

In scenario 0, which represents the scenario with the lowest level of variation, the models BLR, EWA, PAR, and SGT stand out for presenting the best results from an operational point of view, in which all of them recorded only two loss events, precisely the number of failure events present in the scenario, and exhibited the lowest volumes of lost requests.

Table 10 – FOC in each scenario.

Model	S0	S1	S2	S3	S4
AMFR	3	1	1	5	16
ARF	3	1	1	24	30
BaR	N/S	N/S	1	N/S	8
BLR	2	1	0	2	7
EWA	2	1	1	3	9
HAT	N/S	N/S	1	N/S	9
HTR	3	1	1	11	9
KNN	3	1	2	21	41
LiR	N/S	N/S	0	N/S	7
OXT	3	1	1	5	7
PAR	2	1	0	1419	643
SGT	2	1	1	3	22
SRP	3	1	1	3	9

Source: The author.

This behavior becomes particularly relevant when compared with the MAE values of these models, as shown in Table 5. When comparing, it is possible to observe that although BLR obtained the lowest mean absolute error (129.34), other models with higher MAE values, such as EWA, which reached 184.21, SGT reached 417.68, and PAR, which obtained a significantly

higher value of 2673.54. Even so, they obtained the same pattern of losses in the simulator. This result suggests that, under more stable traffic conditions, the system's scalability depends less on the overall accuracy of the model and more on its ability to respond correctly to sudden changes in traffic, rather than on minimizing the absolute error over time.

In Scenario 1, significantly more uniform behavior is observed among the models, both in terms of the total number of lost requests and the number of loss events, unlike what was observed in Scenario 0, where there was greater dispersion of results and significant variations between models, in this scenario, almost all methods presented very similar performance, with 24 lost requests concentrated in a single event. The exceptions are the models absent from the simulation, such as BaR, LiR, and HAT, which could not be directly evaluated.

The smoother nature of the second failure present in this scenario can explain this homogeneous pattern. The traffic fluctuation observed here is less abrupt and has less structural impact than in Scenario 0, which contributes to more predictable operation and is less demanding on the scalability mechanism.

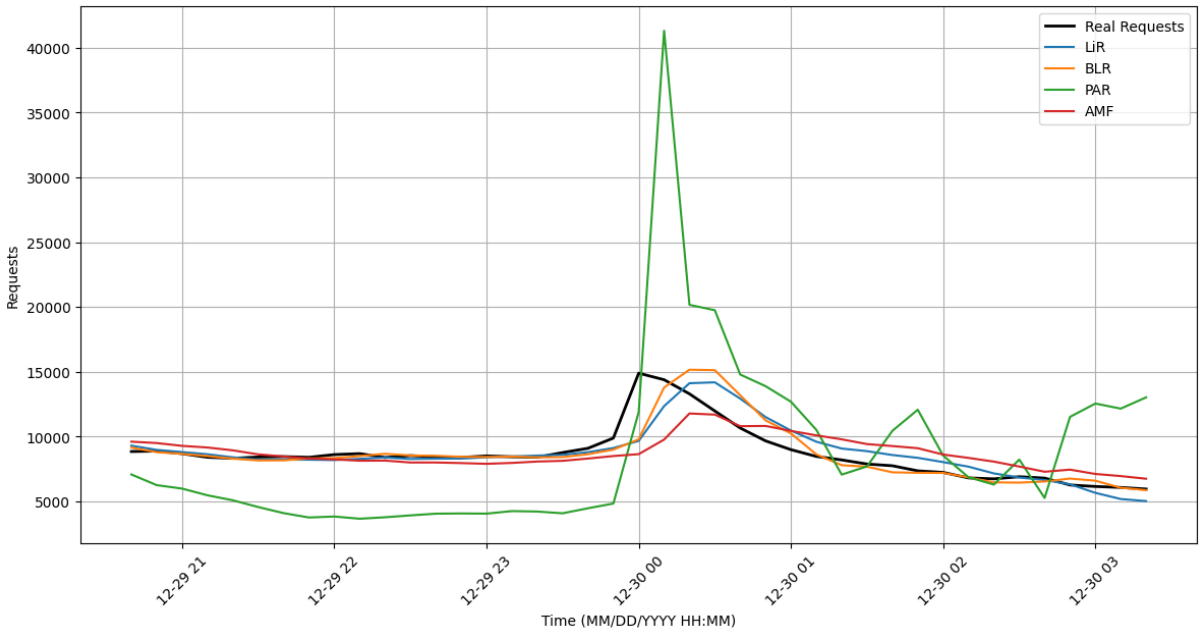
Still, the comparison between the models reinforces the pattern already observed: BLR, EWA, SGT, and PAR continue to record the same loss volumes, despite presenting significant differences in MAE values. This convergence in practical performance, despite discrepancies in the mean error, underscores that in environments with low variability and specific disturbances, system scalability is more closely tied to timely change detection than to overall forecast accuracy.

In Scenario 2, the models' behavior diversifies again. This scenario represents a more stable operational scenario, without abrupt failures or severe traffic fluctuations, allowing us to evaluate the models' effectiveness under regular conditions.

Despite this stability, the results reveal significant differences between the models in terms of the quality of scalability decisions. Models such as BLR, PAR, and LiR stood out by entirely avoiding any lost requests, recording no single failure event. This performance is especially significant considering that these models present quite different levels of mean absolute error: while BLR achieved a MAE of 148.13, PAR recorded 3313.19, one of the highest in the analysis. Even so, both models ensured adequate resource allocation over time.

This behavior is illustrated in Figure 20, which shows a comparison between actual traffic and the predictions of different models during the surge recorded at midnight on the 30th. Although the BLR model underestimated the peak, predicting approximately 9,783 requests for an actual demand of 14,886, the model exceeded the threshold of 9,600 requests, equivalent to 80% of an instance's capacity, leading to the creation of a second AMF and avoiding losses. Similarly, the PAR model, by predicting more than 11,893 requests, also ensured that the system was adequately sized at the critical moment, avoiding any service degradation.

Figure 20 – Real and Predicted Requests for Multiple Models – Scenario 2 (00:00 on Day 30)



Source: The author.

In contrast, other models, such as AMF, only react to observed traffic, making scalability decisions late in the face of the surge, resulting in the loss of 2886 requests.

In Scenario 3, where a significant increase in traffic load is observed, combined with the presence of failures, this scenario poses an especially challenging environment for prediction models. Here, methods must simultaneously handle high request volumes and occasional disruptions caused by failure events, requiring not only statistical accuracy but also rapid response to abrupt pattern transitions and resumptions.

Models such as BLR, EWA, SRP, and SGT exhibited robust performance, with total lost request volumes ranging from approximately 28,000 to 32,000, and FOC between 2 and 3. These results demonstrate that these models, despite some variation in MAE, can effectively capture critical traffic moments and ensure sufficiently responsive scalability to maintain the system operating with good quality of service.

On the other hand, models such as HTR, KNN, and OXT demonstrated clear limitations as complexity increased. HTR lost over 12,500 requests, KNN exceeded 228,000 losses in 21 events, and OXT accumulated nearly 57,000 losses. These numbers reflect a consistent difficulty for these models in maintaining predictions aligned with traffic dynamics under stress, resulting in insufficient scaling decisions.

The PAR case represents the most extreme example of degradation: the model accumulated over 7.6 million lost requests and recorded 1,419 failure events, signaling a complete collapse of the scaling. The explanation lies in its highly dispersed error pattern, with strong negative asymmetry, which leads the model to underestimate demand repeatedly. Given that

the simulator reacts directly to predictions to adjust the number of instances, these distorted predictions led to excessive instance shutdowns, even during peak load periods. As a result, the network remained severely undersized for long periods.

In Scenario 4, the results reflect a particularly demanding scenario from the perspective of traffic variability, despite the absence of explicit network failures. Unlike Scenarios 0 and 3, where the challenge was capturing abrupt transitions caused by traffic drops and resumptions, here the obstacle lies in the high dispersion and structural changes in the load pattern, which requires the models to have a greater capacity for continuous adaptation over time.

As evidenced in Table 9, the volume of lost requests increased significantly for all models. The most extreme case was PAR, which recorded more than 2.78 million lost requests, concentrated in 643 events (Table 10). This result indicates a chronic difficulty for the model in keeping up with the dynamic evolution of the load, leading to recurring undersizing decisions.

Despite the widespread difficulties, models such as BLR and OXT managed to keep the number of events relatively controlled, with seven events, despite significant loss volumes. BLR, for example, accumulated approximately 43,000 lost requests, despite maintaining one of the lowest event frequencies among the models.

KNN, on the other hand, again performed poorly, with 151,000 lost requests in 41 events, highlighting its persistent tendency to underestimate. The same occurred with ARF, which recorded 30 underestimation events, indicating recurring fluctuations in its decision-making.

This scenario highlights that, in environments with highly deviant traffic patterns, it is not sufficient for the model to be merely accurate on average; it must also be sensitive to contextual variations and maintain consistency in predicting trends. Models capable of tracking these structural changes, such as BLR, can preserve service quality with fewer fluctuations, whereas others, even with good statistical performance in more stable scenarios, ultimately fail because they are unable to keep pace with the environment's dynamics.

To better understand the impact of predictions on resource instantiation, the analysis of DIPI, which indicates the difference between the ideal number of instances and the number of AMF instances allocated, reveals how the models behave, both in terms of under-allocation and over-allocation of resources in each scenario.

In Scenario 0 (Table 11), the scenario with the lowest activity and two failure events, all models exhibited restrained behavior, with decisions predominantly made between -1 and $+1$ instances. This behavior reflects a stable environment where severe deviations in resource allocation are less likely to occur. In this context, the BLR and HTR models stand out for concentrating most of their decisions in DIPI equal to zero with 4515 and 4510 instances, respectively, thereby minimizing both losses and wasted instances. In contrast, models such as PAR, with 97 excess activations, and SGT, with 298 excess activations, indicate a more biased approach to overprovisioning, which, despite avoiding losses, compromises resource efficiency.

Table 11 – DIPI distribution for each model in Scenario 0.

Model\DIPI	-4	-3	-2	-1	0	1	2	3	4
AMFR	0	0	0	184	4385	39	0	0	0
ARF	0	0	0	87	4490	31	0	0	0
BLR	0	0	0	69	4515	24	0	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0	0	73	4493	42	0	0	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	0	67	4510	31	0	0	0
KNN	0	0	0	128	4453	27	0	0	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0	0	74	4498	36	0	0	0
PAR	0	0	0	199	4312	97	0	0	0
SGT	0	0	0	49	4261	298	0	0	0
SRP	0	0	0	70	4500	38	0	0	0

Source: The author.

In Scenario 1 (Table 12), a similar dynamic is observed, but with some important variations. The PAR model underestimated the most, with 499 decisions below demand, despite having two records with DIPI `+3`, indicating erratic behavior. KNN and SGT presented the highest volumes of overallocation, with 138 and 70 excess activations, respectively. The BLR model, once again, presented the most balanced behavior, with 4,524 exact hits and only 33 excess activations, reinforcing its good performance in scenarios with low volatility.

Table 12 – DIPI distribution for each model in Scenario 1.

Model\DIPI	-4	-3	-2	-1	0	1	2	3	4
AMFR	0	0	0	84	4427	97	0	0	0
ARF	0	0	0	75	4490	43	0	0	0
BLR	0	0	0	51	4524	33	0	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0	0	97	4468	43	0	0	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	0	77	4488	43	0	0	0
KNN	0	0	0	108	4362	138	0	0	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0	0	63	4508	37	0	0	0
PAR	0	0	0	499	4067	40	0	2	0
SGT	0	0	0	30	4508	70	0	0	0
SRP	0	0	0	80	4478	50	0	0	0

Source: The author.

In Scenario 2, Table 13, despite the absence of abrupt failures, traffic is heavier and occasionally exhibits significant variations. Analysis of DIPI shows that the PAR model, in addition to over-responding to the traffic surge on the 30th, with 632 over-allocation decisions and 36 decisions between `+2` and `+3`, also overreacted at other times. On the other hand, models such as BLR and LiR maintained their decisions with an intense concentration on DIPI equal to zero, with 4253 and 4244 cases, respectively, demonstrating a forecast more closely aligned with actual demand. These results reinforce the previous conclusion that in scenarios without failures and with a well-established traffic pattern, consistency in the predictive trend is more decisive than point accuracy.

In Scenario 3, represented in Table 14, a scenario characterized by greater instability and

Table 13 – DIPI distribution for each model in Scenario 2.

Model\DIPI	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
AMFR	0	0	0	89	4110	121	0	0	0
ARF	0	0	0	65	4196	59	0	0	0
BLR	0	0	0	39	4253	28	0	0	0
BaR	0	0	0	75	4211	34	0	0	0
EWA	0	0	0	66	4228	26	0	0	0
HAT	0	0	0	65	4215	40	0	0	0
HTR	0	0	0	56	4229	35	0	0	0
KNN	0	0	0	106	4078	136	0	0	0
LiR	0	0	0	51	4244	25	0	0	0
OXT	0	0	0	52	4231	37	0	0	0
PAR	0	0	0	632	3652	34	0	2	0
SGT	0	0	0	60	4158	102	0	0	0
SRP	0	0	0	72	4199	49	0	0	0

Source: The author.

the presence of failures is observed, which poses significant challenges to the decision-making of online scalability models. In this context, the distribution of decisions among different DIPI values becomes more dispersed, highlighting different allocation strategies when faced with abrupt traffic fluctuations.

Table 14 – DIPI distribution for each model in Scenario 3.

Model \ DIPI	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
AMFR	0	0	4	165	4341	95	3	0	0
ARF	0	0	21	157	4283	144	3	0	0
BLR	0	0	2	73	4489	42	2	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0	2	91	4394	113	6	0	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	0	296	3972	338	2	0	0
KNN	0	2	14	196	4211	175	8	2	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0	4	78	4432	92	2	0	0
PAR	0	0	510	2026	1853	186	19	4	6
SGT	0	0	2	158	4140	306	2	0	0
SRP	0	0	2	86	4409	109	2	0	0

Source: The author.

Models such as BLR, EWA, and SRP showed consistent behavior and were closer to ideal. BLR, for example, concentrated 4,489 decisions in DIPI `0` and 42 in `+1`, with a minimal number of suballocations (only 73 in `−1` and 2 in `−2`). This distribution reveals a cautious and efficient approach, maintaining stable allocations even under adverse conditions. The SRP and EWA models followed a similar pattern, with a predominance of `0` and `+1`, and a moderate number of underallocations—reinforcing their adaptability to an unstable environment.

On the other hand, the PAR model again stood out negatively, with an aggressive and volatile pattern. 510 decisions were recorded with DIPI `−2` and 2,026 with `−1`, indicating recurring underallocation in a critical scenario. Furthermore, the model exceeded the ideal demand at various points, with 215 activations above the actual need, distributed between the values `+1`, `+2`, `+3`, `+4`, and even `+5`. This oscillation reflects a model unable to find a balance between rapid response and stability, which directly contributed to the high volume of lost

requests observed previously.

Finally, in Scenario 4, represented in Table 15, Models such as BLR, EWA, SRP, and LiR maintained a more concentrated distribution in the DIPI `0` and `+1` regions, with small variations downwards or upwards. BLR, for example, recorded 3,970 decisions with optimal allocation and 71 with slight overallocation (`+1`), in addition to a reduced number of underallocations with only 99 in `−1`. This behavior reinforces its conservative and stable nature, even under greater conceptual pressure. BaR and EWA showed a similar pattern, with close values at `0` and discrete variations around it, demonstrating allocation control.

Table 15 – DIPI distribution for each model in Scenario 4 .

Model\DIPI	-5	-4	-3	-2	-1	0	1	2	3	4	7
AMFR	0	1	1	1	275	3690	175	0	0	0	0
ARF	0	1	1	27	223	3791	100	0	0	0	0
BLR	0	1	0	1	99	3970	71	0	1	0	0
BaR	0	1	0	2	117	3913	109	1	0	0	0
EWA	0	1	0	1	131	3870	139	0	1	0	0
HAT	0	1	1	1	111	3903	124	2	0	0	0
HTR	0	1	0	2	136	3871	127	6	0	0	0
KNN	0	2	0	19	242	3629	216	29	6	0	0
LiR	0	1	0	1	119	3879	142	0	1	0	0
OXT	0	1	1	1	109	3932	99	0	0	0	0
PAR	1	0	0	135	1055	2588	318	44	1	0	1
SGT	0	1	0	29	513	3296	303	1	0	0	0
SRP	0	1	1	1	147	3886	107	0	0	0	0

Source: The author.

In contrast, models such as PAR, SGT, and KNN exhibited greater dispersion in DIPI, with multiple decisions falling within the overallocation and underallocation ranges. PAR, once again, stood out negatively by presenting an extremely aggressive performance with 318 decisions with DIPI `+2`, 44 with `+3`, 1 with `+4`, and even one case of `+7`, totaling 364 overallocation decisions above the minimum necessary value. Furthermore, the model recorded one decision in `−5`, 135 decisions in `−2`, and 1,055 in `−1`, indicating, again, its pattern of instability.

The KNN model also exhibited considerable variation, with 216 decisions at `+1`, 29 at `+2`, and 242 suballocations at `−1`, in addition to less frequent values in extreme ranges. This pattern suggests a greater sensitivity of the model to local fluctuations, with more erratic decisions. SGT, although more moderate than PAR, also presented 303 decisions with DIPI `+1` and 513 with `−1`, which may indicate a tendency toward late compensations or imprecise reactions to changes in demand.

5.2 Online and Offline Learning: Performance Analysis

5.2.1 Forecasting and Computing

Table 16 presents the mean absolute error (MAE) values obtained by the LSTM model, representing the offline learning method, and the BLR model, representing the online learning

method, in the five scenarios, in addition to the overall average. The table shows that the BLR model outperforms the LSTM model in all scenarios, with an average difference of approximately 126 points in the total MAE.

Table 16 – MAE values for LSTM and BLR models across all scenarios.

Model	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Mean
LSTM	143.62	177.25	163.81	532.25	1031.95	409.78
BLR	129.34	142.41	148.13	425.73	573.70	283.86

Source: The author.

The results in scenarios 0, 1, and 2, scenarios with less variability in traffic patterns, show that BLR performs competitively with robust and established offline learning methods. The model achieves MAEs of 129.34 and 142.41 in these scenarios, while the LSTM registers higher values of 143.62 and 177.25, respectively. This behavior reinforces the competence of BLR even in scenarios where offline approaches are traditionally established.

As concept deviations increase, a significant degradation in LSTM's performance is observed compared to BLR. In scenario 3, for example, LSTM presents an MAE of 532.25, equivalent to more than 100 points above the value obtained by BLR. In scenario 4, this difference becomes even more significant, where LSTM achieves a MAE of 1031.95, almost double the value recorded by BLR. These results demonstrate the difficulties of using offline learning methods in scenarios subject to changing traffic patterns, while BLR demonstrates greater adaptability to non-stationary environments.

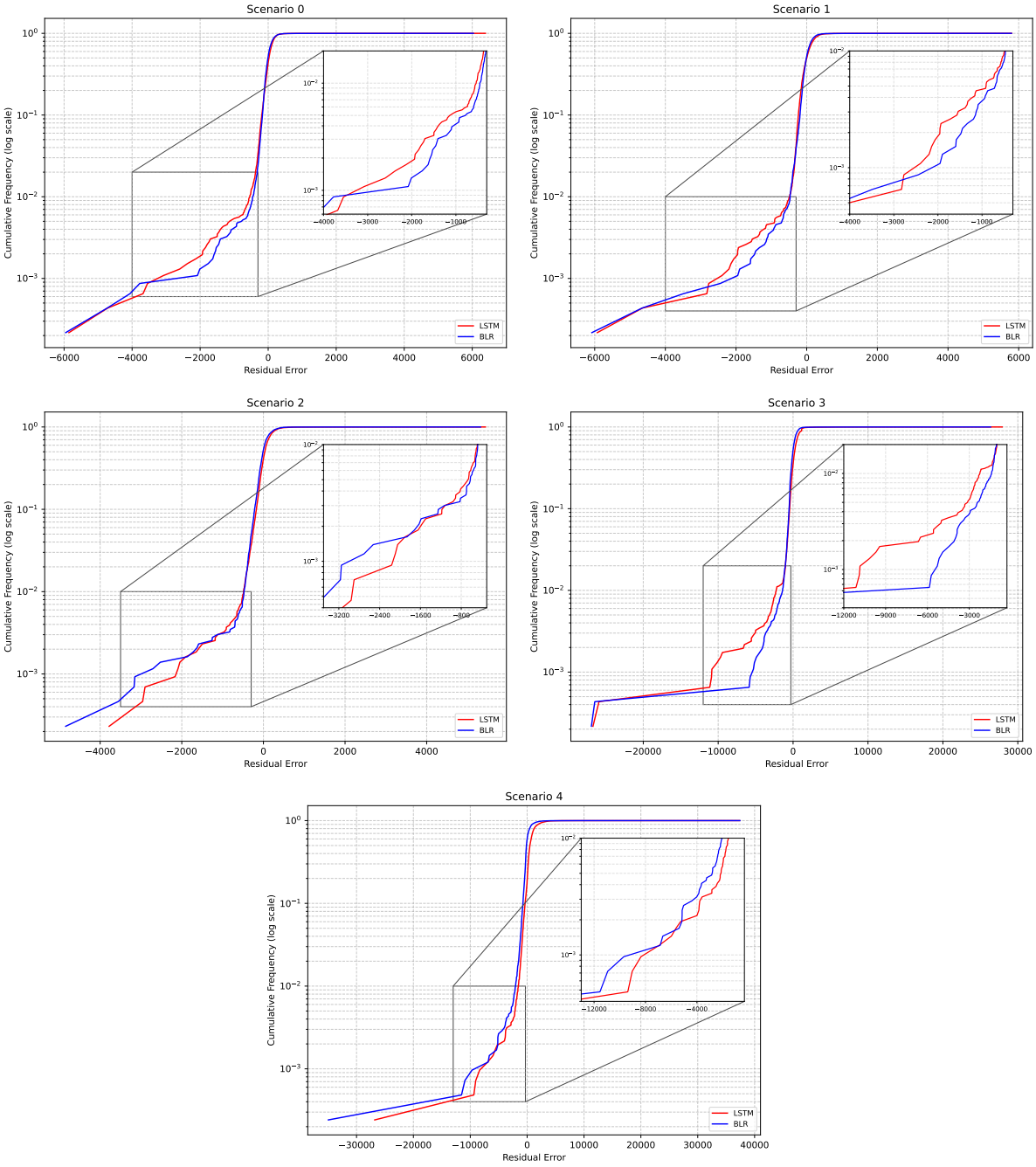
Figure 21 presents the comparison between the LSTM and BLR models using the cumulative distribution function for residual error, calculated separately for the five scenarios analyzed. Overall, a consistent pattern of BLR's superiority over LSTM is observed in most scenarios.

In scenario 0, which represents a scenario with less variability in traffic patterns, the BLR model showed superior overall performance. However, in the residual error range between approximately -3500 and -1000, its CDF curve is below that of the LSTM, indicating a higher incidence of severe underestimations. However, outside this range, the BLR maintains a higher cumulative frequency of hits with more minor errors, resulting in more accurate overall performance.

In scenarios 1 and 2, the BLR's superiority is maintained. In scenario 1, the CDF curve is consistently above that of the LSTM, reflecting less error dispersion. In scenario 2, although the curves are close, BLR maintains a slight advantage over most of the error range.

Scenario 3, on the other hand, exhibits more complex behavior. Although BLR maintains superior performance across most of the domain, especially at the extremes where its curve stabilizes more quickly, an intermediate range of residual error is observed, approximately

Figure 21 – Residual Error CDF of LSTM and BLR Models for All Scenarios



Source: The author.

between -10,000 and 0, in which the LSTM exhibits a steeper slope in the CDF curve, positioning itself above BLR. This result indicates that, within this specific range, LSTM accumulates a greater proportion of predictions with moderate errors, reflecting occasional better performance. However, as the residuals approach zero, BLR once again shows superiority, with less dispersion and a higher cumulative frequency of minor errors.

In scenario 4, BLR's advantage is significant and evident. Its CDF curve is above that of LSTM across virtually the entire domain, indicating a greater concentration of residual errors in smaller magnitude ranges. However, both models exhibit long tails on the negative side of the distribution, with errors exceeding $-20{,}000$, signaling the occurrence of severe and isolated underestimations. Nevertheless, BLR maintains better overall performance, with less dispersion and a higher cumulative frequency over much of the interval, reflecting greater robustness to data variability.

Overall, BLR demonstrated superior performance across the five scenarios analyzed, both in more stable environments and in scenarios of high variability and conceptual change. Visual analysis of the CDF curves, in conjunction with the MAE values previously discussed, reinforces the robustness and adaptability of BLR as an online learning method, capable of outperforming a complex and widely used model, such as LSTM.

Table 17 shows the IT, in milliseconds, of the LSTM and BLR models in the five scenarios analyzed. A highly significant difference can be observed between the two models. LSTM presents average inference times ranging from approximately 44.5 ms to 56.9 ms, with an overall average of 50.79 ms. In contrast, BLR presents a virtually constant inference time across all scenarios, at approximately 0.0066 ms, or around 7,600 times faster than LSTM.

Table 17 – IT (in milliseconds) for scenarios LSTM and BLR.

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Mean**
LSTM	53.6822	51.3057	56.9355	47.4867	44.5302	50.7881
BLR	0.0066	0.0066	0.0067	0.0066	0.0067	0.0066

Source: The author.

This difference is due to the computational complexity of the models. LSTM, being a recurrent neural network with multiple layers and temporal dependencies, demands more processing resources even during the inference phase. BLR, on the other hand, performs inference based on simple matrix operations, which makes it highly efficient.

Thus, although LSTM is a widely adopted model for sequential tasks, its high computational cost makes it unsuitable for streaming data scenarios that require low latency and real-time response. In these contexts, models with reduced inference times, such as BLR, are more suitable, as they enable rapid updates and predictions with minimal computational requirements. Thus, in addition to the accuracy already discussed, BLR offers a significant practical advantage in

environments under strict time constraints.

Table 18 shows the sizes, in kilobytes, of the LSTM and BLR models for each of the five scenarios considered. The LSTM model has an average size of approximately 182.08 KB, with small variations between scenarios, while the BLR model has a fixed size of 57.45 KB. This represents a savings of approximately 68% in memory usage by choosing BLR over LSTM.

Table 18 – Model size (in KB) for scenarios LSTM and BLR.

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Mean**
LSTM	179.87	182.63	182.63	182.63	182.66	182.084
BLR	57.45	57.45	57.45	57.45	57.45	57.45

Source: The author.

This difference is due to the computational complexity of the models. LSTM, being a recurrent neural network with multiple layers and temporal dependencies, demands more processing resources even during the inference phase. BLR, on the other hand, performs inference based on simple matrix operations, which makes it extremely efficient.

Thus, although LSTM is a widely adopted model for sequential tasks, its high computational cost makes it unsuitable for streaming data scenarios that require low latency and real-time response. In these contexts, models with reduced inference times, such as BLR, are more appropriate, as they allow for rapid updates and predictions with minimal computational consumption. Thus, in addition to the accuracy already discussed, BLR offers a significant practical advantage in environments under strict time constraints.

5.2.2 Simulation Results

Table 19 presents the results of the AMF simulation, considering the practical impacts of the predictions made by the LSTM and BLR models. Table 20 shows the distribution of DIPI, which represents the difference between the ideal number of AMF instances and the number effectively allocated, functioning as an indicator of the accuracy in decision-making for resource allocation and system overload.

Table 19 – NLR and FOC per model in each scenario.

Model	Metric	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4
LSTM	NLR	1251	24	2886	28584	45758
	FOC	2	1	1	2	9
BLR	NLR	1251	24	0	28584	43308
	FOC	2	1	0	2	7

Source: The author.

In scenarios 0 and 1, where the traffic pattern is more stable, both models presented similar performance in terms of NLR and FOC, with only two failure events and a low number of

Table 20 – DIPI distribution for LSTM and BLR models across scenarios.

Scenario	Model\DIPI	-4	-2	-1	0	1	2	3
0	LSTM	0	0	50	4523	35	0	0
	BLR	0	0	69	4515	24	0	0
1	LSTM	0	0	66	4514	28	0	0
	BLR	0	0	51	4524	33	0	0
2	LSTM	0	0	62	4226	32	0	0
	BLR	0	0	39	4253	28	0	0
3	LSTM	0	2	172	4396	36	2	0
	BLR	0	2	73	4489	42	2	0
4	LSTM	1	2	261	3844	34	1	0
	BLR	1	1	99	3970	71	0	1

Source: The author.

lost requests. Means that in contexts with low variability, both models can maintain a comparable level of efficient operation. However, when observing the DIPI values, it is noted that, in scenario 0, BLR tends to slightly underestimate demand, with more occurrences in DIPI -1 (69 versus 50 for LSTM), which suggests a greater risk of overloading the AMF instances, behavior consistent with that observed in the CDF curves in Figure 21. In scenario 1, this trend is reversed, with LSTM presenting more episodes of suballocation, having 66 events of DIPI -1, compared to 51 of BLR, indicating a loss of sensitivity even under still moderate variations.

Starting with scenario 2, more notable differences between the models begin to emerge. Although this scenario still represents a scenario with discrete changes, it was enough to cause a failure event in LSTM, resulting in 2,886 lost requests. BLR, on the other hand, did not register any losses in this scenario. Analyzing the profile of DIPI, we observe a growing trend of underallocation in LSTM, with 62 events in DIPI -1, compared to only 39 in BLR, which reinforces its lower ability to adjust to fluctuations in demand. These results indicate a tendency for LSTM to be compromised by the lack of adaptive mechanisms, even when faced with subtle variations in traffic patterns, resulting in overloaded instances and performance losses.

In scenario 3, characterized by greater unpredictability and frequent conceptual changes, both models presented the same number of lost requests (28,584). Although this result may initially suggest performance equivalence, a more in-depth analysis reveals relevant differences in the operational behavior of each approach. LSTM recorded 172 occurrences of DIPI -1, more than double the 73 observed in BLR, indicating a persistent trend of underallocation and, consequently, overload in the AMF instances. This pattern is consistent with the mean absolute error (MAE) values observed in this scenario, in which LSTM underperforms BLR (532.25 versus 425.73), as shown in Table 16. Thus, even if the loss metric does not indicate immediate differences, the overall performance reveals that the system operating with LSTM is subject to greater operational stress, which can compromise its stability and scalability in the long term. The online model, in turn, demonstrates greater efficiency, promoting more consistent allocations aligned with the variable demands of the environment.

Scenario 4 presents the most demanding scenario among those analyzed, characterized

by heavy traffic. In this context, the differences between the models become even more evident. BLR outperformed LSTM in all aspects considered: it recorded approximately 2,450 fewer lost requests (43,308 versus 45,758), in addition to a slightly lower failure frequency, with FOC 7 versus 9. The analysis of DIPI reinforces this advantage. Although both models presented one case of severe underallocation (DIPI -4), LSTM demonstrated a greater concentration of critical underallocation levels, with two occurrences in DIPI -2 and 261 in DIPI -1, compared to only 1 and 99, respectively, in BLR.

These results indicate that, under conditions of greater conceptual drift, the LSTM model begins to operate significantly more stressed, with a higher frequency of underallocation and overloaded instances. As discussed previously, this behavior is consistent with the trend that, as environmental changes become more pronounced, performance degradation in non-adaptive models tends to intensify. On the other hand, BLR can track these transformations with greater accuracy by incorporating continuous update mechanisms, resulting in more stable allocation decisions that align with operational reality.

5.3 Analysis of Confidence Intervals and Empirical Error Distributions

As described in Section 4.6, experimental uncertainty was quantified using two complementary approaches. For continuous performance metrics (e.g., MAE, Inference and Training time, model size, Number of Lost Requests, and Full Occupancy Count), results are presented as mean values with 95% confidence intervals over 30 executions. This allows assessing both central tendency and statistical stability of each model across scenarios.

In contrast, for the discrete DIPI measure, which captures deviations in instance allocation, uncertainty is represented by empirical error distributions. These distributions reveal not only the overall accuracy of each model but also their tendencies toward under- or over-provisioning under different traffic conditions. The following tables summarize these results, organized by metric and scenario.

Table 21 reports the mean absolute error (MAE) with 95% confidence intervals across all scenarios for each evaluated model. Lower values indicate better predictive performance, with narrower intervals suggesting more stable results. It is worth noting that for the PAR model, an overflow issue occurred in Scenario 1, as reflected in the extremely high MAE values. For this case, 7 overflow instances were observed, and the confidence interval was calculated using only the remaining 23 executions.

Table 21 – MAE with 95% confidence intervals across scenarios for all models

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
AMFR	248.30 ± 1.22	298.60 ± 1.47	306.86 ± 1.34	798.70 ± 4.68	1395.79 ± 7.53
ARF	188.89 ± 9.90	196.91 ± 9.14	172.52 ± 0.78	999.67 ± 65.45	1123.98 ± 13.25
BLR	135.95 ± 0.50	148.82 ± 0.49	155.57 ± 0.55	457.60 ± 2.31	608.87 ± 2.54
BaR	5.72E12 ± 1.38E11	3.51E12 ± 1.33E11	186.75 ± 0.51	6.63E13 ± 1.27E12	791.15 ± 2.62
EWA	190.22 ± 0.45	206.45 ± 0.47	206.19 ± 0.48	682.26 ± 2.10	889.86 ± 2.15
HAT	1.29E13 ± 4.09E11	1.41E10 ± 3.49E09	190.98 ± 0.65	1.75E14 ± 2.63E13	902.75 ± 2.60
HTR	361.86 ± 2.00	392.45 ± 0.86	191.07 ± 0.59	1799.80 ± 2.11	906.75 ± 3.66
KNN	322.49 ± 4.30	396.70 ± 4.84	314.35 ± 2.13	1101.34 ± 14.70	1645.48 ± 22.85
LSTM	145.93 ± 2.78	166.32 ± 1.81	162.44 ± 1.38	534.39 ± 10.72	956.68 ± 33.91
LiR	4.49E13 ± 4.52E11	4.56E13 ± 2.69E11	189.33 ± 0.54	4.15E14 ± 7.34E11	854.98 ± 4.84
OXT	184.91 ± 1.58	195.30 ± 1.43	175.07 ± 0.65	725.43 ± 10.07	751.48 ± 4.37
PAR	2689.97 ± 8.33	1.54E26 ± 3.02E26	1.8E07 ± 2.4E07	10060.96 ± 18.48	15771.36 ± 107.83
SGT	420.81 ± 0.59	485.92 ± 0.47	363.75 ± 0.85	1388.35 ± 2.39	2803.26 ± 3.35
SRP	214.06 ± 1.15	233.50 ± 1.13	198.26 ± 0.55	930.48 ± 9.72	911.38 ± 3.28

Source: The author.

Table 22 presents the inference time (IT, in milliseconds) with 95% confidence intervals across all scenarios for each model. These results provide insights into the computational efficiency of the models during prediction.

Table 22 – IT (ms) with 95% confidence intervals across scenarios for all models

Model	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4
AMFR	3.86 ± 0.26	3.34 ± 0.19	3.18 ± 0.29	3.51 ± 0.18	2.89 ± 0.14
ARF	0.15 ± 0.00	0.16 ± 0.01	0.16 ± 0.01	0.15 ± 0.01	0.15 ± 0.01
BLR	0.0064 ± 0.00	0.0064 ± 0.00	0.0064 ± 0.00	0.0064 ± 0.00	0.0064 ± 0.00
BaR	0.147 ± 0.002	0.149 ± 0.002	0.159 ± 0.004	0.162 ± 0.001	0.159 ± 0.001
EWA	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
HAT	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00
HTR	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.01 ± 0.00
KNN	2.01 ± 0.01	2.00 ± 0.01	2.00 ± 0.01	2.05 ± 0.01	1.98 ± 0.01
LSTM	47.26 ± 1.00	45.94 ± 0.55	46.30 ± 0.72	45.91 ± 0.29	45.87 ± 0.47
LiR	0.0038 ± 0.00	0.0038 ± 0.00	0.0037 ± 0.00	0.0037 ± 0.00	0.0037 ± 0.00
OXT	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00
PAR	0.006 ± 0.002	0.005 ± 0.00	0.005 ± 0.00	0.005 ± 0.00	0.005 ± 0.00
SGT	0.006 ± 0.00	0.006 ± 0.00	0.006 ± 0.00	0.006 ± 0.00	0.006 ± 0.00
SRP	0.20 ± 0.01	0.20 ± 0.00	0.20 ± 0.01	0.20 ± 0.00	0.20 ± 0.01

Source: The author.

Table 23 reports the training time (LT, in milliseconds) with 95% confidence intervals across all scenarios. The results highlight the computational cost associated with model updates in a streaming setting.

Table 23 – LT (ms) with 95% confidence intervals across scenarios for all models

Model	Scenario 0	Scenario 1	Scenario 2	Scenario 3	Scenario 4
AMFR	21.58 ± 0.71	20.80 ± 0.75	21.43 ± 0.58	21.70 ± 0.41	20.25 ± 0.17
ARF	2.37 ± 0.09	2.34 ± 0.04	2.13 ± 0.04	2.29 ± 0.05	2.52 ± 0.06
BLR	5.085E-01 ± 1.295E-03	5.094E-01 ± 1.510E-03	5.089E-01 ± 1.502E-03	5.092E-01 ± 1.469E-03	5.093E-01 ± 1.671E-03
BaR	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00
EWA	0.17 ± 0.00	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.00	0.16 ± 0.00
HAT	0.50 ± 0.01	0.48 ± 0.02	0.33 ± 0.01	0.44 ± 0.01	0.31 ± 0.01
HTR	0.33 ± 0.00	0.32 ± 0.00	0.33 ± 0.00	0.30 ± 0.00	0.33 ± 0.00
KNN	2.35 ± 0.02	2.33 ± 0.01	2.31 ± 0.01	2.37 ± 0.01	2.25 ± 0.01
LiR	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
OXT	0.61 ± 0.01	0.61 ± 0.01	0.60 ± 0.01	0.60 ± 0.01	0.62 ± 0.01
PAR	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
SGT	0.70 ± 0.02	0.68 ± 0.01	0.65 ± 0.01	0.61 ± 0.01	0.68 ± 0.01
SRP	10.31 ± 0.12	10.52 ± 0.03	10.54 ± 0.06	9.87 ± 0.17	10.38 ± 0.18

Source: The author.

Model size (Tabela 24): Table 24 summarizes the estimated in-memory size of each model (in kilobytes) with 95% confidence intervals across scenarios. The reported values correspond to the memory required to store the internal data structures of each trained model, rather than the disk space of serialized files.

Table 24 – Model size (KB) with 95% confidence intervals across scenarios for all models

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
AMFR	1E+06 ± 5.01E+03	1E+06 ± 4+03	1E+06 ± 4E+03	1E+06 ± 3E+03	1E+06 ± 3E+03
ARF	9103.11 ± 623.58	9703.17 ± 500.34	14464.14 ± 273.12	4555.86 ± 414.05	4701.79 ± 330.61
BLR	57.45 ± 0.00	57.45 ± 0.00	57.45 ± 0.00	57.45 ± 0.00	57.45 ± 0.00
BaR	45.09 ± 0.00	45.09 ± 0.00	45.09 ± 0.00	45.09 ± 0.00	45.09 ± 0.00
EWA	12.96 ± 0.00	12.96 ± 0.00	12.96 ± 0.00	12.96 ± 0.00	12.96 ± 0.00
HAT	1570.47 ± 132.88	1234.70 ± 225.64	1313.73 ± 21.39	1641.34 ± 196.47	850.17 ± 14.08
HTR	1050.63 ± 47.57	1104.55 ± 22.61	1378.32 ± 42.16	1240.69 ± 32.07	862.07 ± 13.58
KNN	17.01 ± 0.00	17.01 ± 0.00	17.01 ± 0.00	17.01 ± 0.00	17.01 ± 0.00
LSTM	192.00 ± 0.01	192.02 ± 0.00	192.05 ± 0.03	192.00 ± 0.00	192.20 ± 0.00
LiR	2.20 ± 0.00	2.20 ± 0.00	2.20 ± 0.00	2.20 ± 0.00	2.20 ± 0.00
OXT	2.34E+04 ± 213.26	2.37E+04 ± 223.12	2.30E+04 ± 215.86	2.28E+04 ± 176.31	2.24E+04 ± 187.13
PAR	3.45 ± 0.00	3.45 ± 0.00	3.45 ± 0.00	3.45 ± 0.00	3.45 ± 0.00
SGT	4.19E+04 ± 230	4.36E+04 ± 204	4.38E+04 ± 443	4.32E+04 ± 305	4.20E+04 ± 318
SRP	3.90E+04 ± 413.25	4.14E+04 ± 449.54	4.27E+04 ± 451.63	3.63E+04 ± 367.01	3.22E+04 ± 244.12

Source: The author.

Table 25 presents the total network losses (NLR) with 95% confidence intervals across scenarios for each model. Lower values are desirable, indicating fewer misallocated resources under the tested conditions. For the PAR model, only the executions without overflow were considered when computing these values.

Table 25 – NLR with 95% confidence intervals across scenarios for all models

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
AMFR	2036.67 ± 24.26	45.87 ± 12.12	3132.07 ± 258.10	54050.43 ± 2226.66	66829.17 ± 2592.43
ARF	2036.67 ± 24.26	120.63 ± 143.92	2893.97 ± 14.76	208506.00 ± 36811.23	68697.23 ± 3818.34
BLR	1265.83 ± 18.09	45.87 ± 12.12	0.00 ± 0.00	28598.10 ± 79.30	43215.47 ± 206.24
BaR	–	–	2893.97 ± 14.76	–	43457.20 ± 275.62
EWA	1265.83 ± 18.09	45.87 ± 12.12	1829.37 ± 506.73	31539.03 ± 98.60	45487.97 ± 295.76
HAT	–	–	2893.97 ± 14.76	–	44539.67 ± 243.49
HTR	2036.67 ± 24.26	45.87 ± 12.12	2893.97 ± 14.76	13038.80 ± 458.74	44539.67 ± 243.49
KNN	2036.67 ± 24.26	46.77 ± 12.42	5362.60 ± 195.07	156473.20 ± 8160.35	195564.63 ± 13987.37
LSTM	1265.83 ± 18.09	45.87 ± 12.12	2893.97 ± 14.76	28620.23 ± 91.39	44602.17 ± 1319.15
LiR	–	–	191.23 ± 260.47	–	43328.47 ± 235.03
OXT	2036.67 ± 24.26	45.87 ± 12.12	2799.60 ± 189.74	40954.57 ± 2476.21	43384.70 ± 402.17
PAR	1253.83 ± 22.14	546.14 ± 163.93	413.75 ± 382.30	7585357.50 ± 27126.41	2797386.13 ± 14139.70
SGT	1340.83 ± 82.50	45.87 ± 12.12	2893.97 ± 14.76	31527.40 ± 421.25	59558.30 ± 435.05
SRP	2036.67 ± 24.26	45.87 ± 12.12	2893.97 ± 14.76	29571.50 ± 193.66	48377.57 ± 751.94

Source: The author.

Table 26 presents the number of Full Occupancy Count (FOC) observed for each model, with 95% confidence intervals across all scenarios. This metric reflects how frequently the models generated infeasible allocations during execution. For the PAR model, only scenarios without overflow were considered in order to ensure comparability.

Table 26 – FOC with 95% confidence intervals across scenarios for all models

Model	**Scenario 0**	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
AMFR	3.00 ± 0.00	0.87 ± 0.12	1.10 ± 0.11	5.00 ± 0.16	15.63 ± 0.95
ARF	3.00 ± 0.00	1.33 ± 0.86	1.00 ± 0.00	26.17 ± 4.88	27.23 ± 2.45
BLR	2.00 ± 0.00	0.87 ± 0.12	0.00 ± 0.00	2.00 ± 0.00	6.73 ± 0.16
BaR	–	–	1.00 ± 0.00	–	6.97 ± 0.27
EWA	2.00 ± 0.00	0.87 ± 0.12	0.63 ± 0.18	3.00 ± 0.00	8.60 ± 0.24
HAT	–	–	1.00 ± 0.00	–	8.93 ± 0.19
HTR	3.00 ± 0.00	0.87 ± 0.12	1.00 ± 0.00	11.17 ± 0.36	8.97 ± 0.20
KNN	3.00 ± 0.00	0.93 ± 0.16	2.03 ± 0.07	14.37 ± 0.95	48.80 ± 2.02
LSTM	2.00 ± 0.00	0.87 ± 0.12	1.00 ± 0.00	2.03 ± 0.07	7.70 ± 1.02
LiR	–	–	0.07 ± 0.07	–	6.77 ± 0.18
OXT	3.00 ± 0.00	0.87 ± 0.12	0.97 ± 0.07	6.10 ± 0.36	5.90 ± 0.32
PAR	2.00 ± 0.00	1.52 ± 0.26	0.14 ± 0.13	1406.17 ± 3.91	641.47 ± 2.80

Source: The author.

Table 27 reports the empirical error distribution of DIPI for Scenario 0. In this low-variability setting, most models concentrated more than 95% of their decisions on the ideal allocation (`0`), with occasional deviations limited to `−1` or `+1`. Models such as BLR, HTR, and ARF exhibited particularly stable behavior, exceeding 97% of exact matches. By contrast, PAR and especially SGT presented a higher incidence of overallocation, with 2.11% and 6.47% of cases at `+1`, respectively, suggesting a bias towards resource overprovisioning even under stable conditions.

Table 27 – DIPI error distribution per model in Scenario 0.

Model\Error	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**
AMFR	0	0	0	3.9930	95.1605	0.8463	0	0	0
ARF	0	0	0	1.8880	97.4392	0.6727	0	0	0
BLR	0	0	0	1.4973	97.9817	0.5208	0	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0	0	1.5842	97.5043	0.9114	0	0	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	0	1.4540	97.8732	0.6727	0	0	0
KNN	0	0	0	2.7777	96.6328	0.5859	0	0	0
LSTM	0	0	0	1.085	98.1553	0.7595	0	0	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0	0	1.605	97.6128	0.7812	0	0	0
PAR	0	0	0	4.3185	93.5763	2.1050	0	0	0
SGT	0	0	0	1.0634	92.4696	6.4670	0	0	0
SRP	0	0	0	1.5191	97.6563	0.8247	0	0	0

Source: The author.

Table 28 presents the DIPI error distribution for Scenario 1. Although the majority of models maintained high concentrations around the ideal allocation, some deviations are notable. BLR once again stood out as the most consistent, with over 98% of decisions at `0`. In contrast, PAR exhibited more erratic behavior, with 10.83% of decisions in the `−1` range and 8.68% at `+1`. On the other hand, LSTM displayed a moderate dispersion, with 1.43% of allocations at `−1`,

97.96% at `0`, and 0.61% at `+1`, showing a slightly more varied performance compared to other models.

Table 28 – DIPI error distribution per model in Scenario 1.

Model\Error	-4	-3	-2	-1	0	1	2	3	4
AMFR	0	0	0	1.8229	96.0720	2.1050	0	0	0
ARF	0	0	0	1.6276	97.4392	0.9332	0	0	0
BLR	0	0	0	1.1068	98.1771	0.7161	0	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0	0	2.1050	96.9618	0.9332	0	0	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	0	1.6710	97.3958	0.9332	0	0	0
KNN	0	0	0	2.3438	94.6615	2.9948	0	0	0
LSTM	0	0	0	1.4323	97.9601	0.6076	0	0	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0	0	1.3672	97.8299	0.8030	0	0	0
PAR	0	0	0	10.8290	88.2595	0.8681	0	0.0434	0
SGT	0	0	0	0.6510	97.8299	1.5191	0	0	0
SRP	0	0	0	1.7361	97.1788	1.0851	0	0	0

Source: The author.

Table 29 shows the empirical distribution of DIPI for Scenario 2. In this stable traffic regime, most models remained highly accurate, with more than 97% of decisions concentrated at `0` and minimal dispersion across other values. BLR, LiR, and EWA are noteworthy for exceeding 98% of exact allocations. However, both LSTM and PAR exhibited some deviations: LSTM distributed 1.44% of decisions to `−1` and 0.74% to `+1`, showing a moderate level of dispersion around the ideal allocation. On the other hand, PAR underestimated demand in 14.63% of cases, emphasizing a more significant deviation from the ideal.

Table 29 – DIPI error distribution per model in Scenario 2.

Model\Error	-4	-3	-2	-1	0	1	2	3	4
AMFR	0	0	0	2.0602	95.1389	2.8009	0	0	0
ARF	0	0	0	1.5046	97.1296	1.3657	0	0	0
BLR	0	0	0	0.9028	98.4491	0.6481	0	0	0
BaR	0	0	0	1.7361	97.4769	0.7870	0	0	0
EWA	0	0	0	1.5278	97.8704	0.6019	0	0	0
HAT	0	0	0	1.5046	97.5694	0.9259	0	0	0
HTR	0	0	0	1.2963	97.8935	0.8102	0	0	0
KNN	0	0	0	2.4537	94.3981	3.1481	0	0	0
LSTM	0	0	0	1.4352	97.8241	0.7407	0	0	0
LiR	0	0	0	1.1806	98.2407	0.5787	0	0	0
OXT	0	0	0	1.2037	97.9398	0.8565	0	0	0
PAR	0	0	0	14.6296	84.5370	0.7870	0	0.0463	0
SGT	0	0	0	1.3889	96.2500	2.3611	0	0	0
SRP	0	0	0	1.6667	97.1991	1.1343	0	0	0

Source: The author.

Table 30 details the DIPI error distribution for Scenario 3, which combines high traffic load with failures, producing the broadest error dispersion observed. While models such as BLR, EWA, and SRP concentrated over 95% of decisions in the $-1, 0, +1$ range, ensuring relatively stable allocations, others struggled to adapt. HTR and KNN displayed higher frequencies of underprovisioning (`−1`), whereas LSTM showed moderate dispersion, with 95.40% of allocations at `0`, 3.73% at `−1`, and 0.78% at `+1`. The most critical case is PAR, which severely underestimated

demand in more than half of the windows (−1 and −2 combined), confirming its instability under adverse conditions.

Table 30 – DIPI error distribution per model in Scenario 3.

Model\Error	-3	-2	-1	0	1	2	3	4	5
AMFR	0	0.0868	3.5807	94.2057	2.0616	0.0651	0	0	0
ARF	0	0.4557	3.4071	92.9470	3.1250	0.0651	0	0	0
BLR	0.0	0.0434	1.5842	97.4175	0.9115	0.0434	0	0	0
BaR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
EWA	0	0.0434	1.9748	95.3559	2.4523	0.1302	0	0.0434	0
HAT	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
HTR	0	0	6.4236	86.1979	7.3351	0.0434	0	0	0
KNN	0.0434	0.3038	4.2535	91.3845	3.7977	0.1736	0.0434	0	0
LSTM	0	0.0434	3.7326	95.3993	0.7813	0.0434	0	0	0
LiR	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s	n/s
OXT	0	0.0868	1.6927	96.1806	1.9965	0.0434	0	0	0
PAR	0	11.0677	43.9670	40.2127	4.0365	0.4123	0.0868	0.0868	0.1302
SGT	0	0.0434	3.4288	89.8438	6.6406	0.0434	0	0	0
SRP	0	0.0434	1.8663	95.6814	2.3655	0.0434	0	0	0

Source: The author.

Table 31 presents the DIPI error distribution for Scenario 4, characterized by structural changes and high volatility. Even under these demanding conditions, models such as BLR, EWA, and SRP maintained stable behavior, with more than 93% of decisions at the optimal allocation (0) and limited dispersion around it. Conversely, several models revealed significant degradation: SGT showed 12.38% of underallocations at −1, while PAR displayed a strongly imbalanced distribution, with 25.46% of decisions at −1, 62.47% at 0, and 7.68% at +1, indicating a high level of instability. LSTM, on the other hand, exhibited moderate dispersion, with 6.30% of decisions at −1, 92.78% at 0, and 0.82% at +1, reflecting a relatively stable but slightly erratic allocation pattern.

5.4 Chapter Summary

Analysis of the results obtained with online models highlights their high potential for practical applications, especially in systems that require scalability and continuous adaptation. Several models demonstrated competitive performance in terms of both predictive accuracy and operational response, reinforcing the viability of these approaches in real-world environments.

However, the experiments also reveal that good statistical metrics, such as MAE, do not, by themselves, guarantee robust performance in telecommunications scenarios. For example, in scenario 2, models with drastically different MAEs, such as BLR (148.13), LiR (182.47), and PAR (3313.19), presented the same performance in terms of request loss and failure events. The explanation lies in the ability of these models to respond quickly to traffic growth, even if their predictions were not perfectly accurate.

Based on these data, it is possible to infer that temporal metrics such as LT and IT play important roles. A model that is slow to adjust to new traffic dynamics tends to react too late to

Table 31 – DIPI error distribution per model in Scenario 4.

Model\Error	-5	-4	-3	-2	-1	0	1	2	3	4	7
AMFR	0	0.0241	0.0241	0.0241	6.6377	89.0658	4.2239	0	0	0	0
ARF	0	0.0241	0.0241	0.6517	5.3825	91.5037	2.4137	0	0	0	0
BLR	0	0.0241	0	0.0241	2.3895	95.8242	1.7137	0	0.0241	0	0
BaR	0	0.0241	0	0.0482	2.8240	94.4484	2.6309	0.0241	0	0	0
EWA	0	0.0241	0	0.0241	3.1619	93.4105	3.3550	0	0.0241	0	0
HAT	0	0.0241	0.0241	0.0241	2.6792	94.2070	2.9930	0.0482	0	0	0
HTR	0	0.0241	0	0.0482	3.2826	93.4347	3.0654	0.1448	0	0	0
KNN	0	0.0482	0	0.4586	5.8411	87.5935	5.2136	0.6999	0.1448	0	0
LSTM	0	0.0241	0	0.0482	6.2997	92.7830	0.8206	0.0241	0	0	0
LiR	0	0.0241	0	0.0241	2.8723	93.6278	3.4274	0	0.0241	0	0
OXT	0	0.0241	0.0241	0.0241	2.6309	94.9070	2.3895	0	0	0	0
PAR	0.0241	0	0	3.2585	25.4646	62.4668	7.6755	1.0620	0.0241	0	0.0241
SGT	0	0.0241	0	0.6999	12.3823	79.5558	7.3135	0.0241	0	0	0
SRP	0	0.0241	0.0241	0.0241	3.5481	93.7967	2.5826	0	0	0	0

Source: The author.

peaks, failing to anticipate the necessary measures to avoid service degradation. Furthermore, even when the model detects imminent load growth, a long inference time can result in delayed operational responses, compromising practical performance. Therefore, evaluating models in scalable systems requires considering not only their accuracy but also their ability to react quickly to load transitions.

Another crucial aspect concerns the size and portability of models. In cloud-native architectures, with on-demand instantiation and continuous replication, overly large models can compromise the overall agility and scalability of the system. Models such as SRP and OXT, although among the best in terms of performance and accuracy, are pretty significant, over 23 MB and 37 MB, respectively. In environments with high turnover and geographic distribution, this transportation and activation cost can be a limiting factor.

Regarding offline learning models. By comparing the LSTM and BLR models in different traffic scenarios, it was possible to extract a series of lessons about the implications of choosing between these different approaches in the proactive scaling task of AMF.

The results demonstrated that, although LSTM is widely adopted and performs satisfactorily in stable contexts, its non-adaptive nature imposes significant limitations in environments subject to temporal variations and conceptual changes. The offline model's performance progressively deteriorates as the traffic pattern deviates from that observed during training, compromising the accuracy of predictions and the quality of allocation decisions.

This behavior was evident in both global error metrics (MAE) and operational indicators such as DIPI, which revealed recurring patterns of underallocation and overload in the instances managed by LSTM. On the other hand, BLR demonstrated greater resilience and adaptability by incorporating a continuous update mechanism, resulting in more consistent predictions and reduced stress, even under conditions of concept drift or high variability.

Another important aspect concerns computational efficiency. BLR not only outperformed LSTM in terms of predictive accuracy, but also demonstrated drastically shorter inference times and significantly less memory footprint. This combination of lightweight and robustness makes it particularly suitable for scenarios where decisions need to be made with low latency, such as in data-driven architectures and edge computing environments.

Besides computational efficiency, it is essential to consider the direct impact on the users' quality of experience. In this regard, the adoption of the online BLR model brought perceptible improvements to the end user. The study presented in (MENG et al., 2023) shows that approximately 45% of all signaling messages correspond to *Service Requests*, responsible for initiating or reestablishing data sessions. The loss of this type of message means that the user cannot start applications or resume real-time communications, directly compromising the perceived responsiveness of the network. In the evaluated scenarios, the lower loss rate achieved by BLR translates into additional successful connections. In the most challenging scenario,

compared to the offline learning model, the difference of about 2,450 unserved requests between the models is equivalent to approximately 1,100 *Service Requests* that did not fail, preserving continuity of use even under overload conditions.

Complementarily, the *S1 Connection Releases* account for almost half of the signaling volume and determine the termination of active connections. Failures in processing these events can generate significant side effects: inadequate resource release results in delays in resuming sessions and increases signaling load, as new connection requests need to be processed. The greater robustness of BLR in overload scenarios, evidenced by FOC recorded, helps mitigate this type of instability. This means fewer episodes of complete saturation, in which the network stops responding to new requests, reducing both the likelihood of retransmission storms and the direct impacts on users' perceived service continuity.

In summary, the results demonstrate that ensuring effective proactive scalability of AMF requires more than average accuracy: it is essential to have models that are responsive, adaptive, and operationally efficient. In this sense, BLR stands out as a promising alternative, capable of sustaining system operation even in the face of continuous changes in network conditions.

Nevertheless, it is important to acknowledge that LSTM can still be suitable in specific scenarios. Due to its ability to capture long-term dependencies in time series, the model tends to perform well when exposed to large volumes of data and stable patterns over extended periods. In this study, the evaluation considered only a two-month traffic window; in broader horizons, it is plausible that LSTM could better capture long-term and subtle trends, reinforcing its potential in large-scale temporal analyses.

Furthermore, LSTM can benefit from periodic retraining. Incorporating new data after concept drifts would allow the model to update its parameters and recover part of the accuracy lost in dynamic contexts. Thus, while LSTM shows limitations in the face of abrupt and non-adaptive changes, future investigations into retraining strategies could reveal a more competitive performance, especially in environments where longer time windows for model updates are feasible.

6 CONCLUSION

In this dissertation, we address the problem of proactively scaling functions in the core of mobile networks from the perspective of concept drift, with a focus on AMF function. This problem is particularly relevant because next-generation telecommunications networks impose stringent requirements for reliability, latency, and availability. However, as new use cases emerge and user behavior evolves, statistical patterns of network traffic change, causing the accuracy of conventional predictive models to degrade.

To address this challenge, we investigated the use of online learning models as a more adaptive approach, capable of continuously updating their predictions and responding efficiently to changes in network behavior. In total, fourteen models were analyzed using different approaches, including incremental trees, adaptive ensembles, and probabilistic regressions. The best-performing model was then compared to an offline learning model, allowing us to evaluate the advantages of continuous learning over a consolidated model in this scenario.

To evaluate the models, a simulation environment representative of the 5G network core was built. This environment enabled the reproduction of scenarios with different types of conceptual deviation, including abrupt, gradual, and recurring changes in traffic patterns. Each model was trained based on a specific traffic pattern and subsequently evaluated for its ability to accurately predict demand under different patterns, thereby reflecting the model's adaptability to real-time changes in data distribution. The generated predictions were used to trigger real-time AMF scaling decisions. Evaluation metrics included mean absolute error, inference time, learning time, and operational impact, such as The total number of requests not served and difference between ideal and predicted AMF instances. This experimental approach enabled a comprehensive analysis of the models' predictive, computational, and functional performance under realistic and dynamic conditions.

The results indicate that adaptive approaches are best suited for telecommunications scenarios subject to constant variations in traffic patterns. Although traditional offline learning methods perform satisfactorily in low-variation scenarios, their performance deteriorates significantly as changes in network behavior become more pronounced. This sharp decline in predictive accuracy results in service degradation, evidenced by an increase in the number of dropped requests and overloaded network functions.

Furthermore, experiments have shown that classic metrics, such as the mean absolute error (MAE), are insufficient for thoroughly assessing the performance of online models. In telecommunications environments, the model's ability to respond quickly to abrupt variations, such as request surges, is critical. In this context, metrics such as inference time and training time become equally relevant, as they directly influence the model's agility in adapting its predictions

and maintaining the network's efficiency.

6.1 Contributions

This dissertation performed an in-depth comparative analysis of fourteen online learning models applied to the problem of proactive scalability in 5G mobile networks. The research aimed to evaluate various modeling strategies, including approaches based on incremental trees, adaptive ensembles, and probabilistic regression models. The evaluation considered not only predictive performance in terms of accuracy, but also the stability of the models under conceptual changes, their computational cost, and their practical impact on scalability decisions. This analysis enabled us to identify the most promising models for scenarios subject to change, taking into account the operational demands of mobile networks.

This dissertation also contributes to the understanding of the limitations of offline learning models when applied to non-stationary environments, which are typical of mobile networks. Experiments showed that, although these models perform competitively in contexts with low variability in traffic patterns, their accuracy deteriorates significantly when longer-term and significant changes occur. This limitation results in inappropriate scalability decisions, leading to functional overload and increased request loss. This contribution reinforces the importance of adopting adaptive approaches, such as online learning, to ensure greater robustness and operational continuity in constantly changing scenarios.

Another significant contribution of this work was the proposal of an architecture designed to enhance the proactive scalability of functions within the core of mobile networks. The architecture was designed to operate in a modular, flexible, and adaptive manner, allowing scalable systems to keep pace with changes in traffic behavior. This contribution establishes a structured foundation for the application of machine learning techniques in real-time scalability decisions, enabling the meeting of the demands of next-generation telecommunications networks.

A significant contribution of this dissertation to the academic community is the development of a simulator for the AMF function, designed to evaluate predictive scalability strategies in mobile networks. The simulator was designed with a focus on flexibility and fidelity to real-world network operations, enabling the testing of various instance allocation policies, traffic patterns, and load levels, with control over events such as function creation, shutdown, and saturation. The simulator enabled a realistic evaluation of the models in scenarios with different conceptual deviations. Furthermore, it was made available as open source, promoting the reproducibility of experiments and encouraging its adoption by other researchers and professionals in the field.

Finally, a significant contribution of this work was the critical discussion on the use of metrics in evaluating predictive models in telecommunications scenarios. The dissertation demonstrated that traditional metrics, such as the mean absolute error (MAE), are insufficient for capturing the effectiveness of models in highly dynamic environments. It was highlighted that,

in contexts requiring rapid responses to abrupt traffic variations, metrics such as inference time, training time, and resilience to request surges become crucial for model selection and application. This reflection broadens the understanding of more realistic evaluation criteria aligned with the operational requirements of next-generation mobile networks.

6.2 Future Work

As future work, we propose implementing a proactive scaling system in a functional instance of the 5GC mobile network core, using the open-source software OpenAirInterface[1]. This integration will enable the evaluation of online learning models in scenarios closer to real-world operation, with data observed at shorter intervals between traffic samples. This implementation enables a more sensitive analysis of network behavior, especially in situations of rapid demand variation. Furthermore, the proposed environment will enable the investigation of the impact of inference and training times on the quality of predictions, including scenarios where the correct answer used to update the model arrives with delays, as occurs in distributed data pipelines. Finally, using OpenAirInterface in conjunction with cloud-based solutions will allow simulating the practical applicability of the models in cloud-native architectures, testing their scalability and viability under real-world infrastructure constraints.

[1] <https://openairinterface.org/>

References

3rd Generation Partnership Project (3GPP). *3GPP TS 23.501 V17.11.0: System Architecture for the 5G System (5GS)*. [S.l.], 2024. Release 17. Accessed: 2025-06-25. Disponível em: <https://www.3gpp.org/DynaReport/23501.htm>. Citado na página 7.

3rd Generation Partnership Project (3GPP). *5G System; Principles and Guidelines for Services Operation*. [S.l.], 2024. Version 18.3.0. Disponível em: <<https://www.3gpp.org/ftp/Specs/archive/29_series/29.501/>>. Citado na página 7.

3rd Generation Partnership Project (3GPP). *5G System; Technical Realization of Service Based Architecture; Stage 3*. [S.l.], 2024. Version 18.3.0. Disponível em: <<https://www.3gpp.org/ftp/Specs/archive/29_series/29.500/>>. Citado na página 7.

3rd Generation Partnership Project (3GPP). *System Architecture for the 5G System (5GS); Stage 2*. [S.l.], 2024. Version 18.7.0. Disponível em: <<https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/>>. Citado na página 7.

6G-PPP Architecture Working Group. *6G Architecture: Vision and Enabling Technologies*. [S.l.], 2022. Available at: <https://5g-ppp.eu/wp-content/uploads/2022/12/6G-Arch-Whitepaper_v1.0-final.pdf>. Accessed: 2025-07-16. Citado na página 57.

AKSHATHA, N. M.; JHA, P.; KARANDIKAR, A. A centralized sdn architecture for the 5g cellular network. In: *2018 IEEE 5G World Forum (5GWF)*. [S.l.: s.n.], 2018. p. 147–152. Citado na página 6.

AKSHLLEY, K.; CARVALHO, M.; LOPES, R. Análise de desempenho de estratégias de autoscaling vertical e horizontal: um estudo de caso com o kubernetes. In: SBC. *Anais Estendidos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2022. p. 201–208. Citado 2 vezes nas páginas 1 and 60.

ALAWE, I. et al. On the scalability of 5G core network: The AMF case. In: *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. [S.l.: s.n.], 2018. p. 1–6. Citado 3 vezes nas páginas 52, 54, and 56.

ALAWE, I. et al. Smart scaling of the 5G core network: An RNN-Based approach. In: *2018 IEEE Global Communications Conference (GLOBECOM)*. [S.l.: s.n.], 2018. p. 1–6. Citado na página 57.

ALAWE, I. et al. An efficient and lightweight load forecasting for proactive scaling in 5G mobile networks. In: *IEEE Conference on Standards for Communications and Networking*. [S.l.: s.n.], 2018. Citado 5 vezes nas páginas 54, 55, 56, 57, and 63.

ALAWE, I. et al. Improving traffic forecasting for 5G core network scalability: A machine learning approach. *IEEE Network*, v. 32, n. 6, p. 42–49, 2018. Citado 6 vezes nas páginas 52, 54, 56, 64, 70, and 73.

ALHARTHI, S. et al. Auto-scaling techniques in cloud computing: Issues and research directions. *Sensors*, v. 24, n. 17, 2024. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/24/17/5551>. Citado na página 10.

BARAKABITZE, A. A. et al. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, v. 167, p. 106984, 2020. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128619304773>. Citado na página 6.

BARLACCHI, G. et al. A multi-source dataset of urban life in the city of milan and the province of trentino. *Scientific data*, Nature Publishing Group, v. 2, n. 1, p. 1–15, 2015. Citado na página 63.

BARTZ, E.; BARTZ-BEIELSTEIN, T. *Machine Learning: Foundations, Methodologies, and Applications*. Springer Nature Singapore Pte Ltd., 2024. (Online Machine Learning). Accessed: 2025-06-23. ISBN 978-981-99-7006-3. Disponível em: <https://doi.org/10.1007/978-981-99-7007-0>. Citado na página 11.

BERGSTRA, J. et al. Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [S.l.]: Curran Associates, Inc., 2011. p. 2546–2554. Citado na página 67.

BIFET, A.; GAVALDA, R. Adaptive learning from evolving data streams. In: SPRINGER. *International symposium on intelligent data analysis*. [S.l.], 2009. p. 249–260. Citado na página 28.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. ISBN 978-0-387-31073-2. Citado na página 21.

CHOUMAN, A.; MANIAS, D. M.; SHAMI, A. A reliable AMF scaling and load balancing framework for 5G core networks. In: *2023 International Wireless Communications and Mobile Computing (IWCMC)*. [S.l.: s.n.], 2023. p. 252–257. Citado 4 vezes nas páginas 52, 54, 56, and 64.

Cisco Systems. *Ultra Cloud Core 5G Access and Mobility Management Function, Release 2024.01 - Configuration and Administration Guide*. [S.l.], 2024. Accessed: 2025-06-25. Disponível em: <https://www.cisco.com/c/en/us/td/docs/wireless/ucc/amf/2024-01/config-and-admin/b_ucc-5g-amf-config-and-admin-guide_2024-01/m_amf-overview.html>. Citado na página 2.

Cisco Systems, Inc.; PLDT Inc. *Designing a Scalable Network to Unleash the True Potential of 5G*. [S.l.], 2020. Accessed: 2025-07-16. Disponível em: <https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/mobile-internet/pldt-designing-a-scalable-5g-network.pdf>. Citado 2 vezes nas páginas 57 and 58.

CRAMMER, K. et al. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, v. 7, n. Mar, p. 551–585, 2006. Citado na página 40.

DAHLMAN, E.; PARKVALL, S.; SKÖLD, J. *5G/5G-Advanced: The New Generation Wireless Access Technology*. 3rd. ed. London, United Kingdom: Academic Press, 2024. ISBN 978-0-443-13173-8. Citado na página 2.

DEALMEIDA, J. M. et al. Abnormal behavior detection based on traffic pattern categorization in mobile networks. *IEEE Transactions on Network and Service Management*, v. 18, n. 4, p. 4213–4224, 2021. Citado 2 vezes nas páginas 63 and 65.

DITZLER, G. et al. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, v. 10, n. 4, p. 12–25, 2015. Citado na página 11.

DO, T. V. et al. Properties of horizontal pod autoscaling algorithms and application for scaling cloud-native network functions. *IEEE Transactions on Network and Service Management*, IEEE, v. 22, n. 2, p. 1889–1898, 2025. Citado na página 59.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2000. p. 71–80. Citado na página 25.

Ericsson. Building robust critical networks with the 5g system. *Ericsson Technology Review*, 2023. Accessed: 2025-06-25. Disponível em: <https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/building-robust-critical-networks-with-the-5g-system>. Citado 3 vezes nas páginas 2, 57, and 59.

GANGWAL, G.; GRAY, K. *The 5G Core Network Demystified*. 2023. Accessed: 2025-06-23. Disponível em: <https://infohub.delltechnologies.com/pt-br/p/the-5g-core-network-demystified/>. Citado na página 7.

GAO, Z. 5g traffic prediction based on deep learning. *Computational Intelligence and Neuroscience*, v. 2022, p. 1–5, 2022. Citado na página 11.

GOMES, H. M. et al. Adaptive random forests for data stream regression. In: *ESANN*. [S.l.: s.n.], 2018. Citado na página 16.

GOMES, H. M. et al. On ensemble techniques for data stream regression. In: IEEE. *2020 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2020. p. 1–9. Citado na página 47.

GOMES, H. M.; READ, J.; BIFET, A. Streaming random patches for evolving data stream classification. In: IEEE. *2019 IEEE International Conference on Data Mining (ICDM)*. [S.l.], 2019. p. 240–249. Citado na página 47.

GOUK, H.; PFAHRINGER, B.; FRANK, E. Stochastic gradient trees. In: LEE, W. S.; SUZUKI, T. (Ed.). *Proceedings of the Asian Conference on Machine Learning (ACML)*. PMLR, 2019. (Proceedings of Machine Learning Research, v. 101), p. 1094–1109. Disponível em: <https://proceedings.mlr.press/v101/gouk19a.html>. Citado na página 43.

GUEMDANI, M.; PHUNG, C.-D.; SECCI, S. Experiences with disaggregated ran integrations. In: IEEE. *2024 3rd International Conference on 6G Networking (6GNet)*. [S.l.], 2024. p. 66–68. Citado na página 6.

HOEFFDING, W.; ROBBINS, H. The central limit theorem for dependent random variables. In: ____. *The Collected Works of Wassily Hoeffding*. New York, NY: Springer New York, 1994. p. 205–213. ISBN 978-1-4612-0865-5. Disponível em: <https://doi.org/10.1007/978-1-4612-0865-5_9>. Citado na página 26.

HOI, S. C. et al. Online learning: A comprehensive survey. *Neurocomputing*, v. 459, p. 249–289, 2021. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231221006706>. Citado 4 vezes nas páginas 1, 13, 58, and 61.

HOVAKIMYAN, G.; BRAVO, J. M. Evolving strategies in machine learning: A systematic review of concept drift detection. *Information*, v. 15, n. 12, 2024. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/15/12/786>. Citado na página 57.

IGLESIAS, G. et al. Data augmentation techniques in time series domain: a survey and taxonomy. *Neural Computing and Applications*, Springer, v. 35, n. 14, p. 10123–10145, 2023. Citado na página 76.

IKONOMOVSKA, E.; GAMA, J.; DŽEROSKI, S. Learning model trees from evolving data streams. *Data mining and knowledge discovery*, Springer, v. 23, p. 128–168, 2011. Citado na página 17.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. 1. ed. [S.l.]: Wiley, 1991. 685 p. ISBN 978-0471503361. Citado 2 vezes nas páginas 73 and 74.

KIVINEN, J.; WARMUTH, M. K. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, v. 132, n. 1, p. 1–63, 1997. ISSN 0890-5401. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0890540196926127>. Citado na página 23.

KRAWCZYK, B.; CANO, A. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. *Applied Soft Computing*, v. 68, p. 677–692, 2018. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494617307238>. Citado na página 12.

KURANAGE, M. P. et al. Ai-assisted proactive scaling solution for CNFs deployed in kubernetes. In: *2023 IEEE 12th International Conference on Cloud Networking*. [S.l.: s.n.], 2023. p. 265–273. Citado 5 vezes nas páginas 1, 53, 54, 56, and 57.

KURANAGE, M. P. J. et al. Deep learning based resource forecasting for 5G core network scaling in kubernetes environment. In: *8th IEEE International Conference on Network Softwarization*. [S.l.: s.n.], 2022. Citado 4 vezes nas páginas 53, 54, 56, and 64.

LI, R. et al. Estimating 5G network service resilience against short timescale traffic variation. *IEEE Transactions on Network and Service Management*, v. 20, n. 3, p. 2230–2243, 2023. Citado na página 1.

LIU, S. et al. Leaf: Navigating concept drift in cellular networks. *Proc. ACM Netw.*, Association for Computing Machinery, New York, NY, USA, v. 1, n. CoNEXT2, set. 2023. Disponível em: <https://doi.org/10.1145/3609422>. Citado na página 57.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, Springer, v. 12, n. 4, p. 559–592, 2014. Citado na página 60.

LORIDO-BOTRáN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, v. 12, 12 2014. Citado na página 9.

MANIAS, D. M.; CHOUMAN, A.; SHAMI, A. Model drift in dynamic networks. *IEEE Communications Magazine*, v. 61, n. 10, p. 78–84, 2023. Citado 3 vezes nas páginas 1, 11, and 61.

MASTELINI, S. M. et al. Online extra trees regressor. *IEEE Transactions on Neural Networks and Learning Systems*, v. 34, n. 10, p. 6755–6767, 2023. Citado na página 38.

MEKKI, M.; TOUMI, N.; KSENTINI, A. Microservices configurations and the impact on the performance in cloud native environments. In: *Proceedings of the 47th IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2022. p. 239–244. ISBN 978-1-6654-8001-7. Disponível em: <https://doi.org/10.1109/LCN53696.2022.9843385>. Citado na página 66.

MENG, J. et al. Modeling and generating control-plane traffic for cellular networks. In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2023. (IMC '23), p. 660–677. ISBN 9798400703829. Citado 2 vezes nas páginas 74 and 112.

MOURTADA, J.; GAïFFAS, S.; SCORNET, E. Amf: Aggregated mondrian forests for online learning. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, v. 83, n. 3, p. 505–533, 05 2021. ISSN 1369-7412. Disponível em: <https://doi.org/10.1111/rssb.12425>. Citado na página 13.

NADEAU, T. D.; GRAY, K. *SDN: Software Defined Networks: An authoritative review of network programmability technologies*. [S.l.]: " O'Reilly Media, Inc.", 2013. Citado na página 6.

OZA, N. C.; RUSSELL, S. J. Online bagging and boosting. In: RICHARDSON, T. S.; JAAKKOLA, T. S. (Ed.). *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*. PMLR, 2001. (Proceedings of Machine Learning Research, R3), p. 229–236. Reissued by PMLR on 31 March 2021. Disponível em: <https://proceedings.mlr.press/r3/oza01a.html>. Citado na página 19.

PASSAS, V. et al. Artificial intelligence for network function autoscaling in a cloud-native 5G network. *Computers and Electrical Engineering*, v. 103, p. 108327, 2022. ISSN 0045-7906. Citado 3 vezes nas páginas 53, 54, and 56.

PÉREZ-SÁNCHEZ, B.; FONTENLA-ROMERO, O.; GUIJARRO-BERDIÑAS, B. A review of adaptive online learning for artificial neural networks. *Artificial Intelligence Review*, Springer, v. 49, p. 281–299, 2018. Citado 2 vezes nas páginas 1 and 61.

READ, J.; ŽLIOBAITė, I. *Learning from Data Streams: An Overview and Update*. 2023. Disponível em: <https://arxiv.org/abs/2212.14720>. Citado na página 11.

ROY, D. M.; TEH, Y. W. The mondrian process. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2008. p. 1377–1384. Citado na página 13.

SILVA, T. da et al. An online ensemble method for auto-scaling NFV-based applications in the edge. *Cluster Computing*, Springer, p. 1–25, 2024. Citado 2 vezes nas páginas 1 and 61.

SILVEIRA, L. B. et al. Tutorial on communication between access networks and the 5g core. *Computer Networks*, v. 216, p. 109301, 2022. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128622003528>. Citado na página 66.

SINGH, P. et al. Research on auto-scaling of web applications in cloud: Survey, trends and future directions. *Scalable Computing: Practice and Experience*, v. 20, p. 399–432, 05 2019. Citado na página 10.

STARBUCK, C. *Machine Learning for Streaming Data: With Practical Applications Using Python*. Cham: Springer Nature Switzerland, 2023. ISBN 978-3-031-28674-2. Disponível em: <https://doi.org/10.1007/978-3-031-28675-9>. Citado na página 36.

The Kubernetes Authors. *Kubernetes Documentation: Autoscaling Workloads*. 2025. <https://kubernetes.io/docs/concepts/workloads/autoscaling/>. Accessed: 2025-08-17. Citado 2 vezes nas páginas 10 and 59.

YANG, Q.; GU, Y.; WU, D. Survey of incremental learning. In: *2019 Chinese Control And Decision Conference (CCDC)*. [S.l.: s.n.], 2019. p. 399–404. Citado na página 11.

ZHU, Q.; SUN, L. Big data driven anomaly detection for cellular networks. *IEEE Access*, v. 8, p. 31398–31408, 2020. Citado 2 vezes nas páginas 63 and 65.