RAFE: Resource Auto-Scaling For Multi-access Edge Computing With Machine Learning

Lucas Vinhal Pereira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2023

Lucas Vinhal Pereira

RAFE: Resource Auto-Scaling For Multi-access Edge Computing With Machine Learning

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Flávio de Oliveira Silva, Ph.D.

Uberlândia
2023

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

P436
2023

Pereira, Lucas Vinhal, 1998-
RAFE: Resource Auto-Scaling For Multi-access Edge Computing With Machine Learning [recurso eletrônico] / Lucas Vinhal Pereira. - 2023.

Orientador: Flávio de Oliveira Silva.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2024.11
Inclui bibliografia.

1. Computação. I. Silva, Flávio de Oliveira,1970-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Este trabalho é dedicado a Deus, minha esposa, minha família e amigos.

Agradecimentos

Primeiramente, expresso minha gratidão a Deus pela constante presença ao meu lado, tanto nos momentos desafiadores quanto nos momentos de alegria, abençoando-me e guiando cada passo nessa jornada.

Quero registrar, também, meu profundo agradecimento e reconhecimento à minha esposa Milenny, à minha mãe, Lourdes, e à minha irmã, Letícia, por serem uma base sólida em minha vida, proporcionando paciência, incentivo e apoio mesmo nos momentos mais difíceis.

Agradeço grandemente ao meu orientador, Prof. Dr. Flávio de Oliveira Silva, pela sua sabedoria, apoio, confiança e parceria ao longo de toda essa trajetória.

Aos respeitáveis professores do Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia (UFU), expresso minha gratidão pelos valiosos ensinamentos transmitidos e pelos conhecimentos compartilhados.

Agradeço à Universidade Federal de Uberlândia, seu corpo docente, direção e administração, por desempenharem um papel crucial ao orientar este novo caminho, fundamentado nos princípios da confiança, mérito e ética, valores que permeiam toda a instituição.

Por fim, estendo meu reconhecimento a todos que, de maneira direta ou indireta, contribuíram para a realização deste trabalho. O esforço coletivo tornou possível alcançar este marco, e sou profundamente grato por cada colaboração.

“Education is the most powerful weapon you can use to change the world.”
(Nelson Mandela)

Resumo

Para fornecer conectividade a vários dispositivos, incluindo IoT, o 5G depende de tecnologias como Virtualização de Funções de Rede (NFV) e Computação de Borda Multiacesso (MEC). Devido aos fluxos de rede continuamente variáveis, o gerenciamento de recursos desses dispositivos é uma das tarefas mais importantes que requerem algoritmos dinâmicos para dimensionar os recursos finitos de forma eficiente e satisfazer os requisitos de QoS. Por esta razão, a combinação de mecanismos reativos de escalonamento automático e modelos de estimativa de recursos baseados em IA são previstos como facilitadores promissores.

Este trabalho propõe o RAFE *(Resource Auto-scaling For Everything)*, um framework para dimensionar automaticamente aplicações VNF e MEC, reagindo e antecipando mudanças nos requisitos de recursos através de Aprendizado de Máquina, processos de treinamento distribuídos, múltiplos modelos de IA, e revalidação. Para tanto, primeiramente conduzimos uma análise aprofundada e comparação de vários algoritmos de ML aplicados em diversos contextos comumente enfrentados por aplicações de borda e nuvem. Empregando conjuntos de dados abertos, conduzimos uma avaliação abrangente de desempenho desses algoritmos em vários cenários frequentemente encontrados na borda da rede. Avaliamos sua eficácia em contextos univariados e multivariados, abrangendo previsões em uma e várias etapas, bem como tarefas envolvendo regressão e classificação. Além disso, apresentamos detalhadamente a arquitetura e os mecanismos do framework proposto e apresentamos um testbed de orquestração baseado em Docker para avaliar seu desempenho e funcionalidade em uma configuração adequada.

Além disso, foi validado e comparado o desempenho dos mecanismos de escalonamento automático implementados não apenas na carga de trabalho de rede esperada, mas também em uma carga de trabalho diferente e não vista, para avaliar o desempenho em relação a mudanças expressivas nos padrões aprendidos. Além disso, a integrabilidade e os efeitos de longa operação do RAFE através da eficácia da revalidação são validados. Os resultados experimentais mostram que o esquema proposto alcançou excelente desempenho na previsão e gestão de recursos, ao mesmo tempo que exigiu um curto período de tempo para treinar os modelos de previsão. Ademais, as soluções híbridas e preditivas superam a solução reativa no que diz respeito à latência para reação às mudanças de tráfego, mas principalmente, a abordagem híbrida é fundamental para alcançar a relação custo-benefício e, ao mesmo tempo, garantir bons resultados sobre padrões imprevistos. Por fim, RAFE apresentout um desempenho geral notável para escalonamento automático de aplicações de borda e em nuvem, apresentando também grande integrabilidade.

Palavras-chave: Escalonamento automático. Aprendizado de máquina. Redes Neurais Profundas. Gestão de recursos. Funções de rede virtual. Computação de borda multiacesso. Previsão de recursos..

RAFE: Resource Auto-Scaling For Multi-access Edge Computing With Machine Learning

Lucas Vinhal Pereira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação de Mestrado, 30/2023, PPGCO				
Data:	27 de dezembro de 2023	Hora de início:	13:02	Hora de encerramento:	16:04
Matrícula do Discente:	12112CCP018				
Nome do Discente:	Lucas Vinhal Pereira				
Título do Trabalho:	RAFE: Resource Auto-Scaling For Multi-access Edge Computing With Machine Learning				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rodrigo Sanches Miani - FACOM/UFU, Paulo Jorge Freitas de Oliveira Novais - Departamento de Informática da Escola de Engenharia da Universidade de Minho (Portugal) e Flávio de Oliveira Silva - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Paulo Jorge Freitas de Oliveira Novais e Flávio de Oliveira Silva - Braga, Portugal. O outro membro da banca e o aluno participaram da cidade de Uberlândia.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Flávio de Oliveira Silva, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

Ressalta-se que o examinador Paulo Jorge Freitas de Oliveira Novais, por ser estrangeiro, residente em outro país e não possuir CPF registrado no Brasil não assinará a ata de defesa.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 29/12/2023, às 17:37, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 08/01/2024, às 10:10, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5073504** e o código CRC **DD038A2D**.

Referência: Processo nº 23117.091170/2023-88 SEI nº 5073504

Abstract

To provide connectivity to multiple devices, including IoT, 5G relies on technologies like Network Function Virtualization (NFV) and Multi-access Edge Computing (MEC). Due to the continuously varying network flows, the resource management of these devices is one of the most important tasks that require dynamic algorithms to scale the finite resources efficiently and to satisfy QoS requirements. For this reason, the combination of reactive autoscaling mechanisms and AI-driven resource estimation models are foreseen as promising enablers.

This work proposes RAFE *(Resource Auto-scaling For Everything)*, a framework to auto-scale VNF and MEC applications, reacting and anticipating resource requirement changes through Machine Learning (ML), distributed training processes, multiple AI models, and revalidation. To this end, we first conduct an in-depth analysis and comparison of several ML algorithms applied in diverse contexts commonly faced by edge and cloud applications. Employing open datasets, we conducted a comprehensive performance evaluation of these algorithms in various scenarios frequently encountered at the network's edge. We assessed their effectiveness in univariate and multivariate contexts, encompassing one-step and multistep forecasting and tasks involving regression and classification. Furthermore, we detail the architecture and mechanisms of the proposed framework and present a Docker-based orchestration testbed to assess its performance and functionality in a suitable configuration.

Moreover, we validate and compare the performance of the implemented autoscaling mechanisms over the expected network workload and a different and unseen workload to assess the performance over expressive changes in the learned patterns. Additionally, we evaluated RAFE's integrability and long operation effects through the revalidation effectiveness. Experimental results show that the proposed scheme achieved outstanding performance in predicting and managing resources while requiring a short time to train the forecasting models. Additionally, the hybrid and the predictive solutions outperform the reactive solution in terms of latency to traffic change reaction. Still, principally, the hybrid approach is fundamental to achieving cost-effectiveness while ensuring good results over unforeseen patterns. Finally, RAFE shows outstanding overall performance for auto-scaling edge and cloud applications, presenting great integrability.

Keywords: Auto-Scaling. Machine Learning. Deep Neural Networks. Resource Management. Virtual Network Functions. Multi-Access Edge Computing. Resource Prediction..

List of Figures

List of Tables

Acronyms list

5G Fifth Generation

B5G Beyond 5G

6G Sixth-Generation

AOF Append Only File

AR Augmented Reality

ARIMA AutoRegressive Integrated Moving Average

AMF Access and Mobility Management Function

AWS Amazon Web Services

AI Artificial Intelligence

Bi-LSTM Bidirectional Long Short-Term Memory

BRNN Bidirectional Recurrent Neural Networks

CNN	Convolutional Neural Networks
CapEx	Reduced Capital Expenditure
CIoT	Cloud-centric Internet of Things
DNN	Deep Neural Networks
DT	Decision Tree
DL	Deep Learning
ETSI	European Telecommunications Standards Institute
EMS	Element Management System
ETS	Exponential Smoothing
Enc-Dec-LSTM	Encoder-Decoder LSTM
Enc-Dec-CNN-LSTM	Encoder-Decoder CNN-LSTM
FNN	Feedforward Neural Network
GRU	Gated Recurrent Unit
GKE	Google Kubernetes Engine
GCP	Google Cloud Platform
HPA	Horizontal Pod Autoscaler

IoT Internet of Things

IDS Intrusion Detection System

ISG ETSI Industry Specification Group of the European Telecommunications Standards Institute

KNN K-Nearest Neighbors

LSTM Long Short-Term Memory

MEC Multi-Access Edge Computing

ME Mobile Edge

MANO Management and Orchestration

MLP Multilayer Perceptron

ML Machine Learning

MSE Mean Squared Error

MAE Mean Absolute Error

NFV Network Functions Virtualization

NN Neural Networks

OpEx Operating Expenditure

OSM	Open Source MANO
PHPA	Predictive Horizontal Pod Autoscaler
P-CSCF	Proxy-Call/Session Control Functions
QoS	Quality of Service
RFECV	Recursive Feature Elimination with Cross-Validation
RNN	Recurrent Neural Networks
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SLA	Service Level Agreement
SDN	Software Defined Networking
SVM	Support Vector Machines
SVR	Support Vector Regression
SMDP	Semi-Markov Decision Process
SFC	Service Function Chaining
SaaS	Software as a Service
SGD	Stochastic Gradient Descent

TSP	Telecom Service Provider
UE	User Equipment
VM	Virtual Machine
VNF	Virtual Network Function
VNFVI	Virtual Network Function Virtual Infrastructure
NFVO	NFV Orchestrator
VNFM	VNF Manager
VMAF	Virtual MEC Application Function
VIM	Virtualized Infrastructure Manager
VR	Virtual Reality
vMEC	Virtual Mobile Edge Computing
VU	Virtual User

Contents

Chapter 1 Introduction

The advent of the 5th generation of mobile technology Fifth Generation (5G) and its successors Beyond 5G (B5G) and Sixth-Generation (6G), in addition to the crescent number of connected Internet of Things (IoT) devices has ushered in a new era characterized by the ever-growing demands of innovative applications across various vertical industries, including manufacturing, automotive, healthcare, and agriculture. These applications are mainly distinguished by ultra-low latency, exceptionally high throughput, and increased connectivity density (LI; XU; ZHAO, 2018). The cloud-centric model is commonly employed to enable these applications. However, it encounters several challenges, including bandwidth, latency, uninterrupted service, resource constraints, and security concerns (KHAN et al., 2019).

To solve these challenges, over the past decade, numerous endeavors have been made to extend the traditional centralized cloud computing model towards a more geographically distributed approach (SITTóN-CANDANEDO et al., 2019). This approach strategically distributes computational, networking, and storage resources closer to data sources or end-user applications. For instance, the geo-distributed cloud computing model (WU et al., 2015) seeks to partition processing tasks and allocate them to data centers near the network's edge. Additionally, the concept of mobile cloud computing has introduced a paradigm where cloud computing resources are provisioned based on physical proximity, leveraging local wireless internet access points (QI; GANI, 2012).

In the context of Edge Computing, particularly Multi-Access Edge Computing (MEC), has emerged as a network architecture concept defined by European Telecommunications Standards Institute (ETSI). MEC sits between the cloud and intelligent end-devices, providing cloud computing capabilities, encompassing computing, storage, data management, and network resources brought to the edges of the cellular network. This strategic placement of MEC facilitates hosting applications that must be near the end-user, significantly reducing user experience delays and relieving the load on the transport network (PRIYA et al., 2019). MEC's ambitions include enhancing interoperability among service providers, enabling real-time processing and analytics, supporting mobility, accommodating geographic distribution, and adapting to various devices and node factors.

All these approaches aim to satisfy and enable the crescent demands for hardware resources. However, these resources are finite. The resources available at Mobile Edge (ME) nodes, at the edge network, and even at the cloud are limited in physical and virtual capacity and represent costs for the operators. Therefore, efficient resource management becomes imperative (MUSADDIQ et al., 2018; JENNINGS; STADLER, 2015).

To address this requirement, solutions such as NFV offer flexibility and programmability by virtualizing MEC applications into what are known as Virtual MEC Application Functions (VMAFs) (SCIANCALEPORE et al., 2016). These VMAFs can be easily deployed and scaled in response to the demands of end-users consuming the services. Additionally, one of the fundamental features of NFV is elasticity, which allows VMAFs to acquire and release resources following fluctuating demands dynamically. Therefore, resource allocation is not a straightforward task. While current virtualization platforms offer autoscaling capabilities triggered manually, often in a reactive manner (e.g., scaling up a VMAF when CPU utilization reaches 80%), there's a growing need for a predictive autoscaling mechanism (which can be integrated with the management or control centers, as the MANO in an NFV architecture). Such a mechanism would proactively adjust resources to match the workload handled by the VMAF without human intervention. In this context, Deep Learning (DL) algorithms, including time series forecasting, come into play to develop predictive autoscaling solutions that enhance resource utilization efficiency.

1.1 Motivation

The new era of communication services demands high throughput, low latency, and high availability (JIANG et al., 2021). Even a modest increase in the number of connected or cyber-physical objects brings about significant changes in the computing landscape (JIANG et al., 2021). This shift can potentially unleash a surge of computation and hyper-connectivity that existing infrastructure struggles to handle while maintaining historical levels of service quality (LI et al., 2016). Therefore, precise resource allocation is fundamental for addressing these challenges in cloud-based applications and across the network, in Virtual Network Function (VNF), MEC applications, and IoT devices.

However, resource management in NFV and MEC environments constitutes a complex and critical element (MUSADDIQ et al., 2018). This entails efficiently utilizing computing, storage, and networking resources to ensure optimal performance and user experience. One of the primary challenges in VNF and MEC resource management is coping with dynamic workloads (HAIBEH; YAGOUB; JARRAY, 2022a). The demand for services and applications can fluctuate rapidly due to user behavior, time of day, and special events. As a result, resource allocation must be capable of adapting in real-time to meet these varying demands. Failure to allocate resources dynamically can lead to performance bottlenecks, service degradation, and dissatisfied users. Furthermore, as the number of VNFs and MEC applications grows, the resource management system must scale accordingly to accommodate the increased demand (LLORENS-CARRODEGUAS et al., 2021). Moreover, the number of connected devices, the heterogeneity of the network, the variability of traffic patterns, and the need to maintain low latency and high reliability require the development of sophisticated autoscaling approaches that can enable the VNFs and MEC applications to scale dynamically to meet the demand (HERRERA; BOTERO, 2016).

Designing effective resource allocation policies is vital to optimizing resource utilization in cloud and edge environments. Resource allocation policies determine how resources are distributed among different instances (of VNFs and MEC applications) based on priority, service-level agreements, and Quality of Service (QoS) requirements. Creating policies that prioritize critical network functions and latency-sensitive applications while ensuring fair resource allocation for less demanding workloads is a challenge that requires careful consideration and continuous refinement (QU; CALHEIROS; BUYYA, 2016). Additionally, energy efficiency is an increasingly critical aspect of resource management in this area (ETEMADI; GHOBAEI-ARANI; SHAHIDINEJAD, 2021). Optimizing resource usage to minimize energy consumption becomes essential with the growing focus on sustainability and reduced carbon footprints. Resource managers must incorporate dynamic power management, workload consolidation, and other energy-efficient strategies into their resource management practices. This is especially more complex in VNF and MEC environments, which, due to the heterogeneous hardware landscape of edge environments, can introduce further complexity into resource management (SAHNI; VIDYARTHI, 2017). Different edge servers and devices have varying capabilities, making it challenging to optimize resource allocation to meet the unique requirements of different VNFs and MEC applications. Efficient resource management should consider the capabilities and limitations of diverse hardware to maximize performance and cost-effectiveness.

As the final point, resource management strategies should also include failure detection and recovery mechanisms. Failures are inevitable in a distributed and dynamic environment like in NFV and MEC. To address that, the resource management system should be able to detect failures promptly and react to maintain service continuity and minimize downtime (VINAY; KUMAR, 2016). Moreover, a delicate balance must be struck between cost and QoS. To ensure QoS, more VMAF instances or CPU resources may need to be allocated, but this increases costs. Therefore, the autoscaling mechanism must consider the economic implications of its decisions to minimize total expenditure while maintaining an acceptable QoS level as stipulated in Service Level Agreements (SLAs) between end-users and application providers. Balancing performance requirements with cost considerations demands sophisticated resource optimization algorithms and continuous monitoring of resource usage and demand patterns (SINGH et al., 2019).

1.2 Objectives and Research Challenges

Therefore, this work's main goal is to propose and evaluate a framework for optimum auto-scaling of applications placed both at the edge and at the core of the network, being compatible with ETSI Management and Orchestration reference software stack and enabling hybrid scaling through reactive mechanisms AI-based predictive mechanisms.

To achieve such higher goals, some specific goals should be considered, namely:

- ❏ Investigate and review the state of the art on Machine Learning for Time-Series Forecasting, Network Functions Virtualization, and Multi-Access Edge Computing;
- ❏ Investigate the concept of optimal auto-scaling of edge and cloud applications in terms of performance and cost-effectiveness, reviewing its general objectives, as well as obstacles present in their conception and the limitations to achieve optimum results over complex network scenarios;
- ❏ Select the most suitable algorithms and mechanisms to enable the auto-scaling of different kinds of applications placed anywhere in the network;
- ❏ Design and implement a framework, using open-source technologies, capable of achieving near-optimum performance in dynamic auto-scaling of edge and cloud applications across various network traffic scenarios. The framework must demonstrate adaptability to emerging traffic patterns, ensuring seamless scalability and efficiency in resource utilization.
- ❏ Using experimental approaches, evaluate the framework to assess its features and capabilities.

1.3 Hypothesis

Once the previously presented objectives are established, the hypothesis is supported, and it is possible to propose and evaluate an architecture to address the auto-scaling challenges for both edge and cloud-placed applications. To this end, a hybrid auto-scaling framework can be proposed for optimum management of different applications, including VNF, MEC, and cloud applications. To this end, it should be compatible with ETSI's Management and Orchestration software stack to offer integrability, have a core concept for effectively running over hardware resource constraints, have failure detection and recovery, and adapt itself to unexpected changes in network traffic patterns.

1.4 Contributions

The contributions of this work can be divided into two main parts. The first part focuses on selecting the most suitable forecasting algorithms for the framework proposed in the second part. To achieve this, a comprehensive investigation of various time series forecasting algorithms based on Machine Learning (ML) is conducted. Multiple datasets obtained from VNF monitoring, encompassing diverse trends, seasonality, sizes, and features, are employed to assess the key factors that influence the prediction performance. Moreover, several experiments are carried out to address the dynamic nature of edge environments, involving the implementation and training of each algorithm using different forecasting approaches and variations. These experiments aim to determine the performance of the algorithms in common scenarios encountered in cloud and edge computing. Finally, the overall results are compared to identify the algorithms integrated into the auto-scaling framework proposed in the second part.

In the second part, this work proposes and evaluates a novel approach to address the auto-scaling challenges of both edge and cloud applications. This solution combines reactive and predictive methods to ensure optimal scaling decisions. Moreover, it introduces a core concept that effectively handles hardware resource constraints through the distributed execution of ML training and resource-consuming tasks. It introduces a diverse set of tools designed to address the dynamic nature of edge and cloud environments. A multi-univariate-models approach is applied to achieve optimum results and keep a low time for training the model. A re-validation mechanism is also introduced to effectively manage the workload pattern changes inherent in network environments. This comprehensive suite of tools aims to optimize the performance and adaptability of auto-scaling in these complex and ever-changing contexts.

Therefore, among the main contributions of this work are:

1. an empirical evaluation of multiple ML algorithms and Deep Neural Networkss (DNNs) for predicting the consumption of computational resources in VNF and MEC environments, comparing algorithms and training paradigms using time-series data from multiple datasets with diverse data patterns (we implemented and assessed these approaches across numerous variations commonly encountered when designing forecasting techniques for scaling edge and cloud applications);

2. a hybrid auto-scaling framework that enables a reactive mechanism based on threshold rules and a predictive mechanism based on time series forecasting using deep-learning models (using multi-univariate models instead of a single multivariate model);

3. an infrastructure that satisfies the requests for optimum auto-scaling of different kinds of applications, including VNFs, MEC applications, and cloud-placed appli-

cations;

4. an architecture that enables the use of multiple AI models, instead of the commonly used multivariate models, with a distributed training mechanism and invalidation mechanisms that manage the model's life cycles and enable the adaption to the changes in the network patterns;

5. a testbed including a Docker-based orchestration structure to host VNF/MEC applications and simulate different network traffic patterns;

6. a performance assessment of the proposed framework considering scalability and performance, validating the autoscaling mechanisms over a seen and learned traffic pattern and a different and unseen workload. (this last one validates how the proposed framework and different autoscaling mechanisms perform over expressive changes in the learned traffic workloads);

7. an experimental study assessing the performance of the different autoscaling mechanisms, comparing the reactive, the predictive, and the hybrid approaches.

1.5 Outline

This dissertation is structured into five chapters. In Chapter 2, we delve into the theoretical and practical aspects that clarify the various subjects encompassed by the research objectives and their validity. We extensively explore computing system technologies, including Machine Virtualization, Network Functions Virtualization, the Internet of Things, Edge Computing, Machine Learning Forecasting, and the core concepts of Auto-Scaling Mechanisms.

Chapter 3 outlines the dissertation proposal, offering insights into the implementation specifics and the framework organization, presenting the various components, their respective functions, and the methodologies applied.

Moving on to Chapter 4, we present multiple comprehensive experimental assessments that underscore the achievement of each objective and furnish an in-depth analysis of how this work stands compared to prior proposals.

Chapter 5 discusses the conclusions drawn from developing and experimenting with the proposed solution. Moreover, this chapter identifies potential projects and initiatives, both of a productive and test-oriented nature, that can be realized by adopting the proposed architecture.

Chapter 2

Background and Related Work

Information technology has witnessed a steady evolution, marked by the extensive virtualization of its infrastructure for several years now (CHOWDHURY; BOUTABA, 2010). The network is the next step as operators strive to stay abreast of these rapid changes. Currently, not only cloud-placed applications but also IoT devices, VNFs, and MEC applications are huge consumers of bandwidth, demanding unparalleled levels of flexibility and speed from networks (CRUZ; ACHIR; VIANA, 2022; ABBAS et al., 2018; HAIBEH; YAGOUB; JARRAY, 2022b). However, expanding networks to accommodate peak traffic loads surpasses the capabilities of most operators due to the exorbitant maintenance costs and extensibility (CHIANG et al., 2023).

Incrementing this challenge, the network's edge, with its proximity to end-users and IoT devices, requests unparalleled speed and responsiveness. It's the realm of real-time data processing, low-latency, and mission-critical applications. In the network's midway, the NFV virtualizes network functions to enable the edge's high bandwidth, speed, and flexibility demand. Additionally, the cloud serves as the centralized powerhouse, providing vast computational capabilities and storage. These solutions, as a whole, share a common demand for balancing resource allocation and optimizing performance (HAIBEH; YAGOUB; JARRAY, 2022b). Auto-scaling, the ability to dynamically allocate resources based on real-time demand, plays a pivotal role in this context, offering cost-efficient performance for the applications to fulfill this demand. In this exploration, we delve into the auto-scaling-related fundamentals for applications placed anywhere in the networks, from the edge to the cloud, uncovering its potential to ensure optimal resource allocation, minimize costs, and deliver seamless, high-performance computing experiences.

The remainder of this chapter is structured as follows. To introduce the targeted applications of auto-scaling within this work, firstly, in Section 2.1, we delve into the benefits of Network Functions Virtualization (NFV) and the impacts that drive its adoption. This section provides a foundational understanding of NFV, including its architectural framework and components. In Section 2.2, we provide an overview of Edge Computing, Multi-Access Edge Computing (MEC) applications, and the Internet of Things (IoT) paradigm, delving into their core concepts, principles, and advantages. This section particularly focuses on the network's edge and fog, key technologies for the current and future networks, and emerging solutions. Section 2.3 introduces the concept of time-series forecasting as a key element in predictive auto-scaling solutions. We elucidate the main mechanisms for predictive scaling, which center around machine-learning algorithms, presenting various categories and key algorithm architectures. In Section 2.4, we delve into the concepts and enablers related to auto-scaling, elucidating the scalable resources and monitoring essentials for effective resource allocation. Finally, Section 2.5 provides a comprehensive summary of the current state of the art in related research and works.

2.1 Network Function Virtualization

Network Functions Virtualization (NFV) is a paradigm shift driven by the need for greater agility, efficiency, and scalability in the telecommunications and networking infrastructure (SOUSA et al., 2019). At its core, NFV is based on server virtualization. It involves the abstraction and virtualization of network functions, traditionally realized through dedicated hardware appliances, into software-based instances that can run on standardized compute resources. NFV also facilitates the establishment of an ecosystem with the capability to oversee, allocate resources, monitor, and deploy virtualized network entities (RAY; KUMAR, 2021).

One of the fundamental technical characteristics of Virtual Network Functions (VNFs) is the concept of decoupling (HAWILO et al., 2014). NFV decouples network functions (such as load balancers, firewalls, or Intrusion Detection Systems (IDSs), traditionally were tightly bound to dedicated and specific hardware appliances) from the hardware, enabling them to operate as independent software components or VNFs on general-purpose servers, storage, and networking equipment. Virtualization plays a main role in NFV once this decoupling is usually achieved through solutions like containerization and hypervisors (HAN et al., 2015), which enables the creation of Virtual Machines (VMs) or containers to host the VNFs. This level of flexibility enhances service agility, allowing network operators to rapidly deploy new services, adjust capacity, and optimize resource allocation (BONFIM; DIAS; FERNANDES, 2019).

As networks continue to evolve to meet the demands of emerging technologies like 5G and edge computing, NFV remains a foundational concept in modern telecommunications infrastructure, decoupling and virtualizing network functions from proprietary hardware to deliver increased scalability, agility, and interoperability to network services (SOUSA et al., 2019). Moreover, NFV also introduces performance optimization and resource allocation challenges once efficiently distributing computational resources, like CPU, memory, and network bandwidth, among various VNFs running on the same physical hardware is essential to optimize utilization and avoid bottlenecks (SCHARDONG;

NUNES; SCHAEFFER-FILHO, 2021).

2.1.1 Benefits of NFV

NFV offers many benefits that revolutionize how telecommunications and networking services are designed, deployed, and managed. Some of the key advantages of NFV include:

1. **Cost Efficiency**: NFV eliminates the need for dedicated, specialized hardware appliances for various network functions. This consolidation leads to significant cost savings on procurement, maintenance, and power consumption.

2. **Flexibility and Scalability**: NFV enables the dynamic scaling of network functions based on real-time demand. Service providers can allocate or release resources as needed, ensuring optimal resource utilization. Moreover, it enables rapid service deployment, accelerating the deployment of new network services and functions and reducing time-to-market for innovative offerings.

3. **Resource Optimization**: NFV allows efficient resource utilization by sharing physical infrastructure among multiple virtualized functions, reducing resource underutilization. Additionally, it enables resource isolation through virtualization technologies, ensuring that one function's performance or security issues do not impact others.

4. **Improved Network Management**: it introduces centralized orchestration and management, simplifying the configuration and monitoring of network services. Furthermore, it facilitates automation operations, reducing manual intervention in network management tasks and minimizing human errors.

5. **Enhanced Service Agility**: NFV enables on-demand services, with rapid provisioning and modification of services to meet changing customer requirements and market demands. Moreover, it includes service-chaining capabilities that allow the creation of complex service chains by orchestrating multiple VNFs in a specific order.

6. **Vendor Neutrality**: NFV promotes vendor neutrality by standardizing interfaces and protocols. Service providers can choose best-of-breed VNFs and infrastructure components from different vendors, offering easier interoperability between them.

7. **Reduced Capital Expenditure (CapEx) and Operating Expenditure (OpEx)**: With NFV, there is less dependency on expensive proprietary hardware, leading to reduced CapEx due to lower hardware costs. Also, NFV simplifies network management and reduces the need for extensive physical deployments, resulting in lower OpEx.

8. **Network Resilience and Fault Tolerance**: NFV supports self-healing mechanisms where failed VNF instances can be automatically replaced, ensuring network continuity. Also, it enables dynamic load balancing to distribute traffic evenly and improve network resilience.

9. **Edge Computing and 5G Enablement**: NFV is crucial for delivering services at the network edge, essential for latency-sensitive applications and 5G network capabilities.

10. **Green Computing**: By consolidating hardware and optimizing resource usage, NFV can reduce energy consumption and contribute to more environmentally friendly networks.

2.1.2 NFV Architecture

NFV enables third-party software to operate on standardized and shared hardware, opening up multiple possibilities for network management (ANVITH et al., 2019). Within this context, the virtualized instantiation of network functions is called a Virtual Network Function (VNF). Each VNF is specifically engineered to fulfill a distinct network function, whether it be routing, switching, firewalling, network load balancing, and so forth (KUNDIMANA; VYUKUSENGE; TSYM, 2021).

Various providers can present a range of VNFs, allowing service providers to select a chain combination of functions that align with their requirements (ANVITH et al., 2019). This diversity in choices underscores the necessity for standardization in their virtual environment management, which is a key driver for its technical success. Industry bodies like the ETSI have defined specifications and interfaces to ensure interoperability between VNFs and NFV infrastructure components. These standards ensure that a deployed VNFs is not restricted to particular hardware configurations and is versatile enough to function in any compatible environment and multi-vendor ecosystem. This allows network operators to select the best functions and infrastructure components while ensuring seamless integration (HOFFMANN et al., 2018). Therefore, a critical requirement is a reference architecture that offers consistency and uniformity in deploying and developing VNFs, regardless of the specific methodologies involved.

The Industry Specification Group of the European Telecommunications Standards Institute (ISG ETSI) has devised over 100 distinct specifications and reports dedicated to the virtualization of network functions (DAHMEN-LHUISSIER, 2021), with a primary emphasis on the management and orchestration of virtualized resources. From an architectural perspective, these NFV specifications outline and delineate the prerequisites for virtualization, the infrastructure components, their interfaces, and the protocols and APIs for managing these interfaces. Another set of specifications ISG NFV presents pertains to the structure and format of artifact deployment and packaging models utilized within the NFV management and orchestration framework (GIUST; COSTA-PEREZ; REZNIK, 2017).

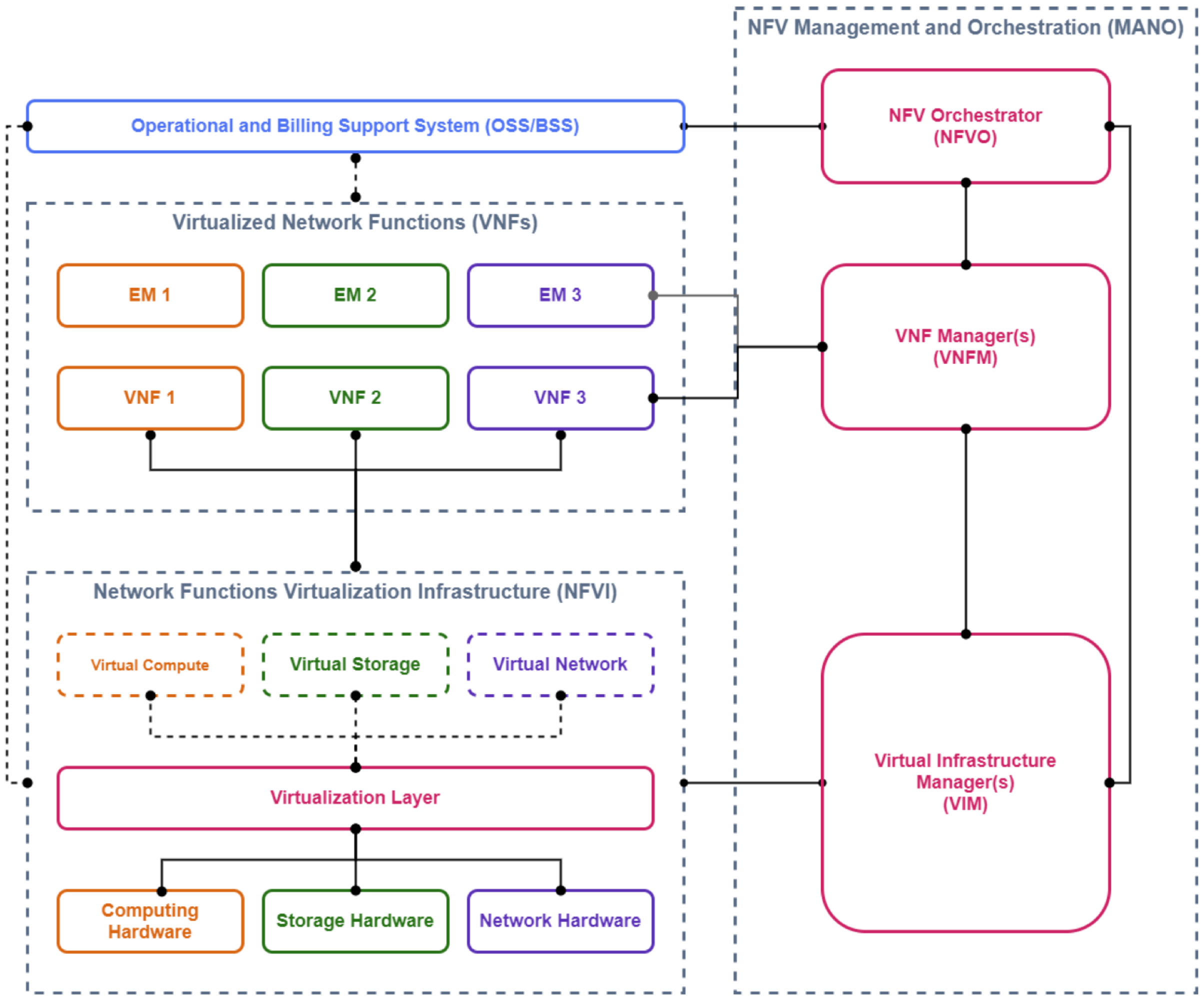

NFV Management and Orchestration (MANO)
Operational and Billing Support System (OSS/BSS)
NFV Orchestrator (NFVO)
Virtualized Network Functions (VNFs)
EM 1
EM 2
EM 3
VNF 1
VNF 2
VNF 3
VNF Manager(s) (VNFM)
Network Functions Virtualization Infrastructure (NFVI)
Virtual Compute
Virtual Storage
Virtual Network
Virtualization Layer
Virtual Infrastructure Manager(s) (VIM)
Computing Hardware
Storage Hardware
Network Hardware

Figure 1 – ETSI VNF Architecture (CUNHA, 2021).

Upon these specifications, a comprehensive architectural framework has been established by ETSI, defining distinct focal areas, as depicted in Figure 1. This architectural framework is the foundational structure for standardization and development efforts and is commonly referred to as the ETSI NFV framework (PANAGIOTIS et al., 2020). ETSI ISG has categorized these functions into three main overarching blocks: the infrastructure block, the virtualized function block, and the management block. In the formal nomenclature defined by ETSI, these blocks are described as follows:

- **Virtual Network Functions**: are the software-based representations of traditional network functions, such as firewalls, routers, load balancers, and more. They are the functional building blocks that provide network services when instantiated within the VNFVI. VNFs can be deployed, scaled, and decommissioned dynamically (XIE et al., 2017).

- ❏ **Virtual Network Function Virtual Infrastructure (VNFVI)**: The VNFVI block is the foundation for the virtualized network functions. It comprises the physical and virtual resources necessary to support the VNFs (MECHTRI et al., 2017). This includes computing, storage, and networking resources, both hardware and virtualized, that are abstracted and managed to create a flexible infrastructure.

- ❏ **Management and Orchestration (MANO)**: The MANO block functions as the control layer with the primary role of orchestrating and overseeing the management and allocating role of both VNFs and the VNFVI (KOMAREK et al., 2017).

The MANO layer, responsible for overseeing and managing all entities within the architecture, possesses a comprehensive awareness of the usage, operational status, and utilization statistics of each architectural block (YU; YANG; FUNG, 2020). As a result, MANO serves as the most suitable interface for operating systems and the central hub for collecting network data. It operates as an independent section within the architecture but collaborates closely with the VNFVI and VNF blocks. Within this layer, comprehensive resource management occurs within the VNFVI layer, encompassing resource creation, deletion, and allocation to VNFs. Furthermore, it guarantees the efficient deployment, maintenance, and optimization of network services to align with SLAs (ORDONEZ-LUCENA et al., 2017).

The MANO management component can be defined as a combination of three functional components: (1) NFV Orchestrator (NFVO): responsible for new network services and VNFs. The NFVO coordinates the overall management and orchestration of VNFs. It receives service requests, interprets them, and translates them into actions to instantiate VNFs and manage resources (ORDONEZ-LUCENA et al., 2017). (2) VNF Manager (VNFM): the VNFM focuses on the lifecycle of individual VNFs. It interacts with the NFVO to instantiate, configure, monitor, and terminate VNF instances. It is also responsible for coordinating and adapting to configure and report the events between NFV infrastructure, and element management systems/network (PAGANELLI et al., 2017). (3) Virtualized Infrastructure Manager (VIM): is responsible for managing the VNFVI resources, controlling and managing the computational, storage, and network resources in NFV infrastructure. It communicates with the NFVO and VNFM to allocate and optimize these physical and virtual resources (PANAGIOTIS et al., 2020).

VNFs depend on the availability of virtual hardware, simulated by software resources running on physical hardware. The ETSI NFV framework makes this possible through the infrastructure component (NFVI), which encompasses physical hardware resources, the virtualization layer, and virtual resources (LAL et al., 2017). This flexible, functional component can be extended and scaled from a single physical host to multiple interconnected local devices. The virtualization layer is integral to the VNFVI functional block. It directly interacts with the host machine's hardware components, providing a simplified and abstracted view of computing, storage, and networking resources for VNFs. In essence, the virtualization layer decouples VNFs from the underlying hardware of the host machine, granting them access to a computational resource management interface (PAGANELLI et al., 2017).

The VNFVI block is composed of three components: (1) Virtualization Layer: This component includes hypervisors and virtualization technologies responsible for creating virtual machines (VMs) or containers to host VNF instances. It abstracts the underlying hardware, enabling resource sharing and isolation among VNFs (ALWAKEEL; ALNAIM; FERNANDEZ, 2019); (2) Physical Infrastructure: The physical infrastructure represents the actual hardware, such as servers, switches, and storage devices, on which the virtualized resources are instantiated (PANAGIOTIS et al., 2020). (3) Resource Pool: This encompasses all available resources within the VNFVI, including CPU, memory, storage capacity, and network bandwidth. Resource management ensures efficient allocation to VNFs based on their requirements (KAUR; MANGAT; KUMAR, 2022)

Finally, the VNF layer is the deployment ground for virtualized functions and comprises the VNFs block along with its associated management functional block known as VNF-Manager. It typically falls under the purview of an Element Management System (EMS). The EMS is pivotal in various functions, including adjustments, monitoring, fault remediation, configuration, accounting, performance assessment, and security. Its scope aligns with the traditional EMS, acting as a communication bridge, connecting the northern and southern parts of the network management landscape and interfacing with network management systems and VNFs, respectively. In the Telecom Service Provider (TSP) environment, an EMS holds the capacity to furnish critical information through operational support systems.

When transitioning from physical to virtual devices, network operators often prefer not to overhaul existing management tools and applications, including Operations Support System and Business Support System (OSS/BSS) (XIE et al., 2017). The ETSI framework does not force a change in these tools as part of the NFV transformation. Instead, it encourages improving and developing tools and systems capable of utilizing the functional blocks within the management architecture and reaping benefits like elasticity and agility. The ETSI framework offers a solution in the form of the NFV Orchestrator, which extends the current OSS/BSS functionalities and manages both the operational and deployment aspects of VNFVI and VNF (GONZALEZ et al., 2018).

2.1.3 Message Flow in NFV Architecture

To provide a more comprehensive insight, Figure 2 illustrates the intricate interplay among the core components of the ETSI architecture in the context of basic service implementation. This process unfolds through the following steps:

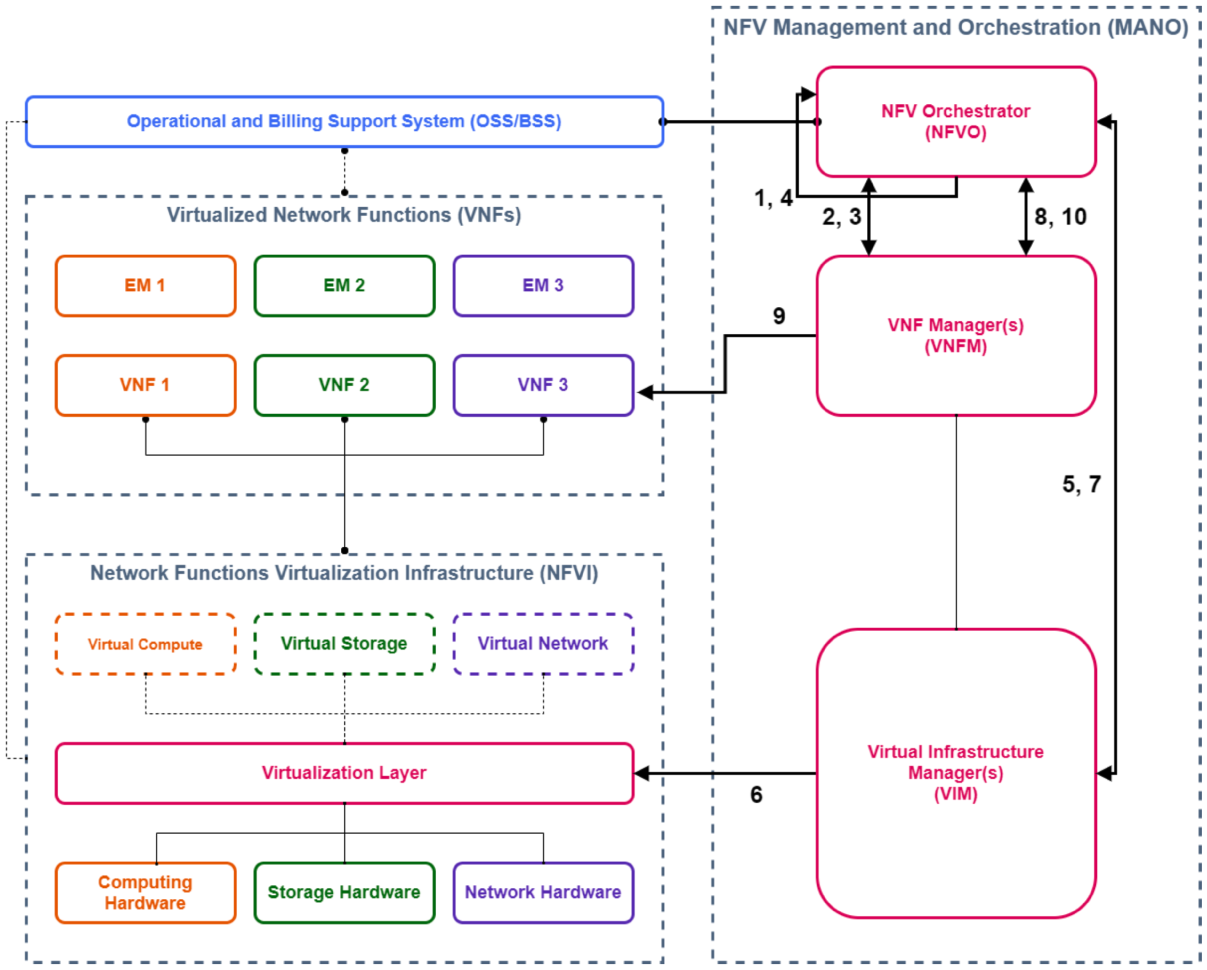

NFV Management and Orchestration (MANO)
Operational and Billing Support System (OSS/BSS)
NFV Orchestrator (NFVO)
Virtualized Network Functions (VNFs)
EM 1
EM 2
EM 3
VNF 1
VNF 2
VNF 3
1, 4
2, 3
8, 10
9
VNF Manager(s) (VNFM)
5, 7
Network Functions Virtualization Infrastructure (NFVI)
Virtual Compute
Virtual Storage
Virtual Network
Virtualization Layer
6
Virtual Infrastructure Manager(s) (VIM)
Computing Hardware
Storage Hardware
Network Hardware

Figure 2 – Message Flow in the ETSI NFV architecture (CUNHA, 2021).

1. NFVO gains a holistic understanding of the end-to-end topology.

2. NFVO initiates the instantiation of the requisite VxFs and communicates this action to VNFM.

3. VNFM conducts a detailed assessment, determining the necessary number of VMs and their specific resource requirements. This data is then relayed back to NFVO to facilitate VNF creation.

4. Armed with knowledge about hardware resources, NFVO performs a validation check to ensure the availability of adequate resources for VM creation. Subsequently, NFVO triggers a request for VM creation.

5. NFVO dispatches a request to VIM, instructing it to create the VMs and allocate the required resources to these virtual machines.

6. In response to NFVO's directive, VIM engages the virtualization layer to orchestrate the creation of the designated VMs.

7. Upon the successful completion of VM creation, VIM acknowledges this achievement and communicates the status back to NFVO.

8. NFVO promptly informs VNFM that the necessary VMs are ready, poised to facilitate the deployment of VxFs.

9. The VxFs block proceeds to configure the VxFs by specific parameters.

10. Following the successful configuration of the VxFs, VNFM notifies NFVO that the VxFs are fully prepared, configured, and primed for immediate utilization.

2.2 Edge Computing and IoT

The concept of fog computing was introduced by industry and academia to address the challenges mentioned in the last section and fulfill the demand for a computing paradigm that operates near connected devices (PAN et al., 2018). Fog computing serves as a bridge between the cloud and edge devices, facilitating computing, storage, networking, and data management at network nodes situated at the network's edge. It's important to emphasize that edge computing is not situated directly on IoT devices but is positioned as close as a single network hop away from them (ADEGBIJA; LYSECKY; KUMAR, 2019). Therefore, within a local IoT network, the edge can sometimes extend beyond a single network hop away from IoT devices.

2.2.1 Edge Computing

Edge computing represents a departure from the traditional centralized cloud computing model, offering a geo-distributed approach that brings computing resources closer to the network's edge, where data is generated and consumed. The original concept of cutting-edge computing is to provide computing, storage, and network resources close to the user, following open standards and ensuring widespread accessibility (CAO et al., 2020).

The core principles of edge computing can be summarized as follows:

1. **Proximity to Data**: Edge computing situates computational resources closer to data sources at the network's periphery. This minimizes latency and empowers real-time data processing, making it well-suited for applications demanding rapid response times (ADEGBIJA; LYSECKY; KUMAR, 2019).

2. **Redundancy and Reliability**: Edge systems often integrate redundancy and fault-tolerance mechanisms to ensure high availability and reliability, even in challenging and unpredictable environments (BHARDWAJ; KRISHNA, 2021).

3. **Data Filtering and Analysis**: Edge devices can filter, pre-process, and analyze data at its origin. This reduces the volume of data requiring transmission to centralized cloud servers, lowering bandwidth demands (CAO et al., 2020).
4. **Distributed Architecture**: Edge systems adopt a distributed structure composed of local processing nodes or devices. These nodes encompass a variety of components, including IoT devices, edge servers, gateways, and more, enabling data processing at the source (BHARDWAJ; KRISHNA, 2021).

It's worth noting that while fog computing and edge computing both share the common objective of relocating computing and storage resources toward the network periphery and closer to end devices, it's important to note that these paradigms are not synonymous. According to the OpenFog Consortium, edge computing is often called fog computing. However, they draw a distinction, highlighting that fog computing operates hierarchically, delivering computing, networking, storage, control, and acceleration capabilities across a wide spectrum, from IoT devices to the cloud. In contrast, edge computing typically focuses on cutting-edge computational tasks (CHIANG et al., 2017). Furthermore, Chiang et al. (2017) articulate that "fog encompasses cloud, core, edge, customers, and things," emphasizing that fog computing aims to provide continuous support for cloud computing services for IoT devices rather than treating network edges as isolated computing platforms. They also stress that fog computing establishes a horizontal platform designed to facilitate common fog computing functions across diverse sectors and application domains, encompassing traditional telecommunications services and beyond.

2.2.2 Multi-Access Edge Computing

In the context of Edge Computing, Multi-Access Edge Computing, often referred to simply as MEC, is a paradigm that extends the capabilities of edge computing to support multiple access technologies, such as 4G, 5G, B5G, 6G, wired connections, and so on. Unlike traditional cloud computing, where data is sent to distant data centers for processing, MEC pushes computing power to the edge of the network, often within the proximity of base stations or access points (closer to the point of data generation and consumption) (SPINELLI; MANCUSO, 2021). This paradigm, formerly called Mobile Edge Computing, has evolved beyond its initial focus on mobile-specific tasks, expanding its scope to encompass a broader array of applications (MAO et al., 2017).

Both edge computing and MEC services operate at the periphery of the Internet, functioning effectively even with limited or no Internet connectivity. However, MEC differs in that it establishes connectivity through various networks, including WAN, WiFi, and cellular connections. At the same time, edge computing is generally agnostic to the type of connectivity (e.g., LAN, WiFi, cellular) (TALEB et al., 2017). The diverse applications of MEC span a wide range of industries and use cases, including connected vehicles, smart healthcare, video analytics, Augmented Reality (AR), Virtual Reality (VR), smart cities, IoT, and Industry 4.0 (ALI; GREGORY; LI, 2021).

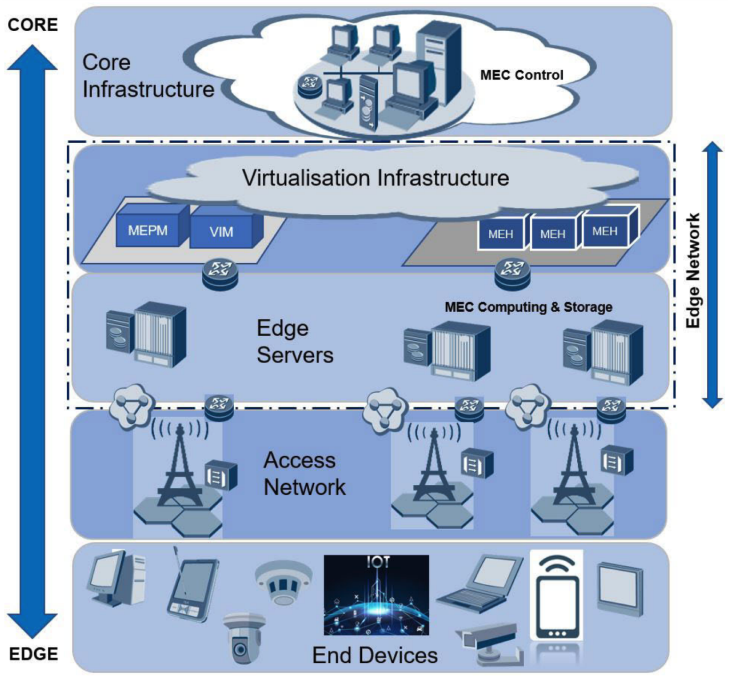


Figure 3 – MEC Functional Structure (ALI; GREGORY; LI, 2021).

As a technology, MEC is poised to play a pivotal role in the next generation of network evolution and act as a critical catalyst for edge applications leveraging 5G networks, as highlighted by the 5G-PPP (QUESETH et al., 2021). The functional framework of MEC consists of four distinct layers (NDIKUMANA et al., 2020): core infrastructure, edge network, access network, and end devices, as illustrated in Fig. 3. Within this framework, the core infrastructure is the central hub for overseeing MEC's control and management functions concerning mobile end devices. The edge network amalgamates the MEC and NFV principles, typically owned by infrastructure providers via multi-tenancy arrangements (MAO et al., 2017). Therefore, MEC can be deployed across multiple edge networks that maintain continuous collaboration and remain interconnected with the traditional cloud infrastructure. The access network is the conduit linking these functional layers to the broader Internet (NADEEM et al., 2021). Within this framework, the layer of end devices encompasses various connected devices, including IoT devices, IP cameras, and mobile terminals (PORAMBAGE et al., 2018).

MEC democratizes access to edge computing, enabling a broad spectrum of mobile devices to benefit from reduced latency and more efficient mobile core networks. Moreover, MEC facilitates the implementation of mission-critical, latency-sensitive applications across mobile networks (MAO et al., 2017). It has also integrated Software Defined Networking (SDN) and NFV capabilities and leveraged 5G technologies. SDN allows for the straightforward management of virtual networking devices through software APIs (HUANG et al., 2017), while NFV streamlines the deployment of networking services through virtualized infrastructure. Furthermore, SDN and NFV enable network engineers and, potentially, enterprise application developers to create their orchestrators, which coordinate resource provisioning across multiple layers (YALA; FRANGOUDIS; KSENTINI, 2018).

2.2.3 Virtual MEC Application Functions

Virtual MEC Application Functions (VMAF) is a joint of concepts and technologies, NFV and MEC, to use the VNF's flexibility and programmability to enable MEC applications to be virtualized, which enables them to be easily deployed and scaled based on end-users demands (SCIANCALEPORE et al., 2016). VMAFs is based in the Virtual Mobile Edge Computing (vMEC) paradigm, which leverages cloud computing resources to provide low-latency and high-performance services at the edge of the network and host it at the fog/edge layer of the network using VNFs (NKENYEREYE et al., 2021).

One key function of VMAF applications is real-time data processing and analytics. By processing data at the network's edge, VMAF applications can reduce latency significantly compared to traditional cloud computing approaches. This capability is crucial for applications such as autonomous vehicles, augmented reality, and industrial automation, where timely decision-making is imperative (CHAUDHRY et al., 2020). Dynamic resource allocation is another important function of VMAF applications. These applications can intelligently distribute computational resources to meet the varying demands of different services and devices at the edge. This function optimizes resource utilization and ensures that critical applications receive the necessary resources to operate efficiently, enhancing the overall quality of service (SCIANCALEPORE et al., 2016).

Additionally, VMAF applications facilitate seamless mobility and connectivity. They can seamlessly migrate and replicate services across different edge nodes, ensuring users experience uninterrupted service even as they move within the network. This capability is vital for applications like video streaming and online gaming, where continuity of service is critical (NKENYEREYE et al., 2021). Security is another critical aspect of VMAF application functions. These applications incorporate robust security measures to protect data and services at the edge. They can implement encryption, access control, and threat detection mechanisms to safeguard against cyber threats, ensuring the integrity and confidentiality of data processed at the edge.

Furthermore, VMAF applications support efficient content delivery. They can cache and deliver content closer to the end-users, reducing the load on the core network and minimizing latency. This function is especially beneficial for content-rich applications, distribution networks, and online services, enhancing the user experience (MA et al., 2022). Consequently, these functionalities underscore the pivotal role of VMAF as an indispensable technology in shaping the future of edge computing.

2.2.4 Internet of Things

The Internet of Things (IoT) has been outlined as "Things that have identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within the social, environmental, and user contexts" (TAN; WANG, 2010). It involves the integration of a multitude of devices and sensors into a coherent network, enabling them to collect, transmit, and exchange data autonomously, ultimately facilitating real-time decision-making and control.

This area holds the promise of making a significant positive impact on the global economy. Projections indicate that the global economic influence of IoT could range from USD 2.7 trillion to 6.2 trillion by the year 2025 (AL-FUQAHA et al., 2015). It represents a vast ecosystem encompassing an array of heterogeneous physical objects, ranging from appliances and facilities to vehicles, farms, factories, and more. The primary goal of IoT is to enhance the efficiency of numerous applications, spanning logistics, manufacturing, agriculture, urban computing, home automation, assisted living, and real-time computing (ALSHARIF et al., 2023).

This comprehensive network of interconnected objects has ushered in a new era of connectivity and automation. Typically, an IoT system adheres to a cloud-centric architecture known as the Cloud-centric Internet of Things (CIoT). In this architectural framework, physical objects are resources managed by central servers (BHOLE et al., 2023). Additionally, devices at the network's periphery connect to the internet via intermediary gateway nodes, including modems, routers, switches, cellular base stations, and similar components (WU et al., 2021).

Despite the common utilization of the CIoT model in IoT implementations, this architecture faces a set of burgeoning challenges in the IoT landscape:

1. **Latency**: Meeting stringent end-to-end latency control requirements poses challenges for the cloud, particularly in industrial smart grid systems, autonomous vehicle networks, virtual and augmented reality applications, healthcare, and elderly care applications, where network latency can have detrimental consequences (BHOLE et al., 2023).

2. **Bandwidth**: The exponential growth in high-frequency data rates generated by IoT objects surpasses the current available bandwidth. For instance, a connected car can produce tens of megabytes of data per second, encompassing route information, speeds, vehicle conditions, driver status, environmental conditions, weather, etc. Autonomous vehicles, in particular, can generate gigabytes of data per second due to real-time video streaming, rendering exclusive reliance on a remote cloud impractical (MINOVSKI; ÅHLUND; MITRA, 2020).

3. **Resource Constraints**: Many IoT devices grapple with limited resources, hindering their ability to execute complex computational tasks and resulting in significant energy expenditure for continuous data transmission to the cloud (BHOLE et al., 2023).

4. **Uninterrupted Connectivity**: The substantial distance between the cloud and IoT devices can lead to unstable and intermittent network connectivity issues. In scenarios like CIoT-based vehicles, disconnections at intermediary nodes between the vehicle and the distant cloud can impede proper functioning (WU et al., 2021).

5. **Security**: A substantial portion of IoT devices may lack sufficient resources to defend against cyber-attacks. Devices reliant on the distant cloud for security software updates can be targeted by attackers, who may also manipulate IoT devices by sending counterfeit data to the cloud (Mohamad Noor; HASSAN, 2019).

In addressing the challenges mentioned above, key technologies such as edge computing, VNF, MEC, and VMAF emerge as pivotal solutions poised to fulfill the requirements of IoT devices (YU et al., 2018). Consequently, IoT stands out as a main beneficiary and consumer of the capabilities offered by edge computing. Ongoing IoT initiatives hold the potential to bridge socioeconomic disparities, enhance the equitable allocation of global resources to those in greatest need, and deepen our comprehension of the planet (BUYYA; SRIRAMA, 2019). This, in turn, enables us to adopt a more proactive rather than reactive stance, addressing edge computing solutions as a possible enabler.

A crucial aspect to highlight is that, in the context of the network's edge and fog environment, a significant attribute is its proximity to end users. This proximity exposes it to human events and migrations and positions it as a facilitator for real-time, ultra-low latency, and critical applications. However, this distinctive feature poses a notable challenge, primarily due to the constrained availability of hardware in these environments. Consequently, achieving optimal resource allocation becomes imperative to unlock the full potential of this setup, which requires optimum auto-scaling solutions.

2.3 Machine Learning for Time-Series Forecasting

In the past decade, there has been a significant surge in interest in handling vast volumes of data. This can be attributed to the widespread adoption of smart sensors and the continuous data generation by social media platforms (PLAGERAS et al., 2018). However, this situation presents new challenges, including data storage on disks and providing necessary computational resources. In this context, the field of big data analytics emerges as a crucial process (SAGIROGLU; SINANC, 2013). It efficiently gathers, organizes, and analyzes large datasets to uncover patterns and extract valuable insights.

This area encompasses numerous research fields. Nevertheless, regression techniques have recently gained significant popularity, primarily driven by advancements in machine learning-based architectures designed specifically to handle temporally indexed data (LARA-BENITEZ; CARRANZA-GARCIA; RIQUELME, 2021). A prime example of this trend is the application of regression methods to time series data, particularly in the domain of time series forecasting (AHMAD; CHEN, 2020).

2.3.1 Time-Series Forecasting

A time series is a sequence of values arranged chronologically and observed over time. While time is a continuously measured variable, the values within a time series are sampled at regular intervals, maintaining a fixed sampling frequency (CHATFIELD, 2003). Given this definition, it's challenging to identify any physical or chemical phenomena that don't involve variables changing over time. Consequently, developing time series forecasting methods is valuable and prevalent across numerous scientific fields.

Time series models can be categorized as either univariate (involving a single time-dependent variable) or multivariate (involving multiple time-dependent variables). For univariate time series, let $y = y(t-L), \ldots, y(t-1), y(t), y(t+1), \ldots, y(t+h)$ represent a given time series with historical data comprising L values. Here, each $y(t-i)$, for $i = 0, \ldots, L$, represents the recorded value of the variable y at time $t-i$. The forecasting process involves estimating the value of $y(t+1)$, denoted as $\hat{y}(t+1)$, to minimize the error, typically represented as a function of $y(t+1) - \hat{y}(t+1)$. This prediction can also extend to situations where the prediction horizon, h, is greater than one, i.e., when the goal is to predict the next h values after $y(t)$, denoted as $y(t+i)$ for $i = 1, \ldots, h$. In such cases, the optimal prediction is achieved by minimizing the function:

$$\sum_{i=1}^{h} (y(t+i) - \hat{y}(t+i))^2 \tag{1}$$

On the other hand, multivariate time series can be represented in matrix form, as in Equation 2, where $y_i(t-m)$ denotes a set of time series with $i = \{1, 2, \ldots, n\}$. Here, m ranges from 0 to L, as $m = \{0, 1, \ldots, L\}$, encompassing historical data and the current sample, and for the future h values, $m = \{-1, -2, \ldots, -h\}$. Typically, one time series is designated as the target series (the one to be predicted), while the remaining ones are called independent time series.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_T \end{bmatrix} = \begin{bmatrix} y_1(t-L) & \dots & y_1(t-1) & y_1(t) & y_1(t+1) & \dots & y_1(t+h) \\ y_2(t-L) & \dots & y_2(t-1) & y_2(t) & y_2(t+1) & \dots & y_2(t+h) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_n(t-L) & \dots & y_n(t-1) & y_n(t) & y_n(t+1) & \dots & y_n(t+h) \end{bmatrix} \tag{2}$$

Time-series forecasting, in turn, is a methodical approach employed to examine time series data using statistical techniques and modeling to anticipate future trends and provide valuable insights for strategic decision-making (LIM; ZOHREN, 2021). It is important to note that these predictions are not always precise, and the accuracy of forecasts can vary significantly, particularly when dealing with variables that tend to fluctuate frequently alongside external factors beyond our control (TORRES et al., 2021). Nevertheless, forecasting offers valuable insights into the probabilities of different potential outcomes. Generally, the more comprehensive and detailed the available data, the more reliable the forecasts tend to be (CHATFIELD, 2003).

Time-series forecasting has many applications, to name a few: financial analysis (SEZER; GUDELEK; OZBAYOGLU, 2019, 2019; YAN; OUYANG, 2017); energy and fuels, such as for power system planning and operating (LI et al., 2019; WU et al., 2019; SHAO et al., 2020); image and video analysis (ATTO; BENOIT; LAMBERT, 2020; WANG et al., 2019b; IENCO et al., 2019); industry, such as process planning (MEHDIYEV et al., 2017), construction equipment recognition (RASHID; LOUIS, 2019) or organization improvements (HUANG; ZANNI-MERK; CRéMILLEUX, 2019; WANG et al., 2019a); and health care diagnosis and prognosis (CHAMBON et al., 2018; BUI et al., 2018; ZEROUAL et al., 2020). Additionally, as investigated in this work, time-series forecasting can be used to predict hardware resources for enabling predictive auto-scaling.

Time-series forecasting algorithms are diverse, each with distinct attributes and applications. Generally, they can be broadly categorized into three main groups: Statistical Methods, Regression-Based Algorithms, and Neural Networks (CHATFIELD, 2003). Statistical Methods consist of algorithms that leverage mathematical operations on historical data to forecast future values in the time series. Notable examples within this category include well-known techniques such as AutoRegressive Integrated Moving Average (ARIMA) and Exponential Smoothing (ETS) (JAIN; MALLICK, 2017). Furthermore, Regression-Based Algorithms and Neural Networks are components of machine learning-based approaches, which play a central role in this study and will be explored further in the subsequent sections.

2.3.2 Machine Learning for Time-Series Forecasting

Machine Learning for Time-Series Forecasting is a cutting-edge and multidisciplinary field combining machine learning algorithms' power with the intricate dynamics of temporal data (ZHOU, 2021). Time-series forecasting is vital in various domains, from finance and economics to weather prediction, energy consumption management, and industrial production planning.

It involves predicting future values in a sequential data set based on historical observations, offering invaluable insights into patterns, trends, and potential changes that influence a time-dependent variable (MAULUD; ABDULAZEEZ, 2020). The application of machine learning techniques in this domain has revolutionized the accuracy and precision of forecasts, offering a dynamic framework to tackle complex, real-world challenges where understanding and predicting temporal relationships are critical (ZHOU, 2021).

Within the realm of Machine Learning for Time-Series Forecasting, there is a diverse range of algorithms and models, each catering to different data characteristics and forecasting goals. This field empowers organizations to unlock deeper insights from their data, leading to optimized decision-making, resource allocation, and risk management (RONG; BAO-WEN, 2018).

From traditional statistical methods like ARIMA and Exponential Smoothing to advanced neural network architectures like Long Short-Term Memory (LSTM) and convolutional approaches, the toolkit for time-series forecasting is vast and adaptable (MAULUD; ABDULAZEEZ, 2020). Given the breadth and diversity of this field, this study embarks on a comprehensive comparison of various machine learning algorithms to identify the most suitable candidate for the predictive auto-scaling framework. To this end, the compared algorithms, Decision Trees (DT), Random Forest, and SVR will be discussed in this section, and the ones based on deep neural networks will be discussed in the next section.

2.3.2.1 Decision Trees (DT) - Regression Trees

A decision tree, akin to a tree structure in computer science (KUNDU; BERTINO, 2008), exhibits a hierarchical arrangement featuring nodes interconnected by edges. It effectively categorizes data by posing questions at each node branchingly. Regression trees are a fundamental machine-learning technique for predictive modeling and data analysis. They belong to the broader category of decision trees, which are versatile tools for classification and regression tasks.

Regression Trees work by recursively partitioning the dataset into subsets based on the values of input features and then assigning a numerical prediction to each subset. Here's a step-by-step explanation of how regression trees operate:

1. Selecting a Feature for Splitting: Constructing a decision tree commences with the entire dataset at the root node. This algorithm assesses each feature to identify the

optimal approach for dividing the data into subsets. The feature selection hinges on a criterion, typically the reduction in mean squared error (MSE). The objective is to pinpoint a feature and a partition point that minimizes the MSE within each resulting subset.

2. Data Splitting: The data is segmented into subsets after determining the best feature and partition point. Each subset represents a branch stemming from the current node. For instance, when predicting house prices based on the number of bedrooms, the data can be segregated into subsets like "bedrooms ≤ 3" and "bedrooms > 3."

3. Iterative Procedure: Constructing the tree extends recursively for each branch (child node). The same methodology is applied at each child node to select the most suitable feature and partition the data. This recursive process continues until a predefined stopping criterion is met. Common halting conditions include reaching a maximum tree depth, having insufficient data points in a node, or failing to achieve a significant reduction in MSE.

4. Assignment of Predictions to Leaf Nodes: Once a terminal node or leaf node is reached, i.e., when the stopping criterion is satisfied, the model assigns a numeric prediction. This prediction usually corresponds to the mean or median of the target variable within the dataset associated with that leaf. This prediction is the estimated value for the instances that lead to that leaf.

5. Traversal and Prediction: To generate a prediction for a new data point, it is traversed through the tree by adhering to the decisions at each internal node, contingent on the feature values of the data point. When it arrives at a leaf node, the prediction linked to that leaf is employed as the final output.

As a Decision Tree algorithm, the Regression Tree is a robust and versatile tool suitable for classification tasks and datasets featuring non-linear relationships. It offers the advantage of straightforward inference and the clear delineation of relationships between variables, making it a relatively intuitive choice compared to some other algorithms (LOH, 2014). Furthermore, Decision Trees can automatically select features, particularly when specific conditions are met, leveraging information gain (LOH, 2011). However, it's more susceptible to overfitting (LOH, 2014), a common issue in machine learning where a model captures noise or intricacies in the training data to such an extent that it fails to generalize well to unseen data, and due that, requests some additional steps to prevent the model overfitting, as hyperparameter tuning, pre-pruning or post-pruning processes (BRAMER, 2013).

2.3.2.2 Random Forest

Random Forest is an algorithm that combines multiple randomized decision trees and synthesizes their predictions through averaging. It has demonstrated remarkable effectiveness in scenarios where the count of variables greatly surpasses the number of observations (BREIMAN, 2001).

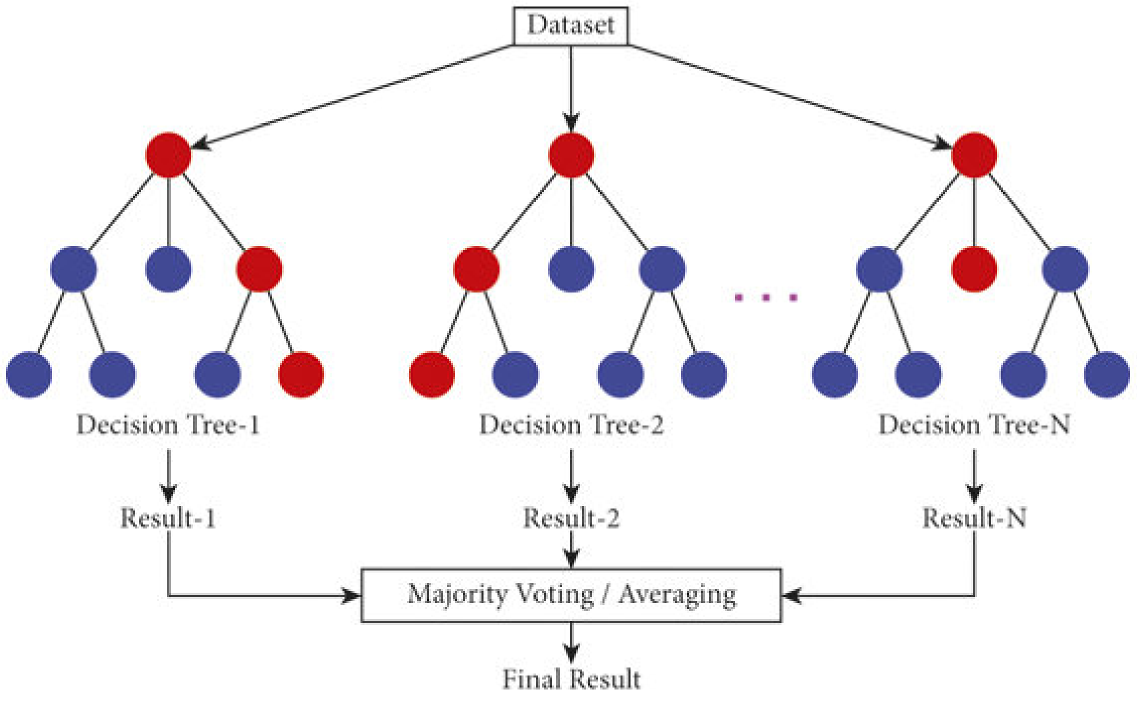


Figure 4 – Random Forest trees structure (KHAN et al., 2021).

As presented in Fig. 4, the Random Forest is a robust ensemble learning algorithm with a foundation in tree-based methods. It assembles a collection of decision trees, each constructed from a randomly chosen subset of the training dataset. These trees work in unison by casting their votes to determine the final class assignment for a given test object. The Random Forest algorithm generates decision trees from diverse data samples, accumulates predictions from each tree, and ultimately arrives at the most favorable outcome through a voting mechanism.

To grasp the functioning of the Random Forest algorithm, we can break it down into the following steps: (1) initiates with the random selection of samples from the dataset, while (2) proceeds to construct a decision tree for each of these samples, subsequently obtaining prediction results from every tree. Moving on to (3), the algorithm engages in a voting process, where each predicted result contributes to the decision-making. Finally, in (4), the algorithm culminates by selecting the prediction result that garners the most votes as the ultimate prediction result.

Random Forest is a potent ensemble machine learning algorithm extensively employed in predictive tasks (BELGIU; DRAGUT, 2016). It excels at averting the problem of overfitting, a common pitfall in single decision tree models, by aggregating predictions from multiple trees and utilizing random data subsets for each tree, thereby exhibiting a propensity for effective generalization on unseen data (SHEYKHMOUSA et al., 2020). It also furnishes a valuable metric for gauging the significance of different features, facilitating feature selection, and enhancing interpretability. Furthermore, the parallelizability of Random Forest is a notable advantage, making it well-suited for high-performance computing and the analysis of large datasets (BELGIU; DRAGUT, 2016).

Nonetheless, the algorithm does come with certain drawbacks. Its computational demands and time requirements can be substantial, especially when dealing with many trees or features (SHEYKHMOUSA et al., 2020). This complexity can restrict its application in real-time scenarios or resource-constrained environments. Moreover, Random Forest may encounter challenges when handling imbalanced datasets, where the majority class tends to dominate the feature selection process, potentially hampering the model's ability to learn from minority class samples (ANTONIADIS; LAMBERT-LACROIX; POGGI, 2021). Additionally, the storage requirements can be onerous, particularly for models with many trees, posing potential difficulties in resource-constrained settings (SHEYKHMOUSA et al., 2020).

2.3.2.3 Support Vector Regression (SVR)

Support Vector Regression (SVR) by (SCHOLKOPF; SMOLA, 2001) is a variant of Support Vector Machines (SVM) and is designed to forecast continuous values, making use of the concept of support vectors and aiming to find the optimal hyperplane to separate data points. It manages the complexity of finding a hyperplane that best fits the data by adding a penalty term to the error function and allowing a margin of error. To illustrate this method using a linear model, let's consider the prediction formula:

$$f(x) = w^T x + b \tag{3}$$

In this equation, w represents the weight vector, b denotes the bias, and x is the input vector. For training data, where $m = 1, 2, \ldots, M$, we have x_m as the m-th training input vector and y_m as the corresponding target output. The error function is defined as:

$$J = \frac{1}{2}\|w\|^2 + C \sum_{m=1}^{M} |y_m - f(x_m)|_\epsilon \tag{4}$$

Here, the first term in the error function penalizes model complexity, while the second term is the ϵ-insensitive loss function. The ϵ-insensitive loss function, $|y_m - f(x_m)|_\epsilon$, is defined as $|y_m - f(x_m)|_\epsilon = \max\{0, |y_m - f(x_m)| - \epsilon\}$. It does not penalize errors that fall below ϵ, providing some flexibility for parameter adjustments to reduce model complexity. The optimal solution for minimizing the error function is as follows:

$$f(x)^* = \sum_{m=1}^{M} (\alpha_m^* - \alpha_m) x_m^T x + b \tag{5}$$

Here, α_m and α_m^* are Lagrange multipliers. Training vectors with non-zero Lagrange multipliers are referred to as support vectors, a fundamental concept in SVR theory. Non-support vectors do not directly influence the solution, and the count of support vectors serves as a measure of model complexity (ZHANG; O'DONNELL, 2020).

This model extends to the nonlinear case through the use of the kernel function K, resulting in the solution:

$$f(x) = \sum_{m=1}^{M} (\alpha_m^* - \alpha_m) K(x_m, x) + b \tag{6}$$

One common kernel is the Gaussian kernel (Assuming a width of σ_K (the standard deviation of the Gaussian function)). However, to achieve optimal results, especially when dealing with diverse and complex datasets, it is imperative to follow a meticulous approach like the one outlined in this work. This approach involves carefully defining hyperparameters, as exemplified in (YANG; SHAMI, 2020), experimenting with various kernel functions to identify the one that most effectively captures the data's underlying patterns and relationships.

2.3.3 Deep Learning for Time-Series Forecasting

Deep Learning for Time-Series Forecasting represents a cutting-edge approach that harnesses the power of artificial Neural Networks (NN) to decipher intricate temporal patterns and deliver highly accurate predictions for sequential data (LIM; ZOHREN, 2021). This field has gained prominence due to its ability to handle complex time-dependent relationships, making it a promising avenue for applications in finance, healthcare, weather prediction, and various other domains (TORRES et al., 2021). Deep learning models, such as Recurrent Neural Networkss (RNNs) and Long Short-Term Memorys (LSTMs), can capture nuanced dependencies in time-series data, enabling organizations to make informed decisions and anticipate future trends with unprecedented accuracy (LIM; ZOHREN, 2021).

Comparing Deep Learning with traditional Machine Learning approaches for time-series forecasting reveals some fundamental distinctions. While both approaches aim to make predictions based on historical data, deep learning models usually perform better in handling long-range dependencies and intricate temporal relationships (AHMED et al., 2010). Deep learning leverages the hierarchical structure of neural networks to uncover hidden patterns within data, which can be particularly advantageous when dealing with intricate, multidimensional time-series data (TORRES et al., 2021). However, deep learning models tend to be computationally more intensive. They may require larger datasets, making them most effective when dealing with complex, high-dimensional time-series data where their capacity to capture intricate patterns shines. In contrast, traditional machine learning methods are often more interpretable. They can be effective in scenarios with less complex data patterns, but their performance may wane as data complexity increases (LECUN; BENGIO; HINTON, 2015a).

In essence, the choice between deep learning and traditional machine learning for time-series forecasting should be driven by the specific characteristics and requirements of the data and the complexity of the forecasting task at hand (SHINDE; SHAH, 2018). This is why this work conducts an empirical evaluation, comparing multiple machine learning and deep learning algorithms to identify the most suitable for the proposed auto-scaling application.

The toolkit for time-series forecasting based on deep learning is vast and adaptable (MAULUD; ABDULAZEEZ, 2020), so this study embarks on a comprehensive comparison of various machine learning algorithms to identify the most suitable candidate for the predictive auto-scaling framework. To this end, continuing to present the compared algorithms, GRU, LSTM, Bi-LSTM, CNN-LSTM, Enc-Dec-LSTM, and Enc-Dec-CNN-LSTM algorithms will be discussed in this section.

2.3.3.1 Gated Recurrent Unit (GRU)

In essence, Recurrent Neural Networkss (RNNs) are better suited for capturing relationships within sequential data types. The basic RNN, often referred to as the simple RNN, incorporates a recurrent hidden state described by the equation:

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{7}$$

Here, x_t represents the external input vector at time t with dimension m, h_t signifies the hidden state with dimension n, g is an element-wise activation function like the logistic function, hyperbolic tangent function, or Rectified Linear Unit (ReLU), and W, U, and b denote appropriately sized parameters, comprising two weight matrices and a bias vector. Specifically, W is an $n \times m$ matrix, U is an $n \times n$ matrix, and b is an $n \times 1$ matrix (or vector). Capturing long-term dependencies with simple RNNs is challenging due to the tendency of stochastic gradients to either vanish or explode in lengthy sequences (LECUN; BENGIO; HINTON, 2015b). To address these issues, two specific models, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been introduced to mitigate the problems of vanishing and exploding gradients.

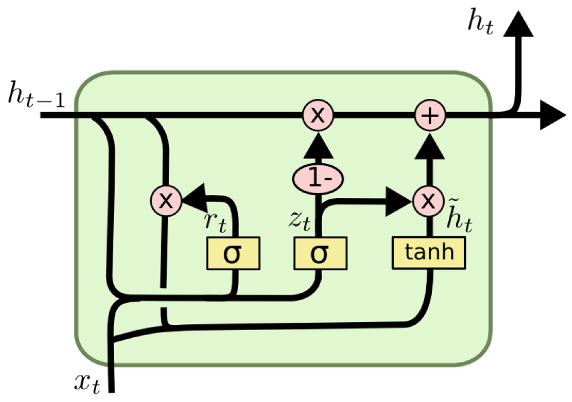


Figure 5 – GRU cell architecture.

The Gated Recurrent Unit (GRU) algorithm, aiming to improve the RNN performance when dealing with long sequences, includes a hidden state in its cell core (as represented by h in Fig. 5). Additionally, to manage this additional and long-term persisted state, the cell core uses a gating mechanism to manipulate the state value, using two gates: an update gate, denoted as z_t, and a reset gate, r_t. The GRU model is illustrated in Fig. 5 and can be formulated as follows:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{8}$$

$$\tilde{h}_t = g(W_x x_t + U(r_t \odot h_{t-1}) + b) \tag{9}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{10}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{11}$$

In equation (8), h_t represents the output value of the hidden state, in (9), $\tilde{h}_t$ represents the value to be changed in hidden state, and in equations (10)-(11), z_t and r_t denotes the two gates aforementioned. In these equations, it is to be noted that the activation nonlinearity σ, is typically the hyperbolic tangent function, although more recent implementations may use the ReLU. The weighted sum in equation (9) is computed through element-wise (Hadamard) multiplication, denoted by $\odot$, for controlling the gating signals. The gating signals z_t and r_t represent the gating signals at time t. These control gating signals essentially mirror the core equation (9), each with its own set of parameters while replacing σ with the logistic function, which confines the gating signals within the range of 0 and 1.

In summary, the GRU algorithm incorporates a threefold parameter increase compared to the simple RNN as presented in Equation (7). More precisely, the total number of parameters in the GRU is given by $3 \times (n^2 + n \cdot m + n)$. However, numerous studies, such as (SHEWALKAR, 2018), have indicated that the GRU model outperforms a simple RNN in most scenarios, mainly when dealing with long-term data sequences.

2.3.3.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is also a type of RNN architecture designed to address the vanishing gradient problem in traditional RNNs.The key feature of the LSTM model is its ability to capture and remember information over long sequences, making it well-suited for tasks like natural language processing, speech recognition, time series forecasting, and more (ZHANG et al., 2021). As the GRU as mentioned earlier algorithm, LSTMs achieve this by using a sophisticated gating mechanism, however, it uses three gating mechanisms, including input, forget, and output gates. These three gates allow the model to control when to retain, update, or forget information.

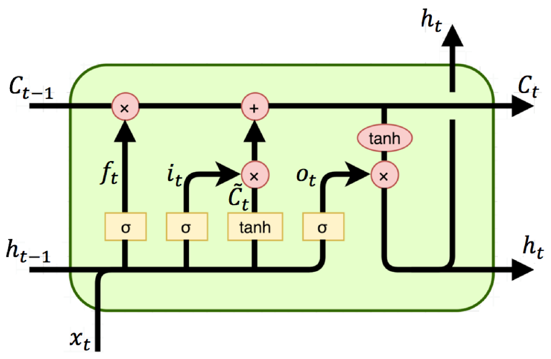


Figure 6 – LSTM cell architecture.

The cell architecture of a LSTM cell, as illustrated in Fig. 6, leverages the computation performed by the simple RNN, as described in Eq. (7), to establish an intermediate candidate for the internal memory cell, denoted as c_t. This intermediate candidate, represented as $\tilde{c}_t$, is combined with the previous internal memory state, c_{t-1}, through an element-wise weighted sum operation. This process yields the current value of the memory cell, c_t, which can be concisely expressed by the following discrete dynamic equations:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{12}$$

$$\tilde{c}_t = g(W_c x_t + U_c h_{t-1} + b_c) \tag{13}$$

$$h_t = o_t \odot g(c_t) \tag{14}$$

In equations (13) and (14), the activation nonlinearity g is typically the hyperbolic tangent function (tanh). However, it can also be implemented with different activation functions, for example, a ReLU operation. The weighted sum operation in Equations (12) and (14), as in GRU, is realized through element-wise (Hadamard) multiplication, denoted by $\odot$, involving the gating signals. The gating signals i_t, f_t, and o_t correspond to the input, forget, and output gating signals at time t. Notably, these gating signals are essentially governed by the basic Equation (14), each having its distinct set of parameters and replacing g with the logistic function. The logistic function confines the gating signals to 0 and 1. The specific mathematical representation of the gating signals can be expressed as vector equations:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{15}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{16}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{17}$$

Here, σ denotes the logistic nonlinearity, and each gate comprises two matrices and a bias vector as parameters. Consequently, the total number of parameters for the three gates and the memory cell structure is as follows: $W_i, U_i, b_i, W_f, U_f, b_f, W_o, U_o, b_o, W_c, U_c, b_c$. All these parameters are updated and stored at each training step.

The number of parameters in the LSTM model is four times greater than that in the simple RNN model presented in Equation (7). Assuming that the cell state is n-dimensional and the input signal is m-dimensional, the total parameters in the LSTM RNN model can be expressed as $4 \times (n^2 + n \cdot m + n)$. However, numerous studies, such as (SHEWALKAR, 2018), have indicated that the LSTM model outperforms a simple RNN in most scenarios, mainly when dealing with long-term data sequences. This happens because, using the hidden state, LSTM can efficiently manage and learn using past information and patterns.

2.3.3.3 Bidirectional LSTM (Bi-LSTM)

While the LSTM cell effectively leverages past information within a sequence to learn data patterns and information, it cannot utilize future information. To clarify, within an input sequence like $[..., t-2, t-1, t, t+1, t+2...]$, at time x when the LSTM model is evaluating the value at t, it is influenced by past values $t-1, t-2, \ldots$ and learns from these historical patterns.

However, the LSTM architecture does not take advantage of the future information, specifically $t+1, t+2, \ldots$, which is already present in the dataset during this period at time x. To address these limitations of the LSTM cell, in (SCHUSTER; PALIWAL, 1997), the authors introduced Bidirectional Recurrent Neural Networks (BRNN). The BRNN comprises two distinct LSTM hidden layers that produce analogous outputs but in opposing directions. This innovative architecture enables the incorporation of both preceding and forthcoming information in the output layer.

Evolving this architecture, the Bidirectional Long Short-Term Memory (Bi-LSTM) architecture, as presented in Fig. 7, consists of two unidirectional LSTMs that analyze

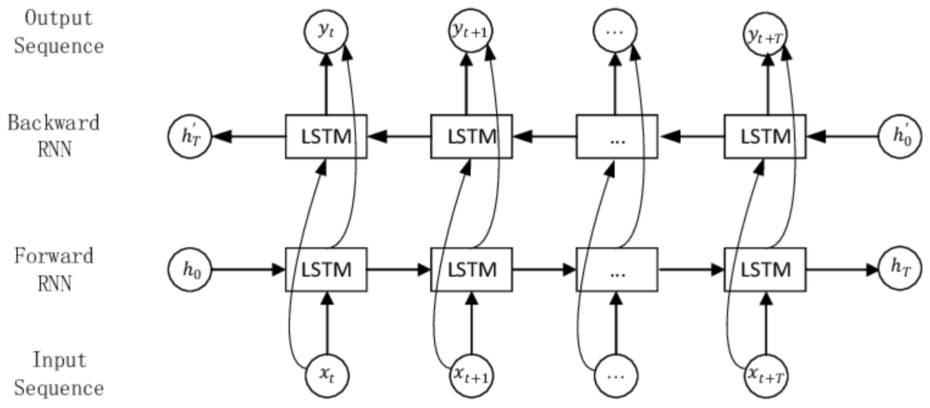


Figure 7 – BI-LSTM architecture (XIANG et al., 2020).

a sequence in both forward and reverse directions. This arrangement can be thought of as two distinct LSTM networks: one receives the input sequence in its original order, while the other processes it in reverse (HUANG; XU; YU, 2015). Each LSTM network generates a probability vector as its output; the ultimate result is a fusion of these two sets of probabilities. In a Bi-LSTM, an input sequence $X = (X_1, X_2, \ldots, X_m)$ is computed in both the forward direction, resulting in $\overrightarrow{h_t} = (\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_m})$, and in the backward direction, resulting in $\overleftarrow{h_t} = (\overleftarrow{h_1}, \overleftarrow{h_2}, \ldots, \overleftarrow{h_n})$. The final output of this cell, denoted as y_t, in LSTM defined in equation (14), in Bi-LSTM is formed by combining both $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$. Consequently, the final output sequence appears as $y = (y_1, y_2, \ldots, y_t, \ldots, y_n)$.

In conclusion, Bidirectional LSTM can offers a versatile and powerful solution for modeling sequences, its ability to capture long-range dependencies and make use of both preceding and forthcoming information in the output layer, proved great adaptability in handling complex architectures in works as in Siami-Namini, Tavakoli e Namin (2019), Song et al. (2022), Raihan e Ahmed (2023).

2.3.3.4 Convolutional Neural Network LSTM (CNN-LSTM)

The Convolutional Neural Networks (CNN) is a neural network model introduced in (LECUN et al., 1998). It is a type of Feedforward Neural Network (FNN), in which the information is only processed in one direction, with connections between nodes not forming cycles. CNN primarily consists of two key components: the convolution layer and the pooling layer. Each convolution layer comprises multiple convolution kernels. After the convolution operation in the convolution layer, data features are extracted. However, these extracted features often have high dimensions. To address this issue and reduce the network training cost, a pooling layer is typically added following the convolution layer to decrease the feature dimensions. The formula for the convolution operation is given as:

$$l_t = \tanh(x_t * k_t + b_t) \tag{18}$$

In Equation 18, l_t represents the output value after convolution, tanh denotes the activation function, x_t is the input vector, k_t signifies the weight of the convolution kernel, and b_t represents the bias of the convolution kernel. The CNN model is renowned for its effectiveness in image processing and natural language tasks (CHUA; ROSKA, 1993) tasks focusing on classification problems.

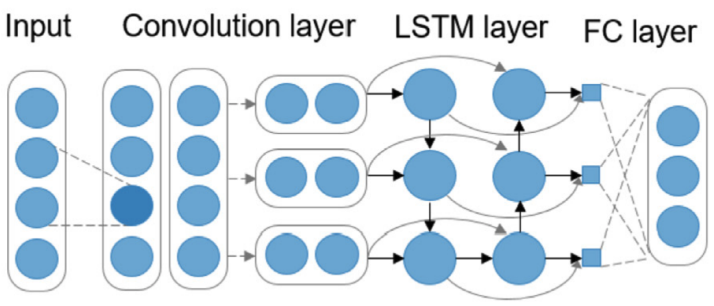


Figure 8 – CNN-LSTM architecture.

Combining the classification effectiveness of CNNs with the ability of LSTM architecture to capture temporal dependencies, the CNN-LSTM model is formed by connecting these two models in a cascaded connection (KIM; CHO, 2019), as presented in Fig. 8. This method facilitates the extraction of intricate patterns and the capture of complex irregular trends (XU et al., 2023). In the CNN-LSTM architecture, the initial segment comprises a CNN layer, which is equipped to receive and handle a diverse range of features and is responsible for extracting the ’meaning’ of the data through the classification. The convolutional layer applies the convolution operation to the incoming multivariate time series data and passes the results to the subsequent layer. This operation is crucial in reducing the number of parameters and enhancing the depth of the CNN-LSTM network (KIM; CHO, 2019). Furthermore, the output is passed to one or more LSTM layers responsible for extracting the temporal references and operations from the input data. The RNN-based layer uses the aforementioned hidden-layer and activation function mechanisms to abstract the time implications from the preceding data. It passes the results to the subsequent layer (TASDELEN; SEN, 2021).

The final layer in the CNN-LSTM architecture is usually, but not necessarily, composed of fully connected layers, as in the example illustrated in Fig. 8. This layer flattens the output from the LSTM unit into a feature vector, denoted as $\mathbf{h}^{(p)}$, where p represents the number of units within the LSTM network. This feature vector is responsible for encapsulating valuable information patterns and trends over time, improving and facilitating accurate forecasting.

2.3.3.5 Encoder-Decoder LSTM (Enc-Dec-LSTM)

In time-series forecasting, a sequence prediction usually involves forecasting the subsequent value in a sequence given input series. This task is typically constrained as one-to-one (one input step corresponds to one output time step) or many-to-one (multiple input steps lead to a single output time step) sequence prediction. However, in some cases, a more complex sequence prediction challenge arises when the desired output is also a sequence, which is categorized as a sequence-to-sequence prediction problem (CHO et al., 2014). One of the primary complexities in handling such problems is the potential variation in the lengths of both the input and output sequences. Given that multiple input time and output time steps are involved, this problem falls under the category of many-to-many sequence prediction (BADRINARAYANAN; KENDALL; CIPOLLA, 2017).

Aiming this problem, encoder-decoder architectures are a class of neural network models that is composed of the unification of two other network models, denoted encoder and decoder, that have the following main roles (BADRINARAYANAN; KENDALL; CIPOLLA, 2017): (1) Encoder: The encoder part is responsible for processing the input data and encoding it into a fixed-length vector or a series of vectors. It captures the essential information from the input, creating a context for generating the output. For example, the encoder processes the source language text in machine translation. (2) Decoder: The decoder takes the encoded information from the encoder and generates the output, often as a data sequence. It conditions its predictions on the encoded context, allowing it to produce meaningful and contextually relevant results. In the context of machine translation, the decoder produces the translated target language text.

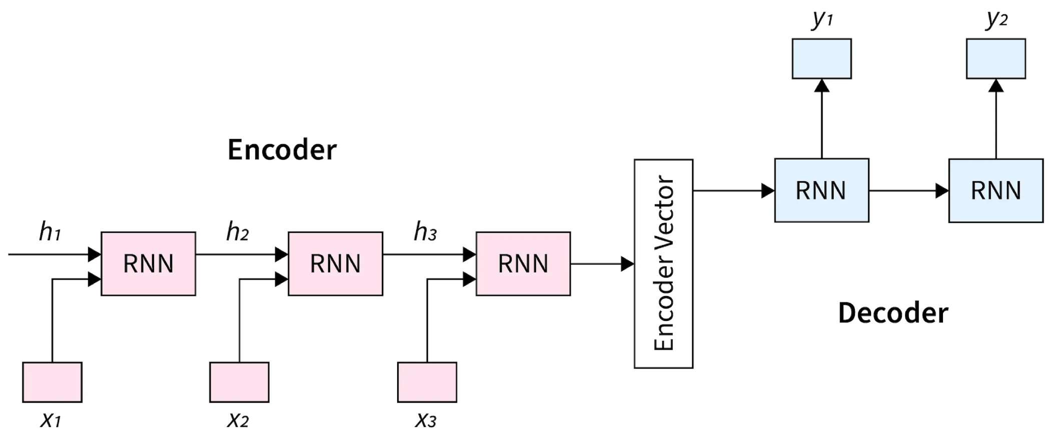


Figure 9 – Encoder-Decoder architecture.

The Encoder-Decoder LSTM (Enc-Dec-LSTM) model is a type of encoder-decoder deep learning architecture that employs LSTM units, an RNN-based architecture, as both encoder and decoder. Figure 9 illustrates an instance of an encoder-decoder architecture utilizing RNN models. This architecture comprises two distinct RNN components: an encoder with three layers and a decoder with two layers. It's important to highlight that, in this particular example, a flattening operation is employed during the transition from the encoder network to the decoder network. This operation is performed to structure the data and ensure its compatibility with the decoder network as input. This practice is commonly adopted to prevent any contextual information from the encoder network from unintentionally affecting the decoder network (BADRINARAYANAN; KENDALL; CIPOLLA, 2017).

Encoder-decoder architectures, such as the Enc-Dec-LSTM, are particularly effective for tasks involving sequences, where the input and output length can vary (WANG; SU; DING, 2021). They have been successfully applied to various natural language processing tasks, including machine translation, text summarizing, and image-to-text (where the encoder encodes an image, and the decoder generates its textual description) or image-related tasks (CHEN et al., 2018; SERBAN et al., 2017; CHO et al., 2014).

2.3.3.6 Encoder-Decoder CNN-LSTM (Enc-Dec-CNN-LSTM)

The Encoder-Decoder CNN-LSTM (Enc-Dec-CNN-LSTM), similarly to the last presented algorithm, Enc-Dec-LSTM, follows the encoder-decoder paradigm in deep learning. However, this architecture combines a CNN unit as the encoder component and an LSTM unit as the decoder component. The concept behind this model is to harness the strengths of a CNN-LSTM structure while incorporating the encoder-decoder framework (VOSOUGHI; VIJAYARAGHAVAN; ROY, 2016).

This algorithm architecture is particularly effective for tasks involving long sequences, multiple outputs, and classification (HAQUE; YOUSUF; RANA, 2018). It has been successfully applied to various tasks, such as natural language processing (VOSOUGHI; VIJAYARAGHAVAN; ROY, 2016), image denoising and restoration (HAQUE; YOUSUF; RANA, 2018), and image forgeries detection (BAPPY et al., 2019).

2.4 Auto-Scaling and Resource Management

In today's rapidly evolving technological landscape, where the demand for online services and applications can strongly vary, auto-scaling and resource management have become crucial (QU; CALHEIROS; BUYYA, 2018). Auto-scaling is a dynamic and adaptive approach to provisioning computing resources that allows systems to adjust their capacity to meet changing workloads (MAO; HUMPHREY, 2011). This is a critical aspect for both cloud computing and edge-based systems, for example, IoT, VNFs, and MEC applications. It enables efficient handling of fluctuations in user traffic and optimizing resource utilization while ensuring the stability and reliability of their services (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014a).

2.4.1 Auto-Scaling

Cloud-hosted applications encompass a wide spectrum of systems, ranging from web applications to batch jobs, map-reduce tasks, video streaming services, and many others. To enable this vast extension, a key characteristic of cloud computing is elasticity (DURAO et al., 2014). However, this attribute presents a duality, for while it empowers applications to seamlessly scale resources up or down, aligning with the workload demands, the fine art of resource allocation remains a complex challenge. The most desirable item for this context would be a system that can autonomously and intelligently adapt resources to the workload that an application handles while minimizing the need for human intervention or even surpassing it (SRIVASTAVA; KHAN, 2018).

Both cloud-hosted and edge or fog-placed, as IoT devices, VNFs, and MEC applications imply a wide variety of applications that request dynamic resource management and auto-scaling. For example: (1) IoT devices generate vast data, so an auto-scaling mechanism is critical for handling the variable and often unpredictable loads generated by these devices. A smart city, for example, with thousands of IoT sensors, may experience peaks in data flow during certain events or times of the day. Auto-scaling ensures that the necessary computing resources are provisioned to process this data in real-time (VERMA; BALA, 2021). (2) MEC brings computing resources closer to the network edge, reducing latency and improving responsiveness. Auto-scaling is used to manage the allocation of these edge resources. As the number of edge nodes or devices connecting to MEC increases, auto-scaling ensures that sufficient compute capacity is available at the edge to meet processing and storage needs. MEC can cache and deliver content at the edge, reducing the load on central data centers. Auto-scaling is vital in ensuring that the edge servers automatically scale up or down to handle content delivery requests efficiently, especially during peak demand periods (LEE et al., 2021). (3) 5G introduces network slicing, where a single physical network can be divided into multiple virtual networks with different characteristics. Auto-scaling plays a crucial role in creating and managing these network slices, allowing for dynamic allocation and optimization of resources based on the unique requirements of each slice. 5G's low latency capabilities enable real-time communication, essential for autonomous vehicles and telemedicine applications. Auto-scaling ensures that the computing infrastructure can respond to fluctuating data rates and maintain low latency even during peak usage (REN et al., 2016).

Traditional, static provisioning methods often lead to over-provisioning, resulting in underutilized resources and inflated costs, or under-provisioning, causing performance bottlenecks and service disruptions during traffic spikes (HEINZE et al., 2014). Auto-scaling addresses these challenges by enabling systems to adapt automatically, scaling resources up or down as needed, thereby ensuring optimal performance, efficient resource utilization, and cost-effectiveness (QU; CALHEIROS; BUYYA, 2018). Moreover, auto-scaling gives the following benefits:

- **Optimal Resource Utilization:** Auto-scaling ensures that computing resources are used efficiently. It allows organizations to scale up during periods of high demand and scale down during lulls, minimizing the waste of resources and reducing operational costs.

- **High Availability:** Auto-scaling enhances system reliability by automatically adjusting capacity. If one part of the infrastructure fails, the system can quickly recover by launching new instances or nodes, ensuring high availability.

- **Improved Performance:** Applications can maintain consistent performance levels even during traffic spikes. Auto-scaling dynamically allocates additional resources, such as more virtual machines or containers, to handle increased workloads.

- **Cost Optimization:** By scaling resources according to demand, organizations can avoid overprovisioning, which can be expensive. Auto-scaling allows businesses to pay for what they use and reduce infrastructure costs.

- **Response to Dynamic Workloads:** In scenarios with unpredictable workloads or variable patterns, such as e-commerce websites during sales events, auto-scaling adapts quickly to accommodate the changing traffic.

2.4.1.1 Auto-Scaling Types and Strategies

Resource scaling can take two primary forms: horizontal scaling and vertical scaling (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014a). In horizontal scaling, the fundamental unit of resource adjustment is the server replica, typically running within a Virtual Machine (VM). New server replicas can be seamlessly added or removed when demand fluctuates to accommodate the workload variations. On the other hand, vertical scaling, also known as scaling up or down, involves modifying resources allocated to an existing VM, such as increasing or decreasing CPU power or memory allocation.

Auto-scaling can be implemented through various strategies tailored to specific use cases and requirements. The most common strategies include but is not limited to:

- **Reactive Scaling:** This strategy responds to predefined threshold values, scaling resources up or down when certain metrics cross critical levels. For example, if CPU utilization exceeds 80%, additional VM instances may be provisioned.

- **Predictive Scaling:** Predictive scaling uses historical data and machine learning algorithms to forecast future resource requirements. This approach is more proactive, allowing systems to scale before performance degrades.

- **Queuing theory:** Queuing theory, a mathematical framework dedicated to the precise analysis of waiting for lines or queues, plays a crucial role in modeling the behavior of systems. When a client request enters the system with an average arrival rate of k, it is queued until it proceeds for processing. However, two primary drawbacks are associated with the application of queuing theory (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014a). Firstly, it relies on certain assumptions that may not always hold in real-time environments, making it less suitable for practical applications. Secondly, queuing theory may not be the best fit for systems that demand high levels of criticality.

- **Scheduled Scaling:** In situations where traffic patterns are known in advance, scheduled scaling can be employed. This is one of the simplest strategies and allows for automatically adjusting resources based on a predetermined schedule, such as scaling up during peak business hours and down during off-peak times.

Additionally, some auto-scaling frameworks employ hybrid approaches, which use multiple strategies simultaneously or interspersed.

2.4.1.2 Auto-Scaling Components

The auto-scaling mechanisms often involve a combination of hardware and software components that adjust resource allocation dynamically (CASTRO et al., 2019). Their primary components include:

- **Control Plane:** The control plane manages the auto-scaling policies and triggers the scaling actions. It takes inputs from the monitoring systems and applies predefined rules and policies to determine when and how scaling should occur.
- **Monitoring and Metrics:** Various metrics and monitoring systems are employed to determine when scaling actions are necessary. These could include CPU utilization, memory usage, network traffic, or custom application-specific metrics.
- **Scalable Resources:** The scaled resources are typically virtual machines (VMs), containers, or serverless functions. These resources are grouped into instances, and scaling decisions affect these groups.

In the next sub-sections, we delve into these components, highlighting the main concepts applied in this work. It is worth highlighting that this work proposes a solution that plays a control plane role in the auto-scaling context.

2.4.2 Virtualization

The scalable resources for an auto-scaling mechanism are usually related, but not limited, to virtualized instances. Scalable resource management refers to the ability to dynamically allocate and distribute resources (such as computing power, memory, storage, and network bandwidth) as needed to meet the demands of an application or system (QU; CALHEIROS; BUYYA, 2018). The primary goal is to ensure that resources are used efficiently and that system performance remains optimal even when workloads change.

Aiming at these scalable requirements, virtualization is a main enabler once it empowers the simultaneous operation of multiple operating systems or applications atop a solitary physical infrastructure. This technology accomplishes it by providing each application an abstract representation of the hardware (MANSOURI; BABAR, 2021). Consequently, these applications or operating systems can function in isolation while utilizing the same hardware resources efficiently.

VNF started as a cloud enabler while many applications and predominantly all network devices relied on proprietary and tailored software. The remarkable success of server virtualization enticed network operators to explore this solution in edge environments (BEHRAVESH; CORONADO; RIGGIO, 2019). Consequently, this shift prompted equipment suppliers and manufacturers to depart from the rigid confines of personalized, single-purpose devices to a virtualized environment (BHARDWAJ; KRISHNA, 2021). Nowadays, virtualization plays a vital role not only in the context of cloud computing but also is fundamental to enable edge computing in NFV infrastructure and MEC environment (BEHRAVESH; CORONADO; RIGGIO, 2019). Therefore, an auto-scaling application can leverage the broad utilization of virtualization mechanisms to use them as scalable resources.

Virtualization can be consolidated using Full, Paravirtualization, Hardware-Assisted, and OS-level virtualization (SIERRA-ARRIAGA; BRANCO; LEE, 2020). These last two showed themselves as the most prominent scalable resource (KUKADE; KALE, 2015) and have virtual machines and containerization as the main examples. These technologies are broadly applied in cloud and edge computing and are commonly used as scaling objects of auto-scaling solutions. However, the choice of virtualization technologies can have pros and cons and impact the overall performance results of the auto-scaling frameworks. To this end, we delve into both technologies in the following sections.

2.4.2.1 Virtual Machines (VMs)

Virtualization establishes a segregated virtual hardware environment where an operating system or application operates independently. This resulting environment is commonly called a Virtual Machine (VM) (ZHANG et al., 2018). As highlighted in Fig. 10 (left image), the VM environment comprises three key components: the hypervisor (or virtual machine manager), the host operating system, and the guest operating system (or server).

The host operating system, often termed the host OS, runs directly on the physical hardware, must support virtualization, and has the necessary applications installed (TAO et al., 2019). The host OS maintains full visibility over the complete hardware resources

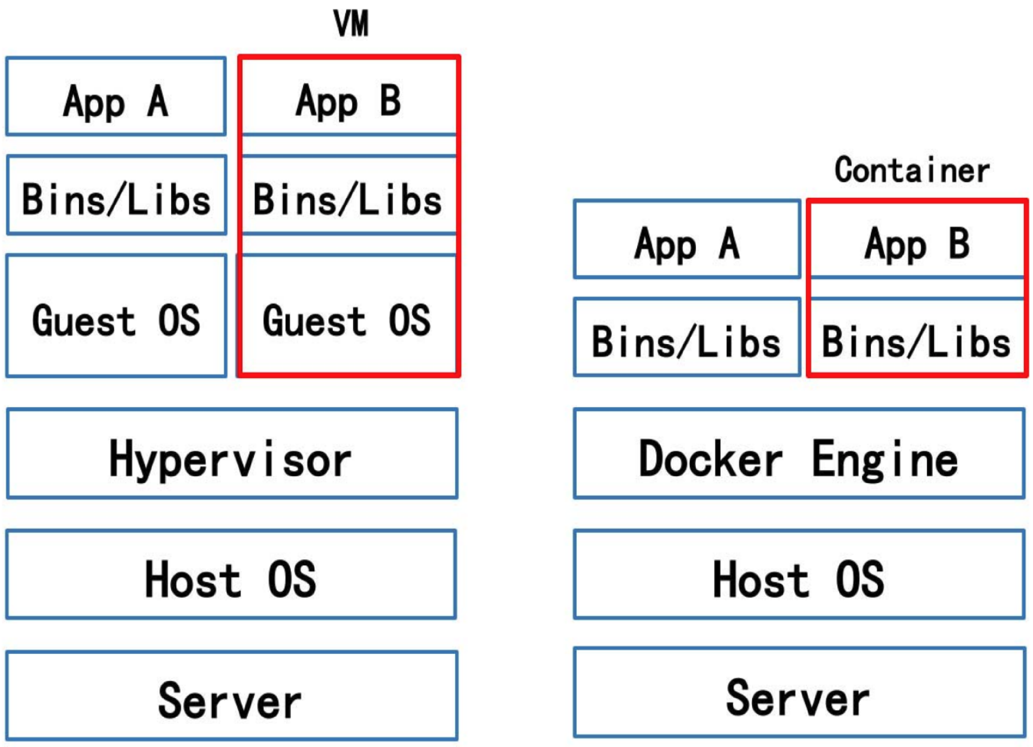


Figure 10 – Virtual machines and containerization architecture comparison.

available for allocation to virtual machines. The hypervisor facilitates the creation of virtual machines, resource allocation, parameter adjustment, and the removal of virtual machines (ZHANG et al., 2018). Finally, the operating system running within this virtual server can vary in type but must be compatible with the hardware capabilities offered by the hypervisor. Importantly, the virtual machine has no insight into the actual hardware resources or awareness of other virtual machines that may be present under the management of another instance of the hypervisor (AALAM; KUMAR; GOUR, 2021), creating the illusion that the guest operating system communicates directly with the physical hardware components.

Once this technology leverages a guest OS, it incurs greater resource consumption and may require longer startup times for new instances compared to other virtualization approaches (ZHANG et al., 2018).

2.4.2.2 Containerization

The virtual machines outlined in the last section offer a highly isolated and self-contained environment. However, it's important to note that there is an associated performance cost and resource overhead incurred due to the emulation of virtual hardware (POTDAR et al., 2020a). In certain scenarios, though, the isolation level provided by virtual machines may prove excessive, and a more balanced compromise with a lighter form of isolation might be entirely suitable. This objective can be accomplished by utilizing a less complex virtualization approach known as container-based virtualization (RANDAL, 2020).

Container-based virtualization is an OS-level virtualization technique that offers a controlled, confined, and segmented environment for running applications within a host operating system (CASALICCHIO, 2019). As pictured in Fig. 10 (right image), different from VMs, containers virtualize at the operating system level, sharing the host operating system's kernel and abstracting the user space, allowing multiple containers to run on the same OS instance. The ability to employ this form of virtualization stems directly and originates in the concerted efforts to establish kernel-level support for isolation between applications (BESERRA et al., 2017).

Containers are highly efficient in resource utilization once they have a smaller footprint and share resources with the host OS, making it possible to run many containers on a single host (POTDAR et al., 2020a). This characteristic is particularly interesting for auto-scaling (even more so in IoT and MEC contexts once they often face resource constraints), as it allows a more cost-effective scaling to meet changing workloads and ensures optimal resource utilization in dynamic computing environments.

Therefore, due to this match of better resource utilization and shorter starting time with the objectives of the desired framework, in this work, we target and use containerization as the virtualization technology and scaling unit in the comparisons.

2.4.3 Monitoring and Metrics

One of the main factors for optimum resource management and auto-scaling is the quality of the monitored metrics (SUKHIJA; BAUTISTA, 2019). Moreover, gaining insights into the condition of a system is of utmost importance in guaranteeing its dependability and equilibrium. Access to information regarding a system's well-being and efficiency enables teams to respond promptly to issues and bolsters security when implementing service modifications (TURNBULL, 2018).

One of the most effective approaches to understanding the system is based on robust monitoring that compiles metrics, visualizes data, and alerts operators whenever anomalies are detected (SUKHIJA; BAUTISTA, 2019). The four key indicators of monitoring (BEYER et al., 2016), are as follows:

❑ **Latency:** Latency quantifies the time required for a system to respond to a request or execute a specific operation. This parameter holds particular significance for systems that demand real-time responsiveness. Elevated latency can indicate performance bottlenecks or limitations in available resources, potentially resulting in an unsatisfactory user experience.

❑ **Traffic:** Monitoring traffic encompasses tracking the volume of data or requests traversing a system. Sudden surges in traffic can exert pressure on system resources and lead to performance degradation. By vigilant surveillance of traffic patterns, teams can preemptively adapt their infrastructure to accommodate increased workloads.

- **Errors:** Errors constitute a fundamental component of system well-being. Monitoring the frequency of errors, whether they manifest as application errors, network glitches, or system malfunctions, is imperative for promptly identifying issues needing attention. Elevated error rates may signify software glitches, network disturbances, or hardware malfunctions demanding immediate resolution.

- **Saturation:** Saturation denotes the degree to which system resources, such as CPU, memory, or disk space, are in use. Monitoring resource saturation aids teams in comprehending when their infrastructure is nearing its capacity thresholds. Timely identification and mitigation of resource saturation can forestall system failures and uphold seamless operations.

2.4.3.1 Prometheus

Prometheus is an open-source monitoring system centered around metrics (TURNBULL, 2018). Developed primarily in Go and distributed under the Apache 2.0 license, Prometheus is characterized by its simple yet powerful data model and a query language that facilitates the analysis of application and infrastructure performance (SUKHIJA; BAUTISTA, 2019). One of Prometheus's distinguishing features is its data model, which uniquely identifies each time series using a name and an unordered set of key-value pairs called labels. This innovation allows for aggregation based on any of these labels, enabling analysis by various attributes such as process, data center, or user-defined service labels (TURNBULL, 2018).

"Prometheus simplifies metric exposure through a straightforward text format, eliminating the need for complex configurations or manual metric definitions. Additionally, Prometheus leverages PromQL, a powerful query language, for creating alerts and dashboards, streamlining the monitoring process (SABHARWAL; PANDEY, 2020). This contrasts with other monitoring systems requiring individual alerts for each machine or application. Furthermore, Prometheus integrates seamlessly with visualization tools like Grafana (GRAFANA, 2023a), enabling comprehensive monitoring of metrics and alerts through user-friendly dashboards.

Prometheus stands out for its efficiency and simplicity. A single Prometheus server can handle millions of data samples per second (TURNBULL, 2018). It's a single, statically linked binary accompanied by a configuration file. All Prometheus components are container-friendly and seamlessly integrate with existing infrastructure, eliminating complexities that can impede configuration management tools. Importantly, Prometheus is designed to be an integral component that integrates with the existing infrastructure rather than being a standalone management platform (CHEN; XIAN; LIU, 2020).

Being a metrics-centric monitoring system, Prometheus' core purpose is to oversee systems' holistic well-being, behavior, and performance rather than becoming entangled in individual events (TURNBULL, 2018). This alignment with auto-scaling principles places it as an essential tool for enabling optimal resource management systems.

2.5 Related Work

This section presents approaches that relate to and underlie our solution. The deployment process in NFV and MEC contexts faces significant challenges, including the resource allocation required by the VNFs or MEC applications.

In Lorido-Botran, Miguel-Alonso e Lozano (2014b) a comprehensive survey about the auto-scaling techniques is presented and authors suggested that the majority of the works can be classified into one or more of the following categories: threshold-based rules, reinforcement learning, control theory, queuing theory, and time series analysis. In Yu, Yang e Fung (2020), the current state of the art on VNF resource allocation is discussed, highlighting the fundamental research challenges and introducing a classification of the main approaches that offer solutions to address them. It categorizes the auto-scaling technologies into four categories: threshold-based rules (reactive auto-scaling), reinforcement learning, control and queuing theories, and time series analysis (predictive autoscaling).

The following sections are structured utilizing these same categories for a better analysis.

2.5.1 Reactive Auto-Scaling

In the reactive approach, previous works can be divided into two main categories based on how threshold levels are defined: statically pre-defined, commonly based on threshold rules, or dynamically updated, usually based on reinforcement learning, control theory, or queue theory.

2.5.1.1 Threshold-based Rules

The approach involves the authors defining performance metrics and predetermined scaling thresholds to trigger a scaling action. The authors commonly define two threshold levels, namely $scale-in_t$ and $scale-out_t$), to determine whether the load reduces below or exceeds the respective limits, thus triggering the scaling process accordingly. In Han et al. (2012) authors presented a lightweight approach based on predefined utilization thresholds of CPU usage to satisfy QoS response time requirements rules, aiming for auto-scaling of cloud applications. In Dutta, Taleb e Ksentini (2016) proposed a scalability mechanism that enables cloud-native 5G systems to dynamically and automatically scale up or down resources of a virtualized environment based on static thresholds. However, such approaches may result in oscillating behavior, mainly when dealing with a spiky workload, where there is a high variance of the metrics, which can present a high response time to respond to a change in the workload, affecting the overall system performance.

2.5.1.2 Reinforcement Learning, Control Theory, and Queue Theory

On the other hand, some other works address the limitations of fixed threshold-based rules, proposing mechanisms that allow the scaling policies to be updated based on dynamic or adaptive thresholds. This context's main enabling technologies and methods are Reinforcement Learning, Control Theory, and Queue Theory.

Reinforcement Learning (RL) is a methodology that enhances a policy concerning a given objective by interacting with an environment, where an agent perceives the state of the environment, takes actions that modify the state of the environment, and receives a reward signal based on those actions (MATSUO et al., 2022). Works using this strategy mainly implement a trial-and-error approach to learn each state's most suitable scaling action. In Arteaga, Risso e Rendon (2017) and Horovitz e Arian (2018), the authors proposed algorithms based on Q-Learning. This model-free off-policy reinforcement learning tries different thresholds and learns their optimal values, enabling dynamic auto-scaling according to the application's behavior. In Lu, Yu e Pan (2022), the authors enable the use of reinforcement learning with the Semi-Markov Decision Process (SMDP), modeling an auto-scaling algorithm for elastic cloud workflow, which enables automatic scaling of cloud workflow services in advance and adapts to changes in traffic earlier. However, despite its benefits, RL algorithms have drawbacks such as extended training duration and suboptimal to poor performance until a satisfactory solution is achieved and learned.

In the auto-scaling context, control theory is used to design controllers that can adjust the resources allocated to a system based on its workload. The controller receives inputs such as the current workload, resource utilization, and performance metrics and produces outputs such as the number of instances to add or remove (DIAO et al., 2005). In Padala et al. (2009), the authors propose an automated control framework for managing multiple virtualized resources in a data center environment. Their approach is based on a control theory framework, which enables the system to automatically adjust the allocation of resources in response to changes in workload demand. The validation results showed particularly good results where workloads can be highly dynamic. In Kalyvianaki, Charalambous e Hand (2014), the authors presented a scheme incorporating the Kalman filter into control theory feedback controllers to dynamically allocate CPU resources to virtual machines. The validation results showed that the proposed controller can self-configure itself with a 4.8% performance penalty in high-intensity workload changes.

On the other hand, the queue theory deals with the analysis of waiting lines and how to optimize them. It can be used to analyze the arrival rate of requests, the service time, and the utilization rate of resources, and with that is possible to determine the expected waiting time for requests and the capacity required to handle them efficiently (YADUVANSHI; SHARMA; MORE, 2019). In Huang et al. (2016), authors used queuing theory to model the web application's behavior in VMs and learn the system's performance under different traffic loads. The proposed approach involved monitoring metrics related to the web application, such as the number of requests per second, response time, and queue length, and using them to estimate the arrival rate and service rate of the system, which are then used to calculate the expected queue length and response time. Based on these calculations, the system can determine whether to add or remove VMs to maintain the desired performance.

In Shahin (2017) authors propose a queuing-theory-based approach that can allocate resources to Software as a Service (SaaS) applications in response to changes in demand, the approach involves modeling the SaaS application as a queueing system, where requests are represented as arriving customers and resources are represented as servers. Based on this model, the authors derive analytical expressions for performance metrics, such as response time and throughput, which can be used to optimize the allocation of resources. The authors then used these expressions to develop a dynamic resource allocation algorithm to adjust the number of servers based on current demand and workload.

However, although these approaches, based on dynamic and adaptive thresholds, perform better than static approaches, they remain reactive solutions with similar weaknesses.

2.5.2 Predictive Auto-Scaling (Time Series Analysis)

The predictive mode utilizes forecasting techniques, such as time series analysis and time series forecasting, to enable systems to learn historical sequences and anticipate future needs, using these predictions to make scalability decisions. This approach is designed to address the limitations of reactive auto-scaling, which can result in delays in scaling up or down and may lead to the over-provisioning of resources (BHARANIDHARAN; JAYALAKSHMI; MAYILVAHANAN, 2022). Predictive auto-scaling utilizes predictive analytics and machine learning techniques to forecast future demand and adjust resources accordingly. This approach analyzes historical usage data to identify patterns and trends in demand. Based on this analysis, machine learning models are trained to forecast future demand and resource needs. Predictive auto-scaling algorithms then use these forecasts to adjust the system's allocated resources proactively (PETROPOULOS et al., 2022). This allows the system to scale up or down before anticipated demand changes, minimizing delays and improving resource utilization.

A thorough examination of various machine learning models for time series forecasting was presented in Bontempi, Taieb e Borgne (2013), which included SVMs, artificial neural networks, and K-Nearest Neighbors (KNN) regressor. There is already a considerable amount of literature in the 5G domain that explores using machine learning models for time series forecasting, which is then used to predict traffic loads and enable VNF autoscaling.

In Alawe et al. (2018), the authors introduced the application of DNN in the VNF requirement prediction field, proposing an LSTM model to learn and predict the future resource needs for a 5G core Access and Mobility Management Function (AMF). This one ensures User Equipment (UE) authentication, authorization, and mobility management in a 5G core network. They trained a model using a collected network dataset, validated it using a discrete event simulator to simulate the traffic, and compared it with the threshold-based solutions. In a similar vein, the authors in Moradi, Ahmadi e Nikbazm (2022), compared multiple machine learning algorithms, including Support Vector Regression (SVR), Decision Tree (DT) and KNN using a real dataset of collected VNFs metrics.

In Kim et al. (2019b), the authors proposed a prediction machine learning model based on CAT-LSTM, that uses a single model to analyze the entire Service Function Chaining (SFC) and predict the future resource demand of a VNF. The analysis of the results showed how SFC data can aid in forecasting the resource consumption patterns of VNFs. Still, this approach presupposes a strong, stable, and constant relationship between VNFs in the SFC. Similarly, in Scalingi et al. (2019), using months of Internet traffic requests, the authors trained and compared the performance of LSTM, Gated Recurrent Unit (GRU) and Arima machine learning algorithms. Moreover, they introduced a prototype architecture based on the docker framework to evaluate the algorithms.

In Zaman, Rahman e Naznin (2019), proposed VNF requirement prediction based on DNN and LSTM networks employing Synthetic Minority Oversampling Technique and Batch Normalization layer to address the issue of an imbalanced dataset. They also examined the effect of varying feature vector sizes and the number of hidden layers on prediction accuracy. Additionally, they introduced and assessed the efficacy of hybrid LSTM models such as CNN-LSTM and Bi-LSTM models. Likewise, in Tao et al. (2021), a prediction model based on LSTM is designed to predict resource demands in NFV-enabled Clouds. Additionally, exploring some cost minimization models and flow routing, the authors proposed a polynomial algorithm based on the Markov chain to find the near-optimal solution for the VNF scaling routing.

2.5.3 Auto-scaling Frameworks

Auto-scaling is an increasingly important feature for modern applications. It is a critical feature for cloud-based and 5G edge applications, enabling them to respond dynamically to changes in demand while ensuring optimal performance and resource utilization (VERMA; BALA, 2021).

In the cloud, auto-scaling is a key feature of commercial cloud platforms such as Amazon Web Services (AWS) (AWS, 2023) and Google Cloud Platform (GCP) (GOOGLE, 2023b) with solutions like Kubernetes (KUBERNETES, 2023). These platforms allow users to define scaling policies based on CPU utilization, network traffic, and queue length metrics. They can scale resources in terms of vertical and horizontal elasticity in response to changes in these metrics. However, a limitation of these commercial solutions is that they are limited to be used with the underlying infrastructure for monitoring and managing associated with themselves.

In the case of the AWS Auto Scaling (SCALING, 2023), this one provides a reactive scaling based on thresholds defined by the user policies and can offer a predictive scaling that will learn patterns if the metrics are exported to CloudWatch (an AWS log solution), and only is available for some metrics like CPU and memory and is restricted run a process every day to use the previous 14 days of data to create an hourly forecast for the next 48 hours (DOCS, 2023). Moreover, Kubernetes Horizontal Pod Autoscaler (HPA) is used by Google Kubernetes Engine (GKE) in GCP to decrease or increase the number of Pods in response to workload's CPU and memory demand (KUBERNETES-DOCS, 2023), and has some open-source projects like Predictive Horizontal Pod Autoscaler (PHPA) (THOMPSON, 2019) that enable predictive auto-scaling. Nevertheless, Kubernetes suffers the same limitations, requiring monitoring and managing tools to be the solutions provided by the framework and limiting the available metrics that can be used in the auto-scaling process.

Similarly, in the 5G edge domain, auto-scaling is becoming increasingly important for applications that run at the network edge, where resource constraints and latency requirements make it challenging to meet changing demand patterns (DUC et al., 2019). One example of a 5G edge application that mostly requires auto-scaling functionality is the MEC platform. MEC allows developers to run applications at the network edge, providing low-latency access to essential resources and services for functionalities like video streaming applications, augmented reality applications, and autonomous vehicle applications (CRUZ; ACHIR; VIANA, 2022). In this approach, most solutions are offered or integrated into network service orchestration tools. The principal is the Open Source MANO (OSM) (OSM, 2020a) is an open-source orchestrator from ETSI that provides VNFs life-cycle management, offering a service architecture based on containers that bring modularity to the solution. However, in the auto-scaling context, OSM only offers a reactive auto-scaling feature based on thresholds that are limited to a few telemetry metrics like CPU, memory, disk usage, and network packet and only when using some specific tools such as VIMs (OSM, 2020b) and does not offer predictive approaches to estimate future workload demand and adjust resources accordingly.

Likewise, in the search and academic work context, a considerable amount of literature proposes different mechanisms and frameworks to enable the auto-scaling of applications. Following, we describe a short survey containing some of these.

In Zafar et al. (2022) proposed a framework based on OpenStack components with Gnocchi and validated the efficacy of using Gnocchi in a VNF auto-scaling scenario, showing better results when compared to the legacy Ceilometer configuration in OpenStack in terms of storage size, memory utilization in processing and management of metrics, and delay in alarm evaluations. The authors in Nicodemus, Boeres e Rebello (2020) introduced VEMoC, a container-based tool incorporating vertical memory elasticity. Their approach monitors the recent memory consumption for each instance. It uses linear equations based on minimum and maximum memory thresholds to predict the future memory need without affecting the containers' performance.

In Vu et al. (2020), the auto-scaling placement and flow migration got more attention. The authors explore the NF state transfer and flow migration in NFV auto-scaling, proposing a framework that handles these functions, adding elasticity to the VNF coordinated control. Moreover, they use OpenStack with a Kubernetes driver to handle the scaling and managing processes. However, the proposed solution does not handle predictive approaches. Similarly, in Al-Dhuraibi et al. (2017) authors proposed ElasticDocker, a tool based on IBM's autonomic computing MAPE-K principles, to enable an autonomous vertical elasticity for Docker containers. Their system scales up and down both CPU and memory container resources according to application workload. However, the proposed system does not support predictive approaches to estimate future workload demand and adjust resources accordingly. Similarly, the paper Bharanidharan, Jayalakshmi e Mayilvahanan (2022) introduces a framework for predictive horizontal scaling of Kubernetes pods using custom metrics and orchestration through Kubernetes and using an approach that leverages LSTM to predict future demand. However, it uses a predefined dataset and does not handle the monitoring and data acquisition process.

2.5.4 State of the art summary

Table 1 highlights a comparative analysis of the diverse solutions identified within the state-of-the-art literature, visually describing the aforementioned works. Several aspects were examined for each proposal. First, the solution's approach can be Threshold-based Rules, Reinforcement Learning, Control Theory, Time Series Analysis, or Policies. Second and third, if the works provide or not auto-scaling reactive and prediction mechanisms. Fourth, the contribution category, as a Research Contribution (work only proposes a concept or idea, without practical implementation), a Research Project (work implements and validates an auto-scaling solution restrained to a specific scenario, not applicable to the market), or an Autoscaling Tool (work provides a full functional auto-scaling tool). Fifth, the main technologies applied in the solution. Sixth, if the work has mechanisms to handle changes in workload patterns, such as new training in predefined intervals or continuous learning. Seventh, for cases where the solution provides a predictive approach based on AI models, this column shows the utilized approach, which can be a single model or, in the case of this work, a multi-model approach. Eighth, the solution's architecture, if it's a monolithic or distributed. And finally, eighth, where the target applications are placed in the network.

Table 1 – State of the art summary.

	Approach	**Reactive Autoscaling**	**Predictive Autoscaling**	**Category**	**Enabling Technologies**	**React to Changes in Workload Pattern**	**AI Model Approach**	**Architecture**	**Applications Network Placing**
Han et al. (2012)	Threshold Rules	✓	○	Research Project	Static thresholds	-	-	Monolithic	Cloud / Core
Dutta, Taleb e Ksentini (2016)	Threshold Rules	✓	○	Research Project	Static thresholds	-	-	Monolithic	Edge
Arteaga, Risso e Rendon (2017)	Reinforcement Learning	✓	○	Research Project	Q-Learning and Gaussian Processes	-	-	Monolithic	Edge
Horovitz e Arian (2018)	Reinforcement Learning	✓	○	Research Project	Q-Learning	-	-	Monolithic	Cloud / Core
Lu, Yu e Pan (2022)	Reinforcement Learning	✓	○	Research Contribution	Semi-Markov Decision Process (SMDP)	-	-	-	Cloud / Core
Padala et al. (2009)	Control Theory	✓	○	Research Project	Control Theory	-	-	Monolithic	-
Kalyvianaki, Charalambous e Hand (2014)	Control Theory	✓	○	Research Contribution	Control Theory and Kalman Filter	-	-	-	-
Huang et al. (2016)	Queue Theory	✓	○	Research Project	Queue Theory	-	-	Monolithic	Cloud / Core
Shahin (2017)	Queue Theory	✓	○	Research Project	Queue Theory	-	-	Monolithic	Cloud / Core
Alawe et al. (2018)	Time Series Analysis	○	✓	Research Contribution	LSTM	-	Single Model	-	Edge
Moradi, Ahmadi e Nikbazm (2022)	Time Series Analysis	○	✓	Research Contribution	SVR, DT and KNN	-	Single Model	Monolithic	Edge
Kim et al. (2019b)	Time Series Analysis	○	✓	Research Contribution	CAT-LSTM	-	Single Model	-	Edge
Scalingi et al. (2019)	Time Series Analysis	○	✓	Research Contribution	GRU	-	Single Model	-	Edge
Zaman, Rahman e Naznin (2019)	Time Series Analysis	○	✓	Research Contribution	LSTM, CNN-LSTM and Bidirectional-LSTM	-	Single Model	-	Edge
Tao et al. (2021)	Time Series Analysis	○	✓	Research Project	LSTM	-	Single Model	Monolithic	Cloud / Core
EC2/ECS - AWS (2023), Scaling (2023)	Threshold Rules and Time Series Analysis	✓	✓	Autoscaling Tool	Static Thresholds and Proprietary AI-Based Algorithm	New training every 24h	Single Model	-	Cloud / Core
GKE - Google (2023b), Kubernetes (2023), Thompson (2019)	Threshold Rules and Time Series Analysis	✓	✓	Autoscaling Tool	Static Thresholds and Linear Regression or Holt-Winters Smoothing	New training every 24h	Single Model	-	Cloud / Core
OSM (2020b)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic / Distributed	Edge
Zafar et al. (2022)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Nicodemus, Boeres e Rebello (2020)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Vu et al. (2020)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Al-Dhuraibi et al. (2017)	Policies	✓	○	Autoscaling Tool	MAPE-K principles	-	-	Monolithic	Edge
Bharanidharan, Jayalakshmi e Mayilvahanan (2022)	Time Series Analysis	○	✓	Research Project	LSTM and GRU	-	Single Model	Monolithic	Cloud

Chapter 3

Resource Auto-Scaling For Everything

Resource management in NFV and MEC environments is a complex and critical aspect that involves efficiently utilizing computing, storage, and networking resources to ensure cost-effectiveness Quality of Service (QoS). Network Functions Virtualization (NFV) implements Virtual Network Function (VNF)s, which are software-based network functions that can be deployed and orchestrated in virtualized environments. In contrast, Multi-Access Edge Computing (MEC) brings computation and storage capabilities closer to end-users at the network edge. By combining VNF and MEC technologies, organizations can benefit from improved flexibility, agility, and reduced latency for various applications and services. However, this integration also brings numerous resource management challenges that must be addressed for successful deployment and operation.

One of the primary challenges in VNF and MEC resource management is handling dynamic workloads (HAIBEH; YAGOUB; JARRAY, 2022a). The demand for services and applications can fluctuate rapidly due to user behavior, time of day, and special events. As a result, resource allocation must be able to adapt in real-time to meet these varying demands. Failure to allocate resources dynamically can lead to performance bottlenecks, service degradation, and dissatisfied users. Moreover, as the number of VNFs and MEC applications grows, the resource management system must scale accordingly to accommodate the increased demand (LLORENS-CARRODEGUAS et al., 2021).

Designing effective resource allocation policies is vital to optimizing resource utilization in VNF and MEC environments. Resource allocation policies determine how resources are distributed among different VNFs and MEC instances based on priority, Service Level Agreements (SLAs), and QoS requirements. Creating policies that prioritize critical network functions and latency-sensitive applications while ensuring fair resource allocation for less demanding workloads is a challenge that requires careful consideration and continuous refinement (QU; CALHEIROS; BUYYA, 2016).

Moreover, energy efficiency is an increasingly critical resource management aspect in VNF and MEC environments (ETEMADI; GHOBAEI-ARANI; SHAHIDINEJAD, 2021). Optimizing resource usage to minimize energy consumption becomes essential with the growing focus on sustainability and reduced carbon footprints. Resource managers must incorporate dynamic power management, workload consolidation, and other energy-efficient strategies into their resource management practices. This is especially more complex in VNF and MEC environments, which, due to the heterogeneous hardware landscape of the edge domain, can introduce further complexity into resource management (SAHNI; VIDYARTHI, 2017). Different edge servers and devices have varying capabilities, making it challenging to optimize resource allocation to meet the unique requirements of different VNFs and MEC applications. Efficient resource management should consider the capabilities and limitations of diverse hardware to maximize performance and cost-effectiveness.

Resource management strategies should also include failure detection and recovery mechanisms. Failures can be inevitable in a distributed and dynamic environment like in NFV and MEC. To handle that, the resource management system should be able to detect failures promptly and automatically recover using strategies such as redistributing workloads to healthy instances to maintain service continuity and minimize downtime (VINAY; KUMAR, 2016). Moreover, achieving cost-effective resource management is a fundamental challenge in NFV and MEC deployments (SINGH et al., 2019). Balancing performance requirements with cost considerations demands sophisticated resource optimization algorithms and continuous resource usage and demand patterns monitoring. Cost optimization is crucial, especially in large-scale VNF and MEC deployments, where efficient resource utilization can significantly impact operational expenses (ETEMADI; GHOBAEI-ARANI; SHAHIDINEJAD, 2021).

3.1 Challenges Addressed

Designing an auto-scaling mechanism that can properly handle the dynamism of both the edge and cloud environments is challenging due to the substantial differences between edge infrastructure and traditional cloud infrastructure (ARABNEJAD et al., 2017). Three intrinsic characteristics distinguish their development: Firstly, traditional cloud computing primarily relies on high-performance servers and networks concentrated in a few centralized locations. On the other hand, edge computing, specifically when dealing with NFV and IoT infrastructures, often face constraints in the available hardware resources. This presents a significant obstacle to employing efficient yet resource-intensive techniques for auto-scaling, such as utilizing machine-learning-based algorithms to predict future resource consumption. Secondly, Multi-Access Edge Computing (MEC) infrastructure comprises a diverse array of heterogeneous resources, each with distinct capacities, reliability, and usability. This heterogeneity introduces challenges in ensuring uniform monitoring and management and achieving optimal resource utilization. Lastly, edge computing resources, especially within IoT, NFV, and MEC environments, exhibit remarkable dynamism. They frequently enter and exit the network based on various factors such as service usage, failures, policies, and maintenance operations. As a result, the auto-scaling system must be adaptable and responsive to cater to these ever-changing conditions.

Aiming these challenges, the study is divided into two main parts. The first part focuses on selecting the most suitable forecasting algorithms for the proposed application. To achieve this, a comprehensive investigation of various time series forecasting algorithms based on ML is conducted. Multiple datasets obtained from VNF monitoring, encompassing diverse trends, seasonality, sizes, and features, are employed to assess the key factors influencing the prediction performance. Moreover, several experiments are carried out to address the dynamic nature of edge environments, involving the implementation and training of each algorithm using different forecasting approaches and variations. These experiments aim to determine the performance of the algorithms in common scenarios encountered in edge computing. The overall results are compared to identify the algorithms integrated into the proposed auto-scaling framework.

In the second part, this work proposes and evaluates the RAFE *(Resource Auto-scaling For Everything)*, a novel framework to address the auto-scaling challenges of both edge and cloud applications. This solution combines reactive and predictive methods to ensure optimal scaling decisions. Moreover, it introduces a core concept that effectively handles hardware resource constraints through the distributed execution of deep-learning model training and resource-consuming tasks. It introduces a diverse set of tools designed to address the dynamic nature of edge and cloud environments. A multi-univariate-models approach is applied to achieve optimum results and keep a low time for training the model. A re-validation mechanism is also introduced to effectively manage the workload pattern changes inherent in network environments. This comprehensive suite of tools aims to optimize the performance and adaptability of auto-scaling in these complex and ever-changing contexts.

The first part, termed "Algorithms Selection", is discussed in detail in the following section and the second part, referred to as the "RAFE Architecture", is presented in sequence.

3.2 Algorithms Selection

This first phase centers on selecting the optimal predictive algorithms tailored to distinct scenarios encountered in time series forecasting of hardware resources within Edge, MEC, and IoT environments. This entails a thorough investigation of diverse machine learning-driven time series forecasting algorithms and the conduction of a series of experiments. These experiments encompass the deployment and training of each algorithm, employing diverse forecasting methods and variations to effectively address the dynamic nature of edge environments.

These processes are presented in this section as follows: (i) the datasets derived from VNFs monitoring, covering a wide range of trends, seasonal patterns, sizes, and attributes, which were chosen to evaluate the key factors that can impact the predictive accuracy; (ii) the pre-processing techniques applied to each dataset and for each evaluated scenario/variation, to prepare the data and improve the results; (iii) the compared ML-based time series forecasting algorithms; (iv) the different variations of time series forecasting approaches evaluated in this work, and why they were chosen due to their impact in the auto-scaling; (v) the hyperparameter optimization processes applied to find the optimal configurations for each algorithm in each scenario; and finally, (vi) the simulation environment built to compare the algorithms and the metrics used to this comparison.

3.2.1 Datasets

Two datasets were used in this work to validate the impact of the data used in the training process on the algorithm's performance. The first comprises a few samples with multiple features (fewer rows but many columns), and the other has many samples with fewer features (many rows but fewer columns). Table 2 highlights their differences and summarizes each dataset's number of samples and features. This discrepancy allowed us to measure the impact in each algorithm of having less or more data and fewer or more features to be processed in the training process.

Table 2 – Datasets composition in the number of samples and features.

	Samples	**Features**
KDN	755	86
BONO	177,038	5

The first dataset, referred to as the *KDN* dataset, was provided in Mestres et al. (2017) and comprises the monitoring results of three distinct VNFs: OVS, Firewall, and Snort. OVS is a versatile virtual switch with multiple implemented network interfaces and protocols. The Firewall acts as a traffic regulator for incoming and outgoing network traffic, enforcing predefined security policies. Additionally, the Snort system functions as an intrusion detection system. The dataset utilized in this study includes 86 input features that capture various traffic characteristics such as port number, source IP, destination IP, and more. The output variable of interest is the measured CPU usage. For this work, we have focused exclusively on the Firewall dataset comprising 755 samples. Figure 11 illustrates the utilized univariate form of this dataset, using the CPU feature.

The second dataset, referred to as the *BONO* dataset, was provided in Bendriss (2018) and is the output of monitoring a Clearwater stack. Clearwater is an open-source environment based on VNF, that utilizes SIP as a call control mechanism for voice and video

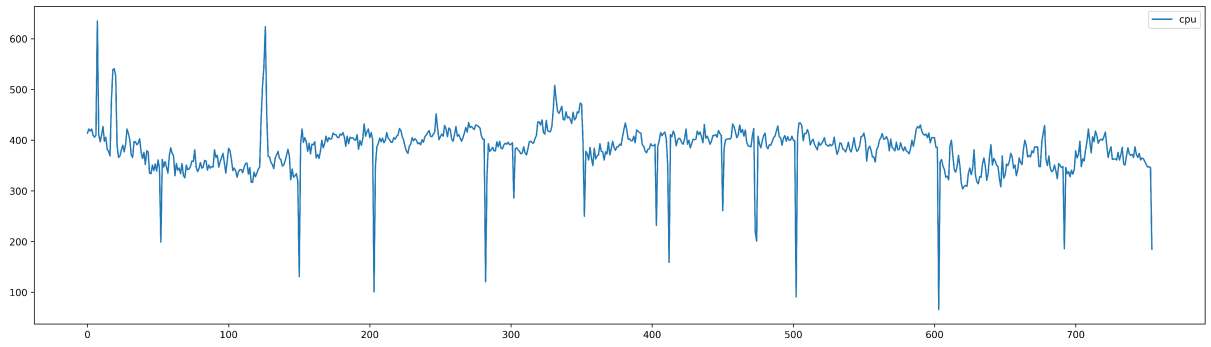


Figure 11 – KDN dataset representation in univariate form.

communications (NETWORKS, 2016). It is structured around the interaction of six VNF components, which include Ellis, Bono, Sprout, Homer, Homestead, and Ralf, each serving a specific purpose. In this work, we have used only the data corresponding to monitoring the Bono VNF, which operates as the Clearwater Edge Proxy, utilizing Sprout (SIP Router) and implementing the Proxy-Call/Session Control Functions (P-CSCF) functions. It serves as the entry point for SIP clients, routing SIP requests to Sprout. The dataset used in this study comprises monitoring results obtained from a simulated stress load involving more than 30,000 distinct SIP profiles connected to the VNF's service chain. It consists of 177,038 samples, capturing resource utilization metrics such as CPU, memory, network (in terms of the number of packets and traffic in bytes), and disk usage. The monitoring was conducted at 30-second intervals. The dataset used in this study is publicly available and can be accessed at Yahia (2020). Figure 12 illustrates the utilized univariate form of this dataset, using the Number of Packets Per Second feature, with a 15-minute resample. Notably, this dataset contains a spike in traffic, presented near point 4000 in the figure, which helps us evaluate the compared algorithms facing this kind of workload pattern.

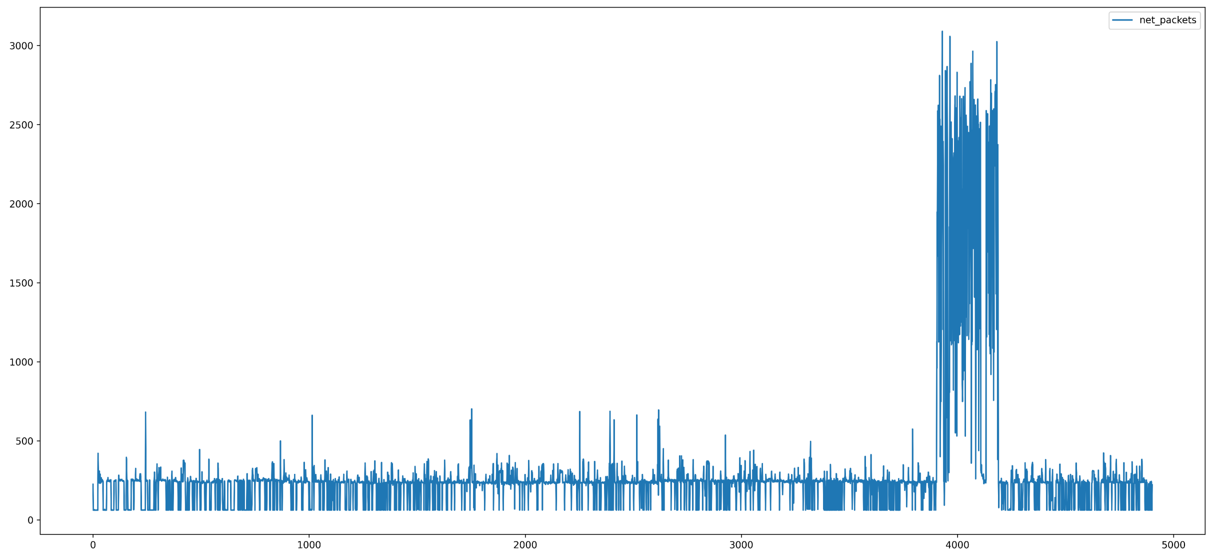


Figure 12 – BONO dataset representation in univariate form and 15-minute resample.

In this work, we analyze and compare multiple ML-based algorithms across various scenarios, considering the number of input features and exploring univariate and multivariate settings. To ensure the datasets align with each scenario's requirements, appropriate adjustments are made. In the univariate variations, where the datasets are composed of a single feature, we specifically use the CPU metric for the KDN dataset (depicted in Fig. 11) and the number of network packets per second for the BONO dataset (depicted in Fig. 12). For the multivariate variations, we utilize all features of each dataset available after the preprocessing.

Dataset Decomposition: To split the datasets, we adopt a 70%:20%:10% rule-of-thumb approach for both datasets. We allocated 70% of the data as training datasets, 20% as validation datasets, and the remaining 10% as test datasets. This decomposition ensures a balanced distribution of data for training, validating, and evaluating the performance of our models.

3.2.2 Data Preprocessing

Data preprocessing plays a crucial role in machine learning, impacting the performance and accuracy of models (MAHARANA; MONDAL; NEMADE, 2022). It involves preparing and transforming raw data into a suitable format that learning algorithms can effectively utilize. By performing tasks such as data cleaning (handling missing values, noise, outliers, and inconsistencies), data integration (merging data from multiple sources), data transformation (normalizing or discretizing data), and data reduction (selecting relevant subsets), data preprocessing helps to enhance data quality, reduce noise, and remove inconsistencies (RAMÍREZ-GALLEGO et al., 2017).

In this work, we initiated the preprocessing phase by implementing a data integration step exclusively for the BONO dataset. This dataset presented a unique challenge as its data was distributed across multiple files, one for each metric. To address this, we consolidated information related to CPU usage, memory utilization, network usage in bytes, and the count of network packets into a unified source. This process was executed by grouping the data based on timestamp values, ensuring consistency across all sources within the BONO dataset. Moreover, a data-cleaning procedure was also applied to the BONO dataset. The missing values were statistically estimated by computing the mean of the nearest values in the sequence, thereby enhancing the overall data quality.

Furthermore, we applied data normalization processing for both datasets, BONO and KDN, using a minimum-maximum re-scaling approach. We employed a min-max scaler, independently translating each feature to a predefined range between -1 and 1. The rescaling process was accomplished by applying the following formula to each feature:

$$X_{\text{scaled}} = \frac{X - \min(X)}{\max(X) - \min(X)} \times (b - a) + a \tag{19}$$

where X represents the original feature values, $\min(X)$ and $\max(X)$ denote the minimum and maximum values of the feature, and a and b represent the desired range (-1 and 1 in this case). This systematic normalization ensures uniformity, improves the comparability of features, and preserves the relationships between data points (MAHARANA; MONDAL; NEMADE, 2022). This normalization was deemed necessary due to the disparate scales and units of features present in both datasets.

In the concluding phase of the preprocessing pipeline, exclusive to the BONO dataset, we opted for a simplification approach to speed up the training processes. This involved re-sampling the dataset at 15-minute intervals, a departure from its original 30-second interval configuration. The re-sampling process entailed selecting the maximum value within each period.

3.2.3 Algorithms

In this work, we selected nine algorithms categorized into three distinct groups: Classical Machine Learning Algorithms, Neural Networks, and Encoder-Decoder Neural Networks. Within machine learning, these algorithms embody supervised learning processes that leverage statistical techniques and methodologies to discern patterns and formulate predictions from data (RAY, 2019). Examples of these machine learning algorithms include, but are not limited to, K-Nearest Neighbor, Support Vector Machines, Decision Trees, and Logistic Regression.

Neural Networks (NN) algorithms represent a class of machine learning models inspired by the structure and function of biological brain networks. Comprising interconnected nodes known as artificial neurons or "units," these networks are organized into layers. Each unit receives inputs, applies weights, conducts computations, and forwards the output to the subsequent layer (SAMEK et al., 2021). Notable examples of NN algorithms include Multilayer Perceptron (MLP), CNN, RNN, and LSTM. Additionally, encoder-decoder neural network algorithms constitute specific variations within the neural network framework. They comprise two integral components: an encoder, responsible for condensing input data into a latent representation, and a decoder, tasked with generating the desired output based on the encoded representation (CHEN et al., 2017). Examples of encoder-decoder neural network architectures include Encoder-Decoder LSTM and Encoder-Decoder CNN-LSTM.

The chosen algorithms can be summarized as:

Table 3 – Compared machine learning algorithms.

Algorithm	**Category**
Random Forest	Classical
Decision Trees (DT)	Classical
Support Vector Regression (SVR)	Classical
Gated Recurrent Unit (GRU)	Neural Network
Long Short-Term Memory (LSTM)	Neural Network
Bi-LSTM	Neural Network
CNN-LSTM	Neural Network
Encoder-Decoder LSTM	Enc-Dec Neural Network
Encoder-Decoder CNN-LSTM	Enc-Dec Neural Network

3.2.4 Time Series Forecasting Modalities/Variations

In the realm of time series forecasting, especially concerning resource prediction, the preparation and organization of data involve several approaches, as highlighted in a study by Lim et al. (LIM; ZOHREN, 2021). The most pertinent options include: (1) Structuring monitored metrics as univariate or multivariate datasets. (2) Executing predictions in either a single-step or multi-step fashion. (3) Approaching the problem by modeling it using either regression or classification methodologies.

Univariate datasets consist of a single variable or feature, distinct from multivariate ones encapsulating multiple variables or features. Univariate analysis is dedicated to comprehending the distribution and patterns of a single variable, while multivariate analysis delves into understanding the relationships between variables and their mutual influences (GARIBO-MORANTE; TELLEZ, 2021).

Moreover, predicting in a single-step manner involves forecasting the value of a time series at a single future time point based on past values. This method, also known as one-step ahead prediction or autoregression (AN; ANH, 2015), is characterized by its focus on immediate future values. In contrast, multi-step prediction entails forecasting the values of a time series over multiple future time points, referred to as k-step ahead prediction. This method is inherently more challenging, requiring the model to capture nuanced patterns and trends in the data while making reasonable assumptions about future behavior (AN; ANH, 2015). Multi-step prediction often demands more sophisticated models and larger datasets for effective training.

Approaching the time series forecasting problem as a supervised learning task involves regression or classification approaches. Regression is employed to predict continuous values, such as prices or probabilities, based on input variables (FAOUZI, 2022). For example, predicting the price of a house based on its size, number of bedrooms, and location. On the other hand, classification is used to predict discrete class labels, such as types of objects or document categories (FAOUZI, 2022). An illustrative instance of classification is predicting whether an email is spam or not based on its content.

Table 4 – Variations

	Features	**Prediction Steps**	**Task Type**
1	univariate	one-step	regression
2	multivariate	one-step	regression
3	univariate	multi-step	regression
4	univariate	one-step	classification

Therefore, to assess the efficacy of each algorithm as a time series forecasting model, it is imperative to implement and compare their performance across different variations. Each algorithm was consequently instantiated in four distinct variations in this study, as detailed in Table 4.

In our initial and foundational variation, we employ a univariate dataset to predict a single step into the future, treating the task as a regression problem. This variation establishes the baseline for our analysis.

Moving to the second variation, we delve into the multivariate perspective, investigating the impact of utilizing multiple features as inputs rather than a singular feature. This consideration proves crucial in scenarios where the forecasted output is contingent upon multiple variables. For instance, incorporating historical CPU and memory consumption data becomes indispensable when predicting the number of VNF instances.

The third variation explores the multi-step approach, evaluating the efficacy of predicting multiple steps rather than just one. This variation can potentially reduce the number of interactions with trained models, easing the burden on monitoring solutions. Accurately forecasting multiple future steps diminishes the necessity for frequent input compositions. This leads to enhanced efficiency and reduced computational overhead—a significant advantage for real-world IoT, MEC or Edge applications.

Lastly, the fourth variation transforms the forecasting problem into a classification rather than a regression task. This exploration assesses the effect of predicting discrete class labels instead of continuous values, simplifying the forecasting solution. This approach proves particularly useful in scenarios with well-defined classes, such as in less complex residential IoT infrastructures, where the range of possible instances to be predicted is limited. By adopting a classification approach, we leverage the inherent categorization of the data, streamlining the forecasting process and facilitating decision-making.

Examining the algorithms across these diverse variations allows us to gauge how each algorithm adapts to the scenarios encountered by IoT and edge devices. This comparative analysis facilitates the assessment of the versatility and suitability of each algorithm in different contexts, providing insights into its strengths and weaknesses. Moreover, it presents an opportunity to identify a potentially universal algorithm that performs well across all scenarios or to determine which algorithm excels in specific contexts.

3.2.5 Hyperparameter Optimization

To find the optimal parameters for a neural network model, an exhaustive exploration of hyperparameters is imperative to achieve accurate predictions for a given input. Our methodology employed techniques such as grid search and babysitting, conducting a systematic search within the hyperparameter space. This approach entailed methodically testing different combinations of hyperparameters to identify the configuration that yields the best results for each model. By leveraging these techniques, we ensured a thorough exploration of the hyperparameter space, striving to determine the most suitable configuration.

In the case of classical machine learning algorithms, distinct hyperparameters may be necessary owing to their reliance on diverse statistical structures. Throughout this process, the parameters considered for the Decision Tree and Random Forest algorithms encompassed the number of estimators, the maximum number of features, the maximum depth, and the bootstrap (if the method for sampling data points is based on replacement). Parameters such as the kernel, gamma, C, and epsilon were considered for the Support Vector Regression algorithm. The ensuing section delineates the specifics of our search space for these algorithms:

RANDOM-FOREST and **DT**:

- ❏ Number of Estimators: 200 to 2000 in intervals of 200.
- ❏ Max Features: sqrt, log2, and auto.
- ❏ Max Depth: 10 to 100 in intervals of 10, or None.
- ❏ Bootstrap: True or False.

SVR:

- ❏ Kernel: rbf, poly and sigmoid.
- ❏ Gamma: 1, 0.1, 0.01 and 0.001.
- ❏ C: 0.1, 1, 10 and 100.
- ❏ Epsilon: 0.05 and 0.1.

For the algorithms based on neural networks, the parameters considered during this process included the number of hidden layers and nodes per layer, the batch size, the learning rate, the regularization parameter (as a dropout probability), the activation function, and the loss function. For the encoder-decoder-based algorithms, aiming to find the optimal structures, as they involve the combination of two networks (an encoder and a decoder), the hidden layers and nodes per layer hyperparameters in the search space were duplicated, allowing independent configuration of these parameters for each network. The following provides an overview of the search space we explored to determine the optimal hyperparameters:

NEURAL NETWORKS:

❏ Hidden layers: 1 to 5.

❏ Nodes per layer: 32, 64, 128, or 256.

❏ Learning rate: 0.01, 0.01, or None.

❏ Batch size: 32, 64, or 128.

❏ Dropout probability: 0.2 or 0.5.

❏ Activation functions: tanh or linear.

❏ Loss function: mean squared error or mean absolute error.

Aiming to determine the optimal values for the combination of input data and forecasting structure, each algorithm's hyperparameter tuning processes were iterated 8 times. This repetition was conducted individually for each prediction modality/variation, and this entire procedure was replicated for each dataset.

3.2.6 Algorithm Comparison

Upon determining the optimal configurations for each algorithm and scenario, we embarked on 10 model training iterations for each of the 72 variations. These variations combined 9 algorithms, 4 forecasting structures, and 2 datasets, resulting in 720 models. This extensive training allowed us to discern the best-performing model for each problem.

In evaluating the models, we considered metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) for regression models and Accuracy and Cross-Entropy for classification models. Additionally, we meticulously calculated and recorded the Time to Train and the Time to Prediction for each iteration. This comprehensive analysis enabled us to gauge the average time required to train a model and the average time for a trained model to generate predictions when given input data.

Implementation of classical machine learning algorithms in this study was executed using Scikit-learn (PEDREGOSA et al., 2011). Meanwhile, neural-network-based algorithms were implemented using TensorFlow (ABADI et al., 2016) and Keras (CHOLLET et al., 2015). Scikit-learn, often called sklearn, is a Python library for machine learning that provides an extensive array of tools and algorithms for tasks ranging from data preprocessing to model evaluation. In contrast, TensorFlow is an open-source library concentrating on neural networks and deep learning. When coupled with Keras, it furnishes a user-friendly and efficient high-level API for defining neural networks, incorporating various pre-defined neural layers like dense, recurrent, and convolutional layers.

All training and comparison processes were conducted within distinct virtual machines hosted and managed on the Google Cloud Platform (GCP) (GOOGLE, 2023b). Each process operated in its isolated instance, equipped with 2 virtual CPUs (vCPUs) and 8GB of memory. This approach ensured dedicated and independent resources for seamless and reliable computations.

3.3 RAFE

In this section, we delve into the second phase of our work, focusing on the RAFE *(Resource Auto-scaling For Everything)* framework. The solution proposes a holistic technique combining responsive auto-scaling, facilitated by threshold-based regulations, and proactive auto-scaling through sophisticated time series analysis. The framework integrates a distributed architecture, self-training mechanisms, auto hyperparameter definition, distributed learning, and a revalidation mechanism to accomplish this.

The ensuing discussion unfolds structured: (i) Problem Statement: A comprehensive exploration of the challenges at hand, offering an overview of the proposed solution and how it seamlessly integrates to address and resolve the auto-scaling problem. (ii) RAFE Architecture: A detailed examination of the RAFE framework's architecture, shedding light on its structural design and the pivotal components that empower its robust auto-scaling capabilities. (iii) Reactive Auto-scaling Approach: Elaboration on the reactive auto-scaling methodology, outlining how the framework promptly responds to fluctuations in demand through threshold-based regulations. (iv) Predictive Auto-scaling Approach: In-depth insight into the predictive auto-scaling approach, elucidating how the framework anticipates future resource requirements using advanced time series analysis. (v) Configurability: A thorough exploration of how RAFE can be configured, providing a practical understanding of the framework's adaptability and customization options.

3.3.1 Problem Statement

Consider a 5G mobile network with multiple base stations, each one equipped with a resource-constrained ME Host that is capable of hosting VNF or MEC applications. Assume that this, as a generic MEC application, is hosted on the ME Host in a virtual machine or a container (in this work, we will only consider applications hosted on containers in reason of the best performance presented by this approach in terms of resource consumption and startup time (POTDAR et al., 2020b)). Suppose that each hosted MEC application has specific resource requirements to perform the demanded computation or to fulfill a required QoS, with a target response time that must be satisfied.

Consequently, due to the dynamic nature of the network environment and its traffic, the number of computing resources assigned to the application must be adaptive according to traffic variations (PEI et al., 2020), increasing/releasing the assigned resources or multiplying/decreasing the number of instances. For the sake of simplicity, in this work, we will exploit only horizontal auto-scaling of containerized edge or cloud applications.

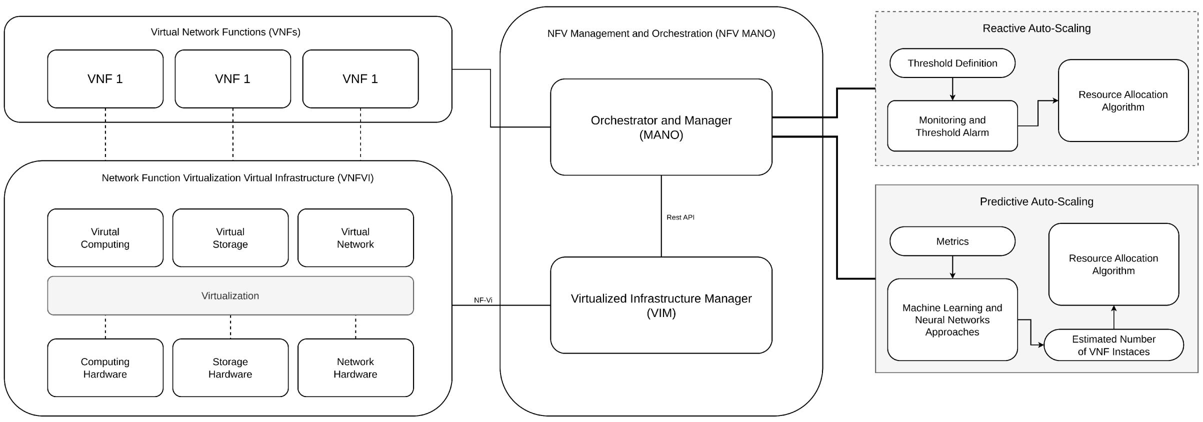

Virtual Network Functions (VNFs)
VNF 1
VNF 1
VNF 1
Network Function Virtualization Virtual Infrastructure (VNFVI)
Virutal Computing
Virtual Storage
Virtual Network
Virtualization
Computing Hardware
Storage Hardware
Network Hardware
NF-Vi
NFV Management and Orchestration (NFV MANO)
Orchestrator and Manager (MANO)
Rest API
Virtualized Infrastructure Manager (VIM)
Reactive Auto-Scaling
Threshold Definition
Monitoring and Threshold Alarm
Resource Allocation Algorithm
Predictive Auto-Scaling
Metrics
Machine Learning and Neural Networks Approaches
Resource Allocation Algorithm
Estimated Number of VNF Instaces

Figure 13 – Auto-Scaling approach integrated with ETSI's NFV reference architecture.

Therefore, in this work, to effectively address optimum auto-scaling performance, we developed a solution that can be represented as the two modules (Reactive and Predictive Auto-Scaling modules) on the right of the Fig. 13, that can interact and is compliant with the ETSI-NFV architecture (KIM et al., 2019a), the latter, is represented as the group in the left of the Fig. 13 by the groups: VNF, VNFVI and NFV MANO. As discussed in previous sections, this model comprises two approaches for auto-scaling, a reactive and a predictive solution, with the union of both outputting a hybrid solution. This dynamic combination ensures a robust and efficient approach to addressing the complexities of achieving optimal auto-scaling performance (QU; CALHEIROS; BUYYA, 2018).

The Reactive Auto-Scaling module, positioned at the top right in Fig. 13, assumes the crucial responsibility of monitoring resource consumption against predefined thresholds. These thresholds, such as the maximum millicore CPU units an application instance can utilize before triggering the need for a new instance, guide the module's actions. When these thresholds are violated, the module invokes the resource allocation algorithm to initiate a scale-out or scale-in operation, thereby increasing or decreasing the number of instances.

The Predictive Auto-Scaling module, depicted in Fig. 13, operates using the collected historical metrics dataset. The module can train an AI-based forecasting model using this dataset as input. Once trained, the algorithm functions as a predictive model, taking the N latest metric values as input and outputting the anticipated number of instances required. This prediction then instructs the resource allocation mechanism to add or remove instances to meet the forecasted number.

To ensure compliance with the ETSI-NFV architecture, RAFE's auto-scaling modules, including their embedded resource allocation mechanism/algorithm, must communicate with the orchestrator, donated MANO in this domain, which is responsible for resource allocation in this NFV architecture (MECHTRI et al., 2017). The auto-scaling modules contact this orchestrator to add or remove VNF instances. Conversely, in a cloud environment, this resource allocation process proposed for RAFE can be seamlessly executed by substituting the MANO and ETSI-NFV architecture components with a container or virtual machine manager, such as Kubernetes Engine (KUBERNETES, 2023) or Elastic Container Service (AWS, 2023).

The subsequent section will delve into RAFE's architecture and implementation details.

3.3.2 Architecture

Each implemented module that composes the RAFE framework is presented in this section, and details of their implementations are discussed. However, due to the complexity, the reactive and the predictive auto-scaling modules are discussed in detail in the following sections, where dedicated focus is given to each. The RAFE's architecture is visually depicted in Fig. 14, offering insight into the interconnected modules. These modules are categorized as follows:

Reactive and Predictive Modules: At the framework's core lie the reactive and predictive auto-scaling modules, serving as the pivotal components responsible for driving scale decisions. The synergy between these two modules gives rise to the hybrid approach. Supporting this core functionality, the additional modules serve as supplements, collectively enhancing the overall performance and functionality of the auto-scaling framework. Due to their complexity, we delve into a detailed examination of their structures and implementations in the subsequent sections.

Northbound Interface (NBI): Defined as an interface facilitating communication with a higher-level component (SHI; ZENG; WU, 2020), the northbound interface in this architecture takes the form of an API constructed using Flask (GRINBERG, 2018). This API is designed to expose services that enable the management and monitoring of the application. Beyond its role in application oversight, the northbound interface serves as a communication channel between the application and the workers. Through strategically defined endpoints, the interface allows workers to request past metric values, transmit trained models, and register the completion of the training process.

Metric Reader: This module serves as a versatile abstraction designed to facilitate communication with diverse monitoring systems, time series databases, or any entity responsible for storing past metric values. Its primary responsibilities include querying

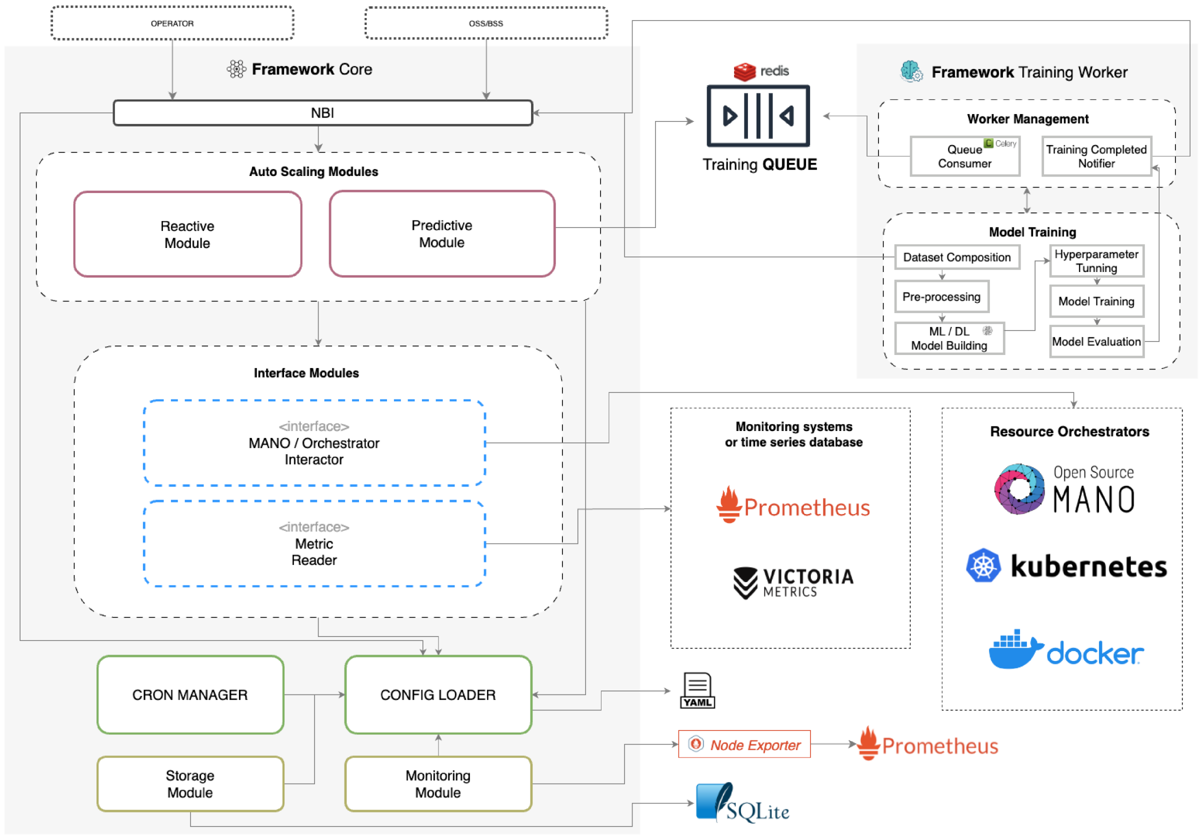


Figure 14 – RAFE architecture.

the configured metrics by requesting these values from the connected source and parsing them into a structure comprehensible to the application. Expanding compatibility with additional monitoring systems can be done simply by implementing a plugin with two distinct methods. The first method receives a metric name, a date range, and a step interval in seconds, returning the metric values within the specified interval and step. On the other hand, the second method receives a metric name, the number of the desired metric registers, and a step interval in seconds, providing the exact number of registers specified as a return. As part of this work, a connection plugin was implemented for Prometheus (RABENSTEIN; VOLZ, 2015), an open-source monitoring system and time series database, demonstrating the adaptability of this module to different monitoring environments.

Orchestrator/MANO Interactor: As defined in the ETSI's NFV reference architecture, MANO serves as the architectural framework responsible for managing and orchestrating VNFs (KIM et al., 2019a). This encompasses the comprehensive management of the entire system, including introducing new network services and VNFs, oversight of the life-cycle of network services, and control over computational, storage, and network resources within the NFV infrastructure. Positioned as a pivotal connection point for the RAFE framework, the MANO Interactor Module assumes responsibility for facilitating communication between the application and the designated resources manager.

Despite its nomenclature, this module extends its utility beyond MANO connections. It offers compatibility with any orchestration tool capable of managing target application instances and executing scale-in and scale-out actions as needed. Providing an abstraction layer, the MANO Interactor Module accommodates diverse MANOs or orchestration mechanisms by implementing plugins. These connections require the definition of four essential methods: one to retrieve the current number of instances for a given VNF or MEC application, and three others that effect changes in the instance count—namely, scale-in (decrementing the instance count by one), scale-out (incrementing the instance count by one), and scale-to (adjusting the instance count to meet a specified number). This module's flexibility is demonstrated through common integration cases, such as OSM (ETSI, 2016a) for edge applications—a project aligned with ETSI's NFV architecture and Kubernetes Engine or Docker for cloud or server applications. Notably, in this study, a Docker and a OSM connector plugin were implemented, facilitating interaction between this module and the container management engine to execute scale actions seamlessly.

Cron Manager: Given the recurrent and scheduled nature of the reactive and predictive modules, a dedicated module was crafted to oversee these processes as scheduled jobs efficiently. Leveraging the APScheduler (APSCHEDULER, 2012) library, this module enables the queuing of Python code execution, offering the flexibility to run tasks as soon as possible, at specified future times, or on a recurring schedule (SOOSAI; JAGDALE, 2014). Notably, the module stands out for its utilization as a scheduler library that supports CRON syntax, an integral feature enhancing flexibility and simplifying the definition of application configurations. Beyond task scheduling, this module takes charge of the persistence and recovery of scheduled arrangements, a critical aspect of the resilience of the RAFE framework. This capability ensures that data and processes can be recovered without loss in case of failures, allowing fast and concise failure recovery. Every scheduled and recurrent process within RAFE, including the periodic evaluations of all auto-scaling mechanisms, is managed and triggered by this module.

Config Loader: The proposed architecture envisions a singular instance of RAFE overseeing auto-scaling operations for multiple applications (which can be VNFs, MEC applications, cloud-hosted apps, etc.) simultaneously. Moreover, it uses a multiple models approach, which will be discussed in the following sections, which train and output multiple models for each application. Given this multiplicity, a robust configuration system must accommodate extensive setups and facilitate the required elasticity. This module allows the configurations to be set statically, at startup-time and during runtime. It achieves this by initially loading the configurations from a YAML syntax or receiving new configuration updates through the NBI. This least, provides a convenient means to specify and modify settings without necessitating a restart. Finally, operating as the custodian of configurations within RAFE, this module oversees a spectrum of settings, including connection strings for integrated monitoring tools, identifications of the target applications, multiple metrics for each application, intervals for scaling evaluations, thresholds, and auxiliary parameters vital for configuring auto-scaling mechanisms. The Config Loader efficiently loads and exposes these configurations to other modules in a structured format designed for easy consumption. A config file example is presented in the following sections.

Storage: In pursuit of resilience and monitorability, RAFE necessitates the persistent storage of the target applications settings, metrics, schedule configurations, and training results. To fulfill this requirement, the Storage Module establishes a layer atop an SQLite Database, a dependable, self-contained, C-language library implementing a small yet fully-featured SQL database engine (SQLITE, 2000). RAFE's architecture also integrates with a Redis database, capable of in-memory storage and disk persistence through point-in-time snapshots or the Append Only File (AOF) persistence option. However, the decision was made to opt for SQLite. This choice was driven by a desire to mitigate data loss in case of failures and to prevent potential performance degradation from operation slowdowns associated with the AOF persistence option (TANG; FAN, 2016). Therefore, tasked with handling read, update, write, and delete operations within the database, the Storage Module extends its role to provide storage as a service to all other modules, ensuring seamless interaction with the persistent data layer.

Monitoring: Dedicated to overseeing the RAFE's architecture, the Monitoring Module provides essential visibility into predicted values, scale decisions, and overall metrics. This functionality is instrumental in managing and assessing the applications within the framework. The Monitoring Module leverages a Prometheus Node Exporter (EXPORTER, 2015) to expose the generated RAFE's metrics. These metrics are then scraped and retained by a Prometheus instance, enabling consumption by analytic tools or facilitation for network operator handling. While not within the scope of this work, it's noteworthy that for visualizing the metrics outputted by this module, we employ Grafana (GRAFANA, 2023a) as a dashboard monitoring tool. Through this integration, we created charts that track RAFE results and performance, enhancing the accessibility and interpretability of the monitored data.

Worker: Time series forecasting poses inherent challenges, particularly with lengthy sequences. Deep Learning (DL) algorithms offer notable advantages in tackling such forecasting tasks, given their ability to automatically handle and learn temporal dependencies and structures (LECUN; BENGIO; HINTON, 2015b). However, it's crucial to acknowledge that the training process for these models can be a resource-intensive task (ZHANG; QI, 2005). The Worker Module was introduced to maintain RAFE's core lightweight and leverage the benefits of distributed and scalable training processes. Operated independently from the core application, the Worker Module adopts an event-driven mechanism structured around Celery (CELERY, 2013), a flexible distributed queue system, and utilizes a Redis database (REDIS, 2009) as a broker for event coordination. This architecture supports the deployment of multiple Worker instances in a distributed manner, all connected to the queues. This enables the training process to be executed in parallel, significantly reducing training time. Upon deployment, these Worker instances actively listen for new events posted on the queue, containing the identification of an application and a metric. Upon receiving an event, the Worker initiates the training process for a new model corresponding to the specified combination. A Worker instance assumes responsibility for executing the resource-intensive steps in the DL-based training process. This involves: (i) requesting past metrics from RAFE's core through the NBI to use as a dataset, (ii) applying pre-processing algorithms to the dataset, (iii) constructing the deep learning model, (iv) executing the hyperparameter tuning mechanism, (v) training the model based on the selected optimal hyperparameters, (vi) evaluating the performance of the model, and (vii) sending back to RAFE's core the created model and the results. This distributed and parallelized approach enhances efficiency and resource utilization in the training phase.

As presented in Fig. 14, the core of RAFE features two pivotal elements: the Reactive Auto-Scaling Module and the Predictive Auto-Scaling Module. Detailed presentations and discussions of these key components follow in the subsequent subsections.

3.3.3 Reactive Auto-Scaling

Auto-scaling approaches can be classified into two main groups: (i) reactive approaches, in which to meet changes in the pattern or resource consumption, the system reacts after the change and, therefore does not anticipate it; and (ii) predictive techniques, where an attempt to forecast future changes in the pattern or resource consumption, performing necessary scaling actions before such changes take place (RODERO-MERINO et al., 2010). RAFE implements both mechanisms.

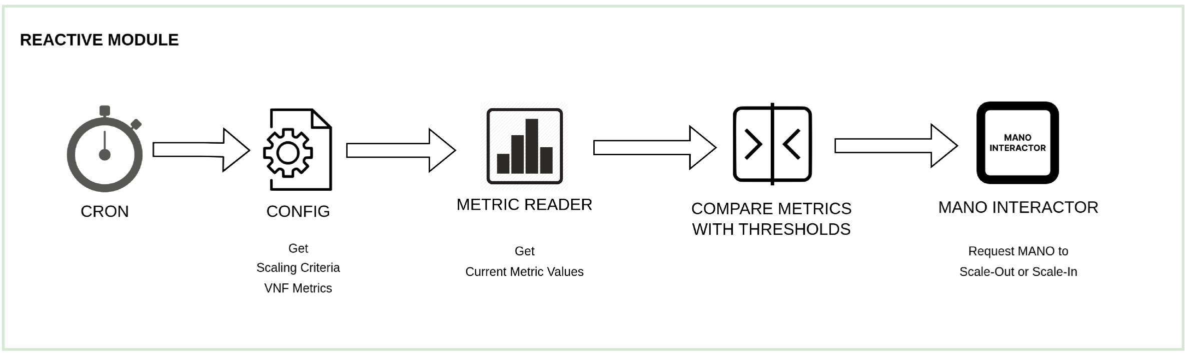


Figure 15 – Reactive Auto-Scaling process through interactions arranged in time sequence.

For this first type of auto-scaling, predictive auto-scaling, in this work, we developed a recurrent procedure represented in Fig. 15. This procedure is repeated respecting a configured *I* interval and, at each interaction, compares the most recent metric values with the specified thresholds to support the scaling decisions. The module in RAFE responsible for this process is the Reactive Auto-Scaling Module. As presented in Fig. 15, it initiates by loading the interval I, configured by the operator, for each configured target application (which can include VNF, MEC or a cloud application). This interval dictates when the reactive auto-scaling process will execute. As an example, a predefined interval with CRON syntax such as **/1 * * * ** triggers the reactive procedure every minute."

At each execution, the reactive auto-scaling process adheres to the steps outlined in Figure 15, encompassing the following key actions: (i) Manage and Trigger Procedure: Utilizing the CRON Manager, the procedure is managed and triggered at regular intervals (I). (ii) Retrieve Metrics and Scaling Criteria: Through the Config Loader Module, all relevant metrics linked to the assessed VNF/MEC/cloud application are fetched. Additionally, the module retrieves the scaling criteria, specifying the configured thresholds for scaling in or out. (iii) Read Current Metric Values: Armed with the knowledge of metrics and their retrieval methods, the Metric Reader Module is employed. This module connects to the designated metric storage, such as Prometheus Storage, to request the latest values for each metric. (iv) Compare Metrics and Make Scaling Decision: Each metric value is compared with the defined scale-in and scale-out thresholds in the scaling criteria. A scaling decision is reached through the decision algorithm, which will be elaborated in detail later in this section. (v) Interact with MANO Module: The MANO Interactor Module is engaged to initiate a scale-out or scale-in action. This action increases or decreases the number of instances by one."

Diving into the threshold validation and the scaling decision processes, they were implemented following the equations (20), (21) and (22).

$$P(t) = \frac{\sum_{i=1}^{X(t)} \gamma(i,t)}{X(t)} \tag{20}$$

$$I(t) = \begin{cases} -1 & \text{if } P(t) < t_{min} \\ 0 & \text{if } t_{min} \leq P(t) \leq t_{max} \\ 1 & \text{if } P(t) > t_{max} \end{cases} \tag{21}$$

As depicted in Equation (20), for each configured metric γ, $X(t)$ represents the number of instances at time t. Here, $\gamma(i,t)$ denotes the value of the metric for instance i at time t. Accordingly, $P(t)$ is defined as the mean value of metric γ at time t for every instance of a single VNF/MEC application. Upon determining the mean value at the evaluated time, the scale decision, or the increment or decrement over the number of instances ($I(t)$), is defined by Equation (21). This equation selects the value based on the configured scaling criteria, where t_{min} is the threshold to scale-in, representing the minimum value of the metric for a single instance, and t_{max} is the threshold to scale-out, representing the maximum value of the metric for a single instance.

$$X(t+1) = \max\left(inst_{min}, \min\left(inst_{max}, X(t) + I(t)\right)\right) \tag{22}$$

Finally, to determine the new number of instances X at the next time $t+1$, $X(t+1)$ can be calculated as specified in Equation (22). This equation represents the sum of the current number of instances $X(t)$ with the increment $(I(t))$. A combination of the min and max functions is then applied to this result, ensuring that the number of instances remains within the bounds of the configured minimum ($\mathrm{inst}_{\mathrm{min}}$) and maximum ($\mathrm{inst}_{\mathrm{max}}$) allowed for the application. These thresholds and limits are configurable by the operator and are retrieved using the Config Loader module.

3.3.4 Predictive Auto-Scaling with Deep Learning

For optimal auto-scaling results, relying solely on a reactive solution may not be sufficient to address swift changes in network traffic and the resulting fluctuations in resource demands. In such cases, the auto-scaling solution might fail to instantiate all the required instances promptly, leading to potential SLA violations, high latency, or even service interruptions (HOFFMANN; TRIVEDI; MALEK, 2007). To mitigate this challenge, a plausible solution involves predicting future resource consumption based on historical data, effectively transforming the problem into a time series forecasting challenge (ALAWE et al., 2018). In alignment with this approach, this work proposes converting the forecasting problem into a supervised learning task using deep learning algorithms.

As depicted in Subramanya e Riggio (2021), time series forecasting becomes incredibly challenging with long sequences, multivariate data (when a time series includes more than one time-dependent variable), and multi-step forecasts. Deep learning approaches offer significant advantages for these cases, including automated learning of temporal dependencies and handling temporal structures such as trends and seasonality.

In this way, constructing a supervised learning model with deep learning and for time series forecasting typically involves the following steps (JIANG; GRADUS; ROSELLINI, 2020):

1. **Collect and Clean the Training Data:** Begin by assembling and organizing a dataset that accurately represents the problem. Thoroughly clean the dataset to eliminate errors or inconsistencies.

2. **Choose a Model:** Navigate through various supervised learning models, including decision trees, support vector machines, and neural networks. The selection process involves identifying the most suitable model that addresses the specific requirements of the problem.

3. **Train the Model:** With a chosen model and a refined dataset, initiate the training process. This involves teaching the model to predict new data by feeding it the training data. Adjust the model's internal parameters to minimize the error between its predictions and the actual values in the training data.

4. **Evaluate the Model:** Following the training phase, assess the model's performance using a separate dataset known as the test set. This evaluation provides insights into how well the model generalizes to new, unseen data.

5. **Fine-Tune the Model:** If the model's performance falls short of expectations, revisit and fine-tune it. Adjust hyperparameters or explore alternative model architectures. This iterative process may involve repeating steps 3 and 4 multiple times.

6. **Deploy the Model:** Once a trained and fine-tuned model is achieved, deploy it for practical use. This deployment may involve integration into a more extensive software system or deployment as a standalone service, allowing the model to predict new data effectively.

Therefore, all these steps must be systematically encompassed to implement a predictive auto-scaling mechanism in RAFE. The subsequent sections will detail the implemented approach adopted in RAFE for these steps.

3.3.4.1 Forecasting Problem Statement

To create a dataset for the training process, we first need to monitor and collect historical data on the target metrics or features. In this process, we define the *step* as the interval between monitoring evaluations, *time-lag* as the number of steps in the past used as input to forecast, and *training-factor* as a multiplier that when multiplied by the time-lag, determines the amount of previous data used for training. For instance, with a 1-minute step, a time lag of 60, and a training factor of 10, metrics are monitored every minute, the module uses the past 60 minutes to predict the next value, and the previous 600 minutes are used for training. In RAFE, these parameters can be individually configured for each monitored application (VNF/MEC/cloud application), and they are loaded using the Config Loader module.

Before delving into the construction of forecasting models in RAFE, it's essential to understand the most common approaches. When dealing with time series forecasting, we have different ways to organize and prepare the data; on the scope of resource prediction, the most suitable options are (LIM; ZOHREN, 2021): (1) deal with the monitored metrics as univariate or multivariate datasets. (2) predict in a single-step or multi-step manner, and (3) model the problem as a supervised learning problem using regression or classification approaches.

Univariate datasets consist of a singular variable or feature, while multivariate datasets encompass multiple variables or features. In a univariate dataset, the emphasis is on scrutinizing and comprehending the distribution and patterns of a solitary variable. This variable can be numerical, such as a person's age, or categorical, such as a person's gender. Techniques for univariate analysis, including histograms and box plots, prove valuable for visualizing and understanding the characteristics of a single variable (GARIBO-MORANTE; TELLEZ, 2021). Conversely, multivariate datasets involve multiple variables, and the focal point is understanding the relationships between these variables and their mutual influence. Multivariate analysis employs techniques like scatter plots and multiple regression analysis to comprehend these relationships and unveil underlying patterns in the data. For instance, a univariate dataset might encompass data on the ages of a group of people. In contrast, a multivariate dataset could include age, gender, and income information within the same group (GARIBO-MORANTE; TELLEZ, 2021). In a multivariate dataset, the analysis could extend to examining how age, gender, and income interact and influence one another.

Single-step prediction entails forecasting the value of a time series at a lone future time point, relying on past values. This method, also known as one-step ahead prediction or autoregression, is exemplified by scenarios like predicting tomorrow's temperature in a daily time series of temperatures. On the other hand, multi-step prediction involves forecasting the values of a time series across numerous future time points, termed as k-step ahead prediction, where k denotes the number of steps into the future for the forecast (AN; ANH, 2015). The complexity of multi-step prediction surpasses that of single-step prediction, as it necessitates the model to discern patterns and trends in the data and make reasonable assumptions about future occurrences (AN; ANH, 2015). Accomplishing multi-step prediction often demands more sophisticated models and an increased volume of training data.

In supervised learning, regression involves predicting a continuous value, such as a price or a probability. A practical regression example is a model forecasting a house's cost based on size, number of bedrooms, location, etc. Regression models undergo training using labeled examples comprising inputs and their corresponding continuous targets (FAOUZI, 2022). On the other hand, classification is the process of predicting a discrete class label, indicating the type of an object or the document category. For instance, a classification model could discern whether an email is spam or not or predict the type of an animal based on its features (FAOUZI, 2022). Classification models are trained using labeled examples of inputs along with their corresponding class labels.

From this perspective, in this work, we modeled the forecasting problem as a **single-step univariate regression** problem for RAFE's predictive module. This decision was significantly influenced by the outcomes of the initial phase of this work, which includes a comprehensive comparison of algorithms across various scenarios. These results are thoroughly presented in Section 4.2.

In broad terms, we opted for a single-step prediction strategy due to its tendency to yield superior results compared to multi-step prediction (AN; ANH, 2015). This strategy not only requires less training data to achieve good performance but also proves more adaptable to changes in network traffic patterns (these facts are supported by our empirical tests and corroborated by studies such as (ZHANG et al., 2021) and (LIJUAN; GUOHUA, 2016)). Additionally, we chose to train one model for each metric employed on predictive auto-scaling. This decision allowed us to maintain all datasets as univariate, simplifying the training process. The advantages include increased speed, improved distribution, reduced resource requirements, and enhanced results. This approach also provides greater flexibility to the framework, enabling it to retrain only the necessary models and eliminating the need for resource-intensive feature selection techniques, such as correlation matrix analysis, Recursive Feature Elimination with Cross-Validation (RFECV), or genetic algorithms (LI et al., 2017; MORADI; AHMADI; NIKBAZM, 2022). Moreover, we chose to treat forecasting as a regression problem. Given its ability to forecast continuous values, this choice mainly allows the framework to harness horizontal and vertical scaling strategies.

Furthermore, adopting the strategy of employing distinct univariate models for each target metric served a dual purpose: not only did it enable us to work with more straightforward pre-processing methods, thanks to the reduced complexity of the datasets, but it also streamlined the overall modeling approach. In this context, RAFE's pre-processing involves a fundamental step—decomposing the generated dataset into training and test subsets. We adhered to a rule-of-thumb decomposition, allocating 85% to the training dataset and 15% to the test dataset. This preparatory step readies the data for model construction and initiates the training phase.

3.3.4.2 Time series forecasting with BI-LSTM

Long Short-Term Memory (LSTM) has been adopted in many time series prediction problems, adding convolutional layers to capture temporal patterns on top of these layers proved immensely supportive in some cases (HOCHREITER; SCHMIDHUBER, 1997). The predictive mechanism developed in RAFE uses a Bidirectional Long Short-Term Memory (Bi-LSTM) architecture to make the predictions. This algorithm was chosen based on results achieved by the first part of this work, which is discussed in Section **??**. Moreover, the main reasons for this choice are highlighted in Section 4.3. We opted for the Bi-LSTM due to its consistently favorable performance and low training times, as demonstrated by our extensive testing across various scenarios. This preference is especially highlighted when dealing with single-step univariate regression tasks, which aligns with the design of RAFE's prediction mechanism.

The Bidirectional Long Short-Term Memory (Bi-LSTM) architecture is built upon the LSTM framework by leveraging not just the past data but also the future insights present in training input as well(HUANG; XU; YU, 2015). This architecture employs two unidirectional LSTMs structures that process a sequence in both forward and reverse directions.

In RAFE, the Bi-LSTM model construction can be described as follows: Initially, the model ingests metric values through the input layer, directing them through one to four dense Bidirectional LSTM hidden layers. These layers are configured with 32, 64, 128, or 256 nodes per layer, where each successive layer contains half the nodes of its predecessor. For instance, if the hyperparameter tuning prescribes two hidden layers with 256 nodes each, the initial layer comprises 256 nodes, while the subsequent layer holds 128 nodes. After traversing these layers, a dropout layer featuring a 0.2 dropout probability is introduced to enhance the model's robustness for regression output. To finalize the model structure, a dense layer incorporating 32 nodes is introduced to compress data dimensionality, with the output layer culminating in a single node dedicated to regression output. Backpropagation is facilitated using an efficient variant of Stochastic Gradient Descent (Stochastic Gradient Descent (SGD)) known as ADAM, employing learning rates set at 0.001, 0.01, 0.1, or 1.0, along with a Mean Squared Error (MSE) loss function. Training persists for a maximum of 200 epochs, during which the loss is computed, and the weight matrix is iteratively updated for subsequent epochs based on the learning rate. The primary objective is to minimize the loss at each epoch and converge toward a local minimum. Once the model is defined and fitted to the training data, it can predict subsequent time steps.

Moreover, in RAFE, each model training procedure incorporates a hyperparameter tuning process to determine the optimal parameters for the Bi-LSTM-based model. The utilized hyperparameter search space for the model composition is discussed in detail in the next section.

3.3.4.3 Model Training

Defining the parameters of a neural network model involves specifying critical aspects such as the number of hidden layers, the nodes within each layer, the activation functions, and other crucial factors. This process entails searching for optimal hyperparameters that yield the most accurate predictions. However, this pursuit can be resource-intensive, particularly in an extensive search space or hyperparameter tuning strategies that evaluate several parameter combinations.

In this work, we tackled these challenges by employing a random search algorithm as our search strategy. This approach aims to conduct a fixed number of iterations, providing results that may not always pinpoint the most accurate parameters but consistently demonstrate similar outcomes. Notably, the random search strategy significantly reduces the time required to identify an optimal set of hyperparameters compared to methods such as grid search (LIASHCHYNSKYI; LIASHCHYNSKYI, 2019). In RAFE, the process of finding optimal hyperparameters for the model is guided by the following search space:

- Hidden layers: 1 to 4.
- Batch size: 32, 64, 128, or 256.
- Nodes per layer: 32, 64, 128, or 256.
- Learning rate: 0.001, 0.01, 0.1, or 1.0.
- Activation function: tanh or linear.
- Dropout probability: 0.2 or 0.4.
- Loss functions: MSE or MAE.
- Maximum of trials: 10.
- Maximum of epochs: 200.
- Early Stopping Patience: 30.

The above hyperparameter search space was chosen to create a versatile solution for predicting metrics based on historical values, focusing on efficient training times. To this end, we had the following motivations: starting with the hidden layers parameter, we opted for a relatively low range as the dataset is consistently univariate, often achieving satisfactory results with a single layer, thereby minimizing training time. Regarding nodes per layer and batch size, we selected a range spanning multiples of 8 between 32 and 256. The batch size was aligned within the same range to ensure each batch contained at least one complete sample. We set a maximum number of epochs at 200 to allow the model to converge. Early stop callbacks were incorporated to halt training if the loss value fails to decrease within a patience range of 30 epochs.

To model the Deep Neural Networks (DNN), we used TensorFlow (ABADI et al., 2016), an open-source machine learning and artificial intelligence library. TensorFlow offers a high-level API known for its cleanliness, uniformity, and efficiency. It allows users to structure the neural network model using the Sequential API or the Subclassing API. These APIs can be seamlessly integrated as standalone and fully configurable modules. TensorFlow also includes a diverse array of predefined neural layers, activation functions, loss functions, and regularization schemes.

The subsequent section will delve into integrating this model into the proposed architecture.

3.3.4.4 Model Revalidation (Suspension and Invalidation)

The network exhibits inherent variability in workload demands (MARSDEN, 2011). This variability poses a significant challenge, particularly in NFV and MEC domains, characterized by diverse resource requirements and network traffic patterns. Managing this dynamic workload landscape presents complex challenges for auto-scaling mechanisms.

The need for efficient resource allocation becomes apparent in these changing workloads, requiring auto-scaling systems to continuously monitor and adapt resource provisioning to align with fluctuating demands (SUBRAMANYA; RIGGIO, 2021).

Predictive auto-scaling mechanisms, significant when leveraging deep learning methodologies, excel in optimizing capacity planning, accommodating complex workloads, improving energy efficiency, and furnishing data-driven insights that empower informed decision-making and elevate the user experience (JOSHI; HADI, 2015). However, despite their prowess in forecasting performance needs, these approaches rely on the data employed during the training phase. These mechanisms learn patterns from the time series data used as input. Consequently, if there is a drastic shift in workload, the model's performance can suffer since it is unacquainted with the new patterns. This knowledge gap can lead to inaccurate estimations, causing performance deterioration during peak utilization and unwarranted resource expenditure during periods of reduced activity. This is also highlighted in our second experiment, presented in Section 5.4.

RAFE introduces a model suspension-invalidation mechanism, also called model revalidation. This process is designed to identify instances where the predictive models lose their accuracy and effectiveness due to changing conditions or unexpected events. RAFE achieves this by monitoring the ongoing performance of these models. At predefined, user-configurable intervals, RAFE captures two crucial metrics: the current number of instances and the number of instances predicted by the model. Therefore, when an expressive difference between these values is detected, the reactive module actively works to adapt to the new workflow pattern, while the predictive module may only be causing disruptions.

This process occurs in two distinct steps. In the initial step, the model is temporarily suspended to enhance overall system performance during short-term fluctuations in workload patterns. Subsequently, in the second step, the model is invalidated, signifying a scenario where the suspension has been invoked multiple times or has been in effect for an extended duration. This dynamic approach enables RAFE to maintain model accuracy and system efficiency in changing conditions. The determination of whether to suspense a model or not adheres to the following expression:

$$\frac{\sum_{i=1}^{n} |o_i - p_i|}{n} \geq t \tag{23}$$

In Equation (23), o_i signifies a vector representing the observed number of instances at time i, while p_i denotes a vector representing the predicted number of instances simultaneously. Here, n represents the total number of data points, and t signifies a user-configurable threshold specified by the operator. Essentially, the model invalidation mechanism computes the Mean Absolute Error (MAE) between the predicted and actual number of instances and subsequently uses a configurable threshold to inform the decision-making process.

Furthermore, validating the suspension status against a user-specified threshold configuration determines the decision to invalidate a model. This threshold can be defined by the number of times a model was inactivated or by the duration of its inactive period.

In RAFE, we suggest discarding inaccurate models rather than subjecting them to continuous learning processes. This choice is based on the RAFE's characteristic of using models for predictive tasks with less complexity regarding the number of layers and nodes per layer to support the multi-model approach, speed up the training process, and have more optimum results. Consequently, they have limitations in terms of incrementally learning without forgetting the old patterns. Furthermore, this approach of invalidating the models leverages the multi-model univariate architecture, which requires less time for training, as confirmed in Section 5.7.

Upon determining the need to invalidate a model, RAFE's model suspense-invalidation mechanism initiates a sequence of actions. It temporarily suspends the predictive mechanism, removes the obsolete model, and triggers a request to generate a new model for the same target application and metric. This approach ensures the system remains adaptable and responsive, promptly replacing outdated models with updated ones to maintain accuracy and efficiency.

3.3.4.5 Integration with the proposed architecture

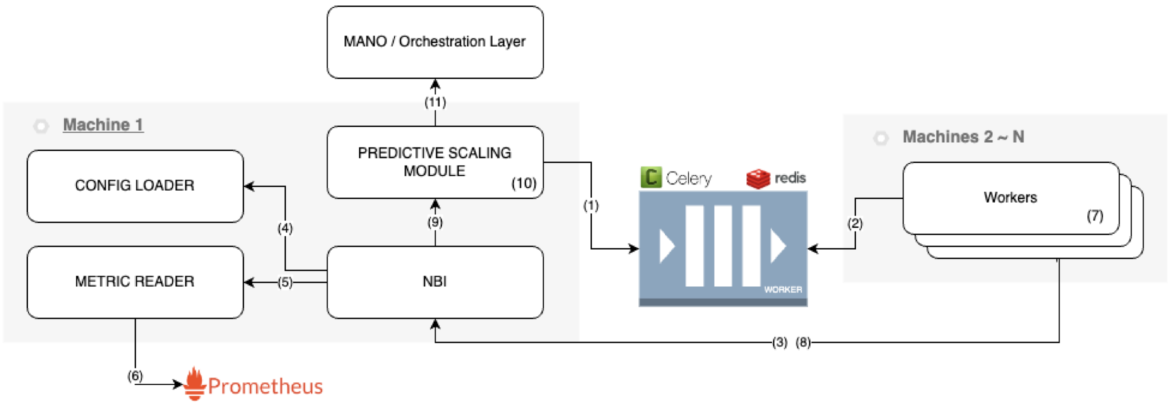


Figure 16 – The AI model training process of the Predictive Auto-Scaling through interactions arranged in time sequence.

Once we have defined how the AI models are constructed, we need to integrate the training process of these models with the proposed RAFE's architecture. This flow is depicted in Fig. 16, and the interactions are numbered by time sequence. As presented, the flow of a new training process is triggered by the Predictive Auto-Scaling Module when it identifies a target metric that doesn't have an associated trained model for prediction. Therefore, as shown by the (1) interaction in Fig. 16, the predictive module posts a training request event (an event specifying the target application and the desired metric identification) on a Celery queue. A Redis database manages this queue, which queues the requests and implements a publisher–subscriber architecture.

Therefore, this event is subsequently consumed by a subscribed worker (2). The worker executes a Celery Job, which involves requesting the necessary data, applying pre-processing functions, and constructing and training a new model. These workers have been designed to be versatile, allowing distribution and hosting in diverse environments. This design choice ensures that the resource-intensive process of training a deep learning model is kept separate from the auto-scaling orchestrator. For example, it permits deploying the auto-scaling orchestrator on the edge while the workers are deployed on cloud servers. However, deploying the RAFE core and one or more workers on the same host may incur performance penalties due to the substantial resources required for machine learning training processes and potential demands for multiple models. In line with the ETSI's NFV reference architecture, we recommend hosting the RAFE core with the MANO and deploying the workers on other hosts. Additionally, a brief communication time can significantly enhance scaling results.

The worker, tasked with managing the training process, initiates the retrieval of historical data to serve as a dataset. It accomplishes this by sending an HTTP request (3) to the exposed RAFE core NBI, specifying the VNF/MEC application and the metric name. Upon receiving this request, the core's service verifies the configured parameters for the specified VNF/MEC application and metric. It retrieves crucial information such as step, time lag, training factor, and details on where and how the metric is stored (4). Subsequently, the core requests past metric values from the metric storage, represented by a Prometheus time-series database (5, 6). While Prometheus is employed as the primary data source in this work, it's noteworthy that the architecture allows for the seamless implementation of alternative data sources. As a final step, the core furnishes the worker with the retrieved past metric values and the corresponding configuration parameters, facilitating the commencement of the training process.

With the acquired data, in step (7), the worker commences the pre-processing and training phase. It begins by decomposing a dataset into training and test sets, adhering to an 85%/15% rule-of-thumb split. Subsequently, the worker constructs an LSTM neural network and iteratively trains it multiple times using various hyperparameters, employing a random search strategy for hyperparameter tuning. The evaluation of generated models involves comparing the mean squared error of the loss function during predictions on the validation dataset. Based on this assessment, the worker identifies the best model, emphasizing its ability to generalize effectively to unseen data. Once the training process is complete and the optimal model is defined, the worker finalizes the procedure by transmitting the model definition file in Hierarchical Data Format (HDF) format (8) through the core's NBI. This file encapsulates the model's weights and network composition. Subsequently, the core forwards the model to the predictive auto-scaling module

(9), integrating it to forecast future values.

As step (10) outlined, the Predictive Auto-Scaling Module can start its operation once it has at least one valid and trained model for the specified metrics. This process can be defined as:

(i) At each T time (which T can be configured through the Config Module as a CRON), the CRON Module triggers a single-step auto-scaling prediction event.

(ii) For each configured metric, the Predictive Auto-Scaling Module requests the Metric Reader Module for the most recent stored metrics and gets these values.

(iii) Loads all the stored AI models through the Model Interactor Module. Using the metrics from the second step, it uses them as input for the models and receives as output the predicted amount of that metric for one step in the future.

(iv) With these predictions, the Predictive Auto-Scaling Module requests the configured maximum value per instance of each metric through the Config Module.

(v) The module then applies the expression defined in Equation 24 (which will be detailed in sequence) and gets. As a result, the number of replicas should be instantiated for the target VNF / MEC application.

(vi) Finally, once the value, the Predictive Module communicates with the configured MANO or Orchestration Layer through the MANO Interactor Module, requesting the scale-in or scale-out of instances to match the exact number of predicted instances.

Equation 24 defines the algorithm used to determine the predicted number of instances based on multiple metrics, models, and predictions. In this equation, D represents the set of configured metrics for the predictive mechanism, $P(x)$ is the predicted value for the metric x, and $M(x)$ is the configured maximum allowed value of the metric x per instance.

$$\max_{x \epsilon D} \left\lceil \frac{P(x)}{M(x)} \right\rceil \tag{24}$$

Therefore, the predicted number of instances is calculated by rounding up the division of the predicted value by the maximum allowed value per instance for each configured metric and then selecting the maximum value among these results. This expression ensures that the minimum number of instances required to handle the predicted workload is ready to manage this demand without compromising the QoS and efficiently utilizing resources.

3.3.5 Configurability

A reusable framework must be broadly configurable and fine-tuned to meet multiple demands (ATOUI et al., 2020). To this end, as previously mentioned, RAFE can be configured using the services exposed in the NBI or a YAML file. In this section, we showcase a real-world configuration for RAFE, highlighting the application's versatility and capacity to satisfy specific demands of different applications, irrespective of its type or location in the network. This example also offers valuable insights into the practical implementation of our proposed solution and its potential applicability within the network landscape.

RAFE's configuration can be done through the use of the following YAML structure:

Listing 1 – YAML structure used for RAFE configuration

```yaml
- app-id: app

  metrics:
    - name: cpu_usage_percentage
      expression: container_cpu_usage_seconds_total{name=~"app.*"}[15s]
      source: prometheus_monitor

  scaling-policy:
    cooldown-time: 120
    max-instance-count: 20
    min-instance-count: 1

  reactive-scaling:
    enabled: true
    cron: "*/10 * * * * *"
    scaling-criteria:
      - metric: cpu_usage_percentage
        scale-in-relational-operation: LT
        scale-in-threshold: 0.15
        scale-out-relational-operation: GT
        scale-out-threshold: 0.4

  predictive-scaling:
    enabled: false
    cron: "*/1 * * * *"
    config:
      scaling-type: single-step
```

```
          future-steps: 10
          time-lag: 72
          training-factor: 14 # multiplied by time-lag to get the amount
                              # of training data || default: 20
          step: 1m
        scaling-criteria:
          - metric: cpu_usage_percentage
            max-value-per-instance: 0.4
        revalidation:
          enabled: true
          cron: "*/5 * * * *"
          monitoring-interval: "*/10 * * * * *"
          threshold: 1.8 # MAE threshold
          invalidate: 5 # invalidate after X suspensions
          steps-limit: 30
```

The document presented in Listing 1 provides an example of configuring RAFE's hybrid auto-scaling for a single application using a single metric (CPU usage in this case). RAFE expects a YAML structure with a list of applications at the root level, where each item defines an application to be monitored and managed for auto-scaling services. Each application should have an associated unique identifier.

For each application, the operator must specify the scaling policy, which includes parameters such as cooldown time and soft minimum and maximum limits for the number of instances. Additionally, there is the option to configure the reactive and predictive modules and enable the hybrid mechanism by activating both of these modules. Each auto-scaling method (reactive and predictive) has specific configurations, as outlined earlier in this chapter in their respective sections. Notably, the revalidation mechanism is customizable within the predictive scaling section. For more flexibility and ease of configuration, all the schedules and intervals can be specified using the cron syntax (RAFE's Cron Module manages all cron-related tasks).

This document section illustrated RAFE's adaptability and flexibility, designed to be seamlessly deployable across diverse network environments.

Chapter 4 Algorithm Selection Results

The contributions of this research can be delineated into two parts. The first part concentrates on the selection of the most appropriate forecasting algorithms to be used in RAFE's prediction mechanism. To accomplish this, an exhaustive exploration of multiple machine learning-based time series forecasting algorithms is conducted. Diverse datasets, sourced from VNF monitoring, encompassing a wide variation of trends, seasonality patterns, dataset sizes, and features, are utilized to evaluate how each algorithm handles each factor. Furthermore, to simulate the dynamic characteristics of edge environments, a series of experiments are executed involving the deployment and training of each algorithm over various forecasting methodologies and configurations. These experiments evaluate the algorithm performance in typical edge and cloud computing scenarios. At last, the collective outcomes are compared to identify the algorithms best suited for integration into RAFE, the proposed auto-scaling solution.

In the following sections, first, we showcase the outcomes from our algorithm comparison and delve into a comprehensive analysis of these findings, analyzing several forecasting categories: univariate/multivariate, one-step/multi-step, and regression/classification. This exploration underscores key takeaways that will serve as a base for our subsequent selection of the most suitable algorithm. These results are pivotal in guiding our choice of the algorithm and approach to be integrated into the RAFE. Moreover, we introduce a Docker-based experimental testbed environment, to evaluate RAFE over multiple scenarios.

This section presents the outcomes from the procedures outlined in Section 3.2. We compare multiple machine learning algorithms to define the most appropriate ones for time series forecasting across various autoscaling strategies. Moreover, this process was undertaken to establish which algorithms would be applied in the autoscaling mechanism proposed by this work. To commence, we present the outcomes of hyperparameter tuning, briefly presenting the optimal configurations chosen for each algorithm within each scenario. Subsequently, we comprehensively appraise each algorithm's performance across each simulated scenario, highlighting comparisons and contrasts. Finally, a comprehensive overview of the algorithmic evaluation is provided, encapsulating the holistic insights garnered from this comparative analysis.

4.1 Hyperparameter Tuning Results

This outcome and the code and algorithms utilized are hosted and openly available in Vinhal (2023). This repository encompasses the code of each algorithm with the chosen hyperparameters and thoroughly compares the architectures employed in both machine-learning algorithms and neural networks.

4.2 Algorithms Comparison Results

In this section, we use the outcomes of the hyperparameter tuning process to define the structure of each algorithm. Many of these algorithms, particularly the neural networks, are influenced by the initial random weights assigned during model generation. To ensure the robustness and quality of our results, we executed the model creation and training process 10 times for each algorithm in each comparison. The mean values obtained from these 10 iterations are then used for comparisons.

To conduct the assessment, using the performed 10 runs for each case, we recorded the resulting Mean Squared Error (MSE) and Mean Absolute Error (MAE) values for each item. The performance values are presented in a Confidence Interval (CI) of 95% likelihood, indicating the range of classification errors expected from the models.

Each subsequent subsection offers a detailed comparison of the results for each algorithm across each forecasting variation (variations are outlined in Section 3.2.4).

4.2.1 Base Analysis (Variation I)

Training each algorithm over the first variation, Table 5 depicts each algorithm's performance. This variation, called base variation, involves organizing the data and algorithms within a univariate single-step regression forecasting architecture. The comparison is conducted by performing 10 simulation runs for each case, and the resulting MSE and MAE values are recorded for each item. The performance values are presented concerning a CI of 95% likelihood. Furthermore, we calculate and present the mean Time to Train and the mean Time to Predict. The former signifies the average time required to build and train the model. At the same time, the latter represents the average time needed to obtain the model output once provided with input values.

For the KDN dataset, which has a smaller number of rows but more features, the LSTM model demonstrated the highest performance, achieving 0.01275 MSE and 0.06261 MAE (its results were plotted in Fig. 17). The GRU model comes in second place with

Table 5 – Performance results in variation I (univariate - one-step - regression). Ordered by best to worst results.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
LSTM	0.01275 (±0.00024)	0.06261 (±0.00279)	1003.44 s
GRU	0.01285 (±0.00021)	0.05966 (±0.00073)	252.32 s
DT	0.01302 (±1e-18)	0.06293 (±1e-17)	0.01 s
BI-LSTM	0.01327 (±0.00034)	0.06109 (±0.00319)	204.41 s
ENC-DEC-LSTM	0.01347 (±0.00049)	0.06183 (±0.00167)	195.89 s
CNN-LSTM	0.01366 (±0.00034)	0.06426 (±0.00262)	353.57 s
RANDOM-FOREST	0.0141 (±0.0)	0.06563 (±1e-17)	3.91 s
SVR	0.01415 (±0.0)	0.06453 (±0.0)	0.05 s
ENC-DEC-CNN-LSTM	0.02463 (±0.00217)	0.10446 (±0.00725)	245.89 s

(a) For KDN dataset.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
BI-LSTM	0.00283 (±4e-05)	0.03928 (±0.00108)	1249.5 s
LSTM	0.00284 (±7e-05)	0.04266 (±0.00245)	1618.67 s
RANDOM-FOREST	0.00298 (±3e-19)	0.03973 (±0.0)	117.34 s
SVR	0.00304 (±3e-19)	0.04547 (±5e-18)	1.08 s
ENC-DEC-LSTM	0.00306 (±0.00018)	0.04389 (±0.00139)	2059.96 s
GRU	0.00322 (±0.00024)	0.04555 (±0.00387)	6023.83 s
ENC-DEC-CNN-LSTM	0.00359 (±0.0004)	0.04471 (±0.00191)	954.01 s
CNN-LSTM	0.00361 (±0.0006)	0.0453 (±0.00451)	3237.01 s
DT	0.01477 (±1e-18)	0.05908 (±7e-17)	0.22 s

(b) For BONO dataset.

comparable results, obtaining a 0.01285 MSE and 0.05966 MAE. The DT algorithm ranks third, showing a 0.01302 MSE and 0.06293 MAE. On the other hand, the ENC-DEC-CNN-LSTM algorithm performs the worst among the evaluated models, with an 0.02463 MSE and 0.10446 MAE, followed by the SVR with an 0.01415 MSE and 0.06453 MAE.

In the case of the BONO dataset, which has a larger number of rows but fewer features, the BI-LSTM model achieved the highest performance, with 0.00283 MSE and 0.03928 MAE (its results were plotted in the second chart in Fig. 18). Following closely, the LSTM model secures the second position with 0.00284 MSE and 0.04266 MAE. The RANDOM-FOREST algorithm ranks third, exhibiting a 0.00298 MSE and 0.03973 MAE. On the other hand, among the evaluated models, the DT algorithm performs the poorest, yielding a 0.01477 MSE and 0.05908 MAE, trailed by the CNN-LSTM model with a 0.00361 MSE and 0.01477 MAE.

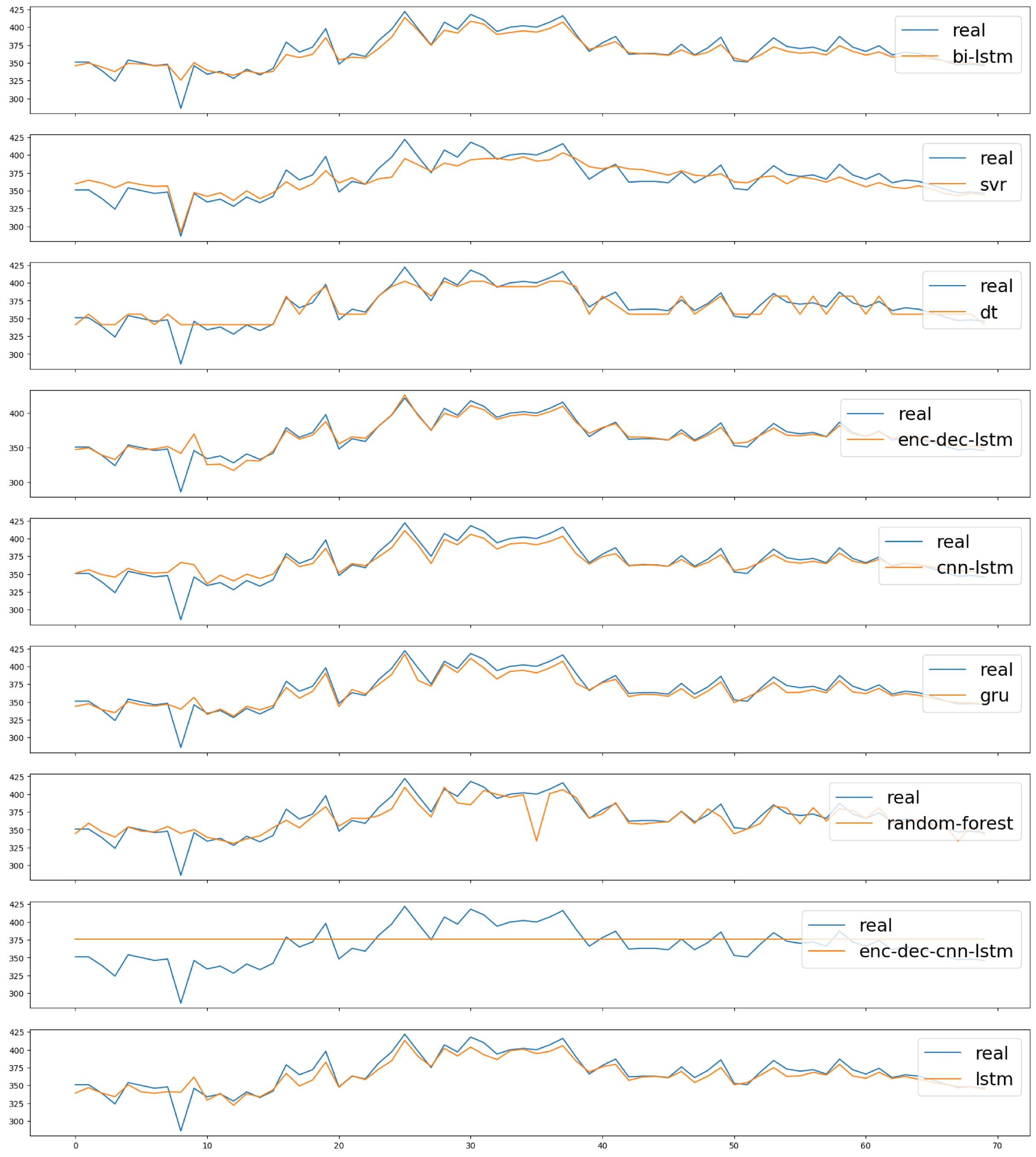


Figure 17 – Predicted values compared to the original/expected values, using the test portion of the KDN dataset and variation I.

It is worth noting that, in both datasets, the neural network models tended to outperform the classical and encoder-decoder algorithms. This observation can be attributed to the fact that in this particular variation involving a univariate structure and less complex inputs, the encoder-decoder algorithms, comprising two neural networks, tend to become overly complex to the univariate architecture, leading to underfitting scenarios. On the other hand, the classical algorithms demonstrated favorable performance in the KDN dataset, which had a limited amount of data. Still, they struggled to handle the BONO dataset effectively due to its more complex data.

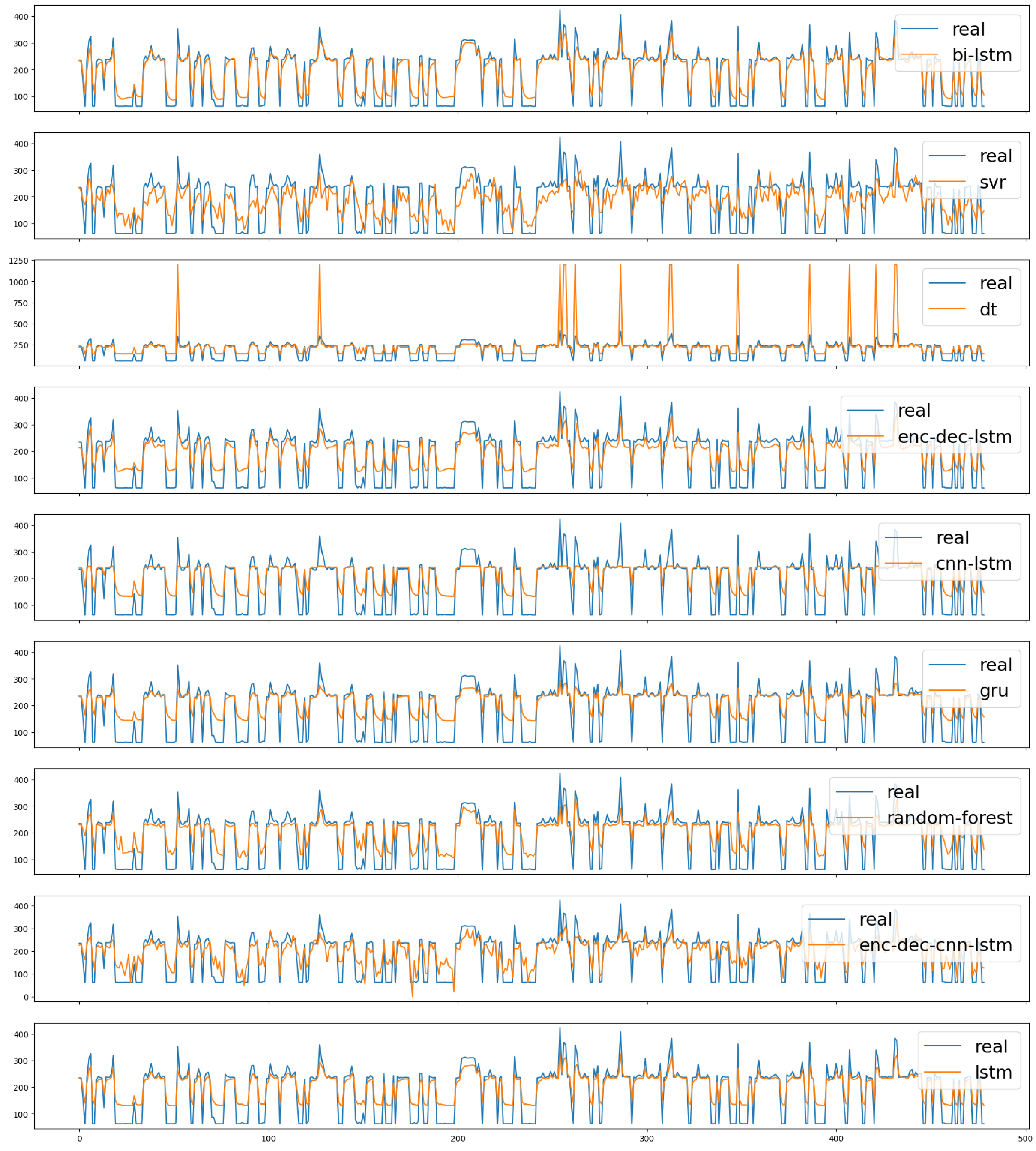


Figure 18 – Predicted values compared to the original/expected values, using the test portion of the BONO dataset and variation I.

Moreover, in a comprehensive analysis, the LSTM and BI-LSTM algorithms demonstrate good performance across both datasets. The predicted values of these algorithms for the test portion of the datasets are shown in Fig. 17 and Fig. 18, showcasing their strong ability to forecast based on unseen data. Although both algorithms yield similar results regarding loss and graphical representation, the BI-LSTM algorithm requires less training time in both comparisons, further highlighting its efficiency.

Finally, this particular variation stood out as the most notable in the overall analysis, surpassing the other variations, which will be discussed in subsequent sub-sections. This highlights the potential of this base variation (univariate, one-step, regression) mostly in an IoT/Edge scenario, which requests solutions that can meet the high variability scenarios requested by the network.

4.2.2 Multivariate Analysis (Variation II)

Table 6 compares algorithm performance in variation II. The table follows a similar structure as Table 5, and the simulation tests were also conducted in accordance. In contrast to the base variation, this particular variation employs a multivariate architecture, utilizing the entire dataset with all features as input during both the training and prediction stages. This deviation departs from the other scenarios, where only a single feature was utilized.

Table 6 – Performance results in variation II (multivariate - one-step - regression). Ordered by best to worst results.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
RANDOM-FOREST	0.0158 (±0.0)	0.07282 (±1e-17)	2267.64 s
BI-LSTM	0.01634 (±0.00104)	0.08724 (±0.00546)	1691.22 s
LSTM	0.01681 (±0.00084)	0.08019 (±0.0038)	804.68 s
SVR	0.01814 (±0.0)	0.08479 (±0.0)	5.58 s
DT	0.01817 (±0.00024)	0.08207 (±0.00108)	0.01 s
GRU	0.01844 (±0.00112)	0.08365 (±0.00352)	5435.93 s
CNN-LSTM	0.01859 (±0.00029)	0.08534 (±0.00217)	4521.76 s
ENC-DEC-LSTM	0.01961 (±0.00265)	0.08835 (±0.0089)	4672.04 s
ENC-DEC-CNN-LSTM	0.0217 (±0.00272)	0.09689 (±0.01154)	2643.98 s

(a) For KDN dataset.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
BI-LSTM	0.00288 (±0.0002)	0.04129 (±0.00275)	5272.65 s
ENC-DEC-LSTM	0.00335 (±0.00077)	0.03984 (±0.00502)	10132.7 s
CNN-LSTM	0.00366 (±0.00071)	0.04175 (±0.00248)	16819.85 s
GRU	0.00386 (±0.00143)	0.0417 (±0.00437)	6955.56 s
ENC-DEC-CNN-LSTM	0.00441 (±0.00023)	0.04578 (±0.00139)	7021.79 s
LSTM	0.00455 (±0.00188)	0.0467 (±0.00762)	6445.8 s
SVR	0.00461 (±0.0)	0.0529 (±0.0)	7.95 s
RANDOM-FOREST	0.00796 (±0.0)	0.06511 (±0.0)	10327.05 s
DT	0.01705 (±3e-17)	0.05872 (±1e-16)	0.81 s

(b) For BONO dataset.

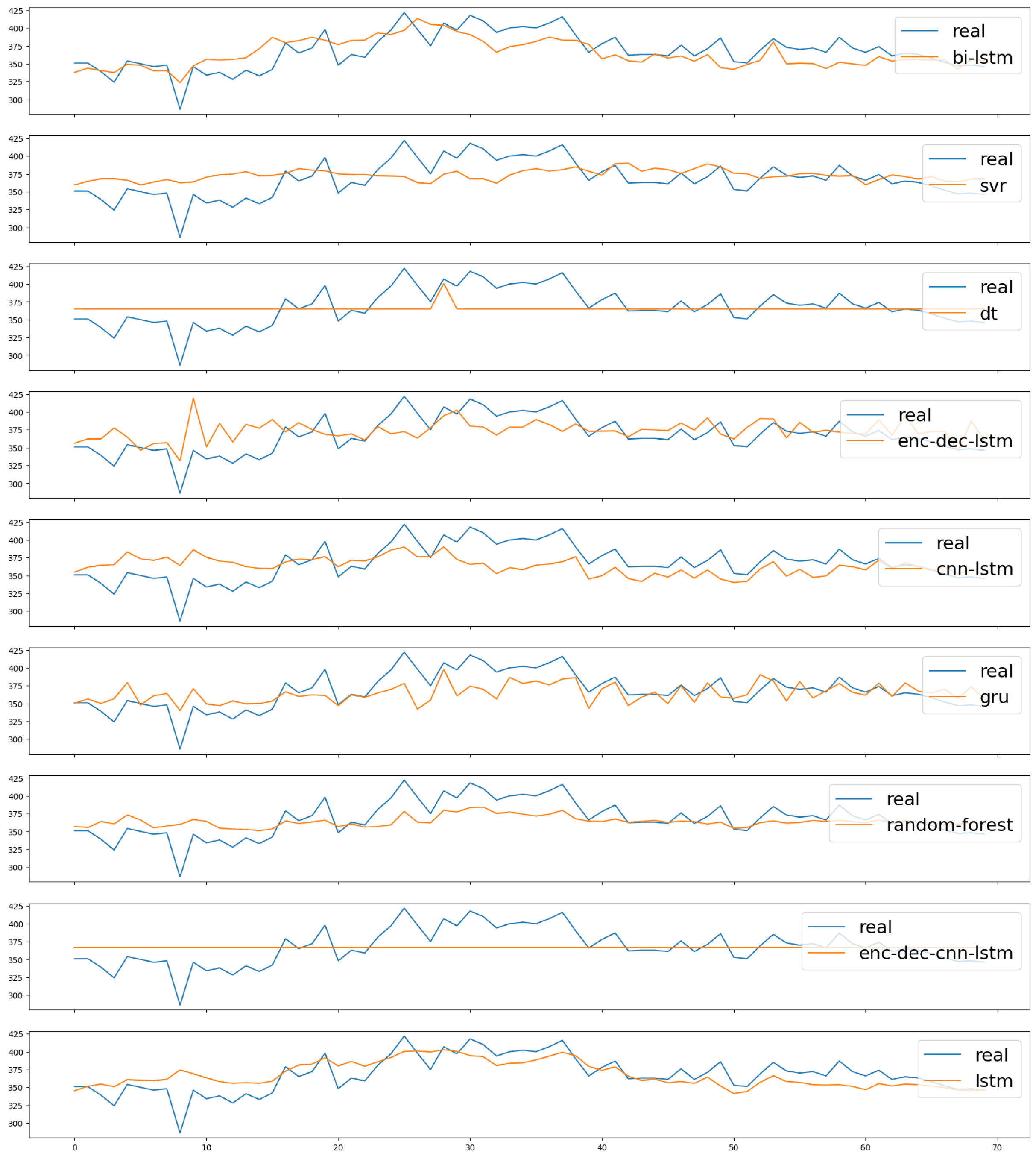


Figure 19 – Predicted values compared to the original/expected values, using the test portion of the KDN dataset and variation II.

In the case of the KDN dataset, in this variation, the RANDOM-FOREST algorithm demonstrated superior performance, achieving an 0.0158 MSE and 0.07282 MAE (its results were plotted in the first chart in Fig. 19). The BI-LSTM model closely followed with comparable results, obtaining a 0.01634 MSE and 0.08724 MAE. The LSTM algorithm ranked third, showing a 0.01681 MSE and 0.08019 MAE. Conversely, the ENC-DEC-CNN-LSTM algorithm performed the worst among the evaluated models, with a 0.0217 MSE and 0.09689 MAE, followed by the ENC-DEC-CNN-LSTM with a 0.01961 MSE and 0.08835 MAE.

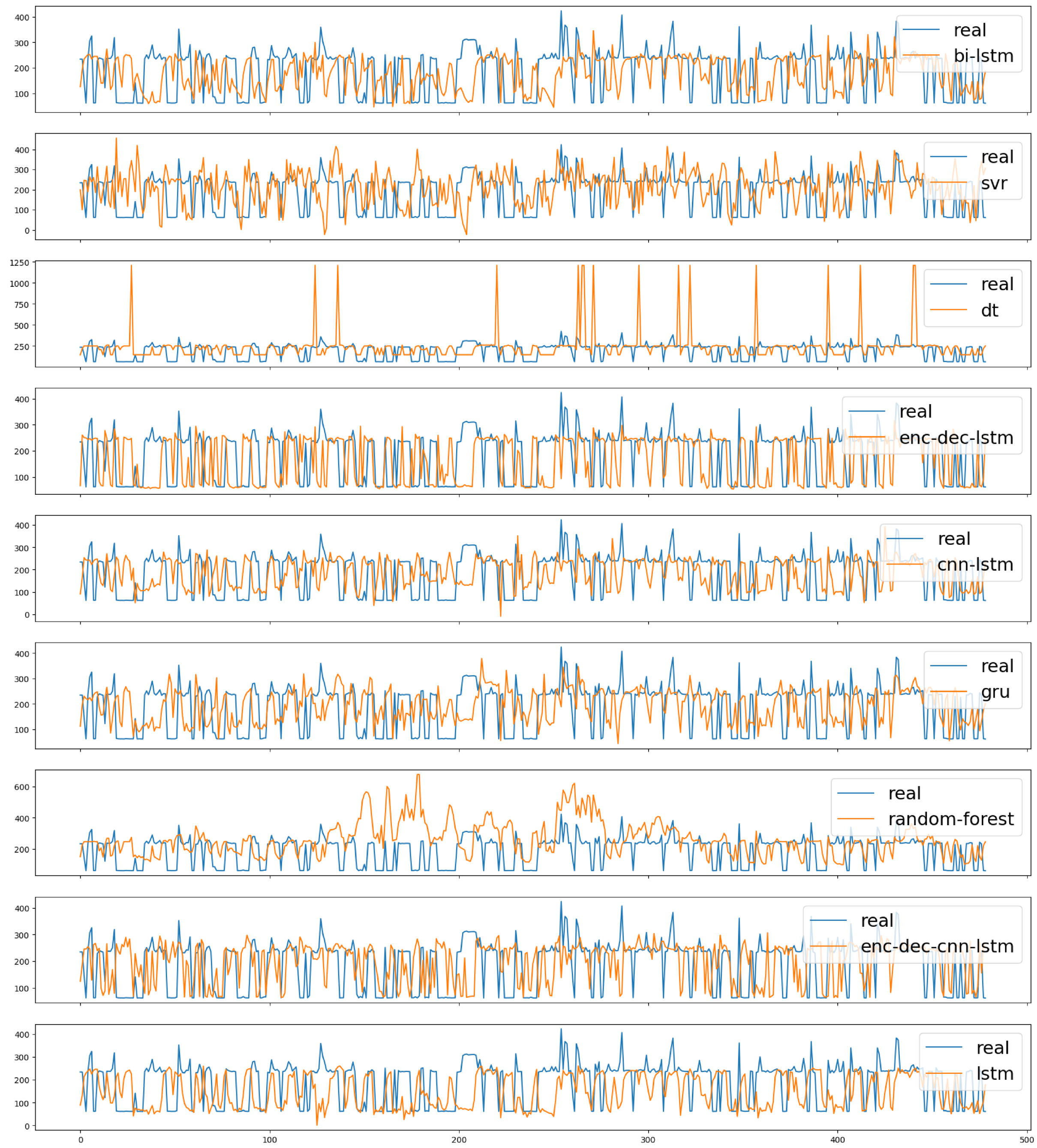


Figure 20 – Predicted values compared to the original/expected values, using the test portion of the BONO dataset and variation II.

For the BONO dataset, which has a larger number of rows but fewer features, the BI-LSTM model achieved the highest performance with 0.00288 MSE and 0.04129 MAE (its results were plotted in the second chart in Fig. 20). Following closely, the ENC-DEC-LSTM model secured the second position with 0.00335 MSE and 0.03984 MAE. The CNN-LSTM algorithm ranked third, exhibiting a 0.00366 MSE and 0.04175 MAE. On the other hand, among the evaluated models, the DT algorithm performed the poorest, yielding 0.01705 MSE and 0.05872 MAE, trailed by the RANDOM-FOREST model with 0.00796 MSE and 0.06511 MAE.

In this particular variation, the RANDOM-FOREST and BI-LSTM algorithms demonstrated the highest performance, with the BI-LSTM ranking first in both datasets, which emphasizes the versatility of this algorithm. However, when comparing the overall results (as highlighted in Fig. 19) to the results obtained in the first variation (as depicted in Fig. 17), the algorithms designed to predict using a multivariate structure yielded to worst results. Additionally, upon comparing the average training times in Table 6 with those in Table 5, it is evident that there was an overall increase for nearly all algorithms.

The inferior results observed can be directly attributed to the utilization of multiple features as input, as this study aims to evaluate performance without employing additional mechanisms to address the curse of dimensionality (DOKEROGLU; DENIZ; KIZILOZ, 2022), which says that the inclusion of numerous features can result in overfitting and a decline in model performance. This underscores the necessity for a feature selection mechanism when dealing with a multivariate approach. Such mechanisms play a critical role in reducing the dimensionality of the input data. They can be implemented using various techniques, such as statistical measures, genetic algorithms, and colony optimization, as presented in works in (ZHOU; HUA, 2022) and (KARIMI; DOWLATSHAHI; HASHEMI, 2023). However, these techniques necessitate additional time and hardware resources (DHAL; AZAD, 2022), which can be limited, particularly in IoT/Edge environments where resource constraints are prevalent.

4.2.3 Multi-step Analysis (Variation III)

Table 7 illustrates a comparison of algorithm performance in variation III. The table and simulation tests followed the same approach as the previously mentioned variants. In this variation, unlike the base scenario, the model generates predictions for multiple future steps instead of just a single step for each input set.

In the case of the KDN dataset, in this particular variation, the ENC-DEC-CNN-LSTM algorithm showcased superior performance, achieving a 0.00452 MSE and 0.04644 MAE (its results were plotted in the first chart in Fig. 21). The ENC-DEC-LSTM algorithm closely followed with comparable results, obtaining a 0.00852 MSE and 0.05917 MAE. The DT algorithm ranked third, demonstrating a 0.01343 MSE and 0.07734 MAE. Conversely, the GRU algorithm performed the worst among the evaluated models, with a 0.02144 MSE and 0.10902 MAE, followed by the LSTM with a 0.01991 MSE and 0.101 MAE.

For the BONO dataset, which contains a larger number of rows but fewer features, the ENC-DEC-CNN-LSTM model achieved the highest performance with 0.00249 MSE and 0.01567 MAE (its results were plotted in the second chart in Fig. 22). Following closely, the ENC-DEC-LSTM model secured the second position with an 0.00282 MSE and 0.01935 MAE. The GRU algorithm ranked third, exhibiting a 0.00394 MSE and 0.05056 MAE. On the other hand, among the evaluated models, the DT algorithm performed the

Table 7 – Performance results in variation III (univariate - multi-step - regression). Ordered by best to worst results.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
ENC-DEC-CNN-LSTM	0.00452 (±0.00098)	0.04644 (±0.00012)	602.96 s
ENC-DEC-LSTM	0.00852 (±0.00417)	0.05917 (±0.01656)	912.46 s
DT	0.01343 (±0.0)	0.07734 (±0.0)	0.09 s
SVR	0.01425 (±1e-18)	0.08438 (±0.0)	0.11 s
RANDOM-FOREST	0.01705 (±0.0)	0.09536 (±0.0)	27.84 s
BI-LSTM	0.01859 (±0.00384)	0.09761 (±0.01246)	1841.95 s
CNN-LSTM	0.01936 (±0.00259)	0.1019 (±0.00989)	670.98 s
LSTM	0.01991 (±0.00276)	0.101 (±0.0041)	478.98 s
GRU	0.02144 (±0.00223)	0.10902 (±0.00676)	110.81 s

(a) For KDN dataset.

Algorithm	**MSE (CI, 95%)**	**MAE (CI, 95%)**	**Time To Train**
ENC-DEC-CNN-LSTM	0.00249 (±0.00102)	0.01567 (±0.00013)	1546.93 s
ENC-DEC-LSTM	0.00282 (±0.00021)	0.01935 (±0.00122)	1675.26 s
GRU	0.00394 (±0.00014)	0.05056 (±0.00175)	2887.38 s
LSTM	0.00395 (±0.00013)	0.05018 (±0.00156)	971.13 s
RANDOM-FOREST	0.00397 (±6e-19)	0.04872 (±0.0)	1187.83 s
SVR	0.0041 (±6e-19)	0.05584 (±5e-18)	12.22 s
CNN-LSTM	0.00419 (±0.00026)	0.04977 (±0.00134)	4113.52 s
BI-LSTM	0.0042 (±0.00022)	0.04968 (±0.00112)	2686.76 s
DT	0.01004 (±7e-05)	0.07163 (±0.00021)	0.06 s

(b) For BONO dataset.

poorest, yielding a 0.01004 MSE and 0.07163 MAE, trailed by the BI-LSTM model with a 0.0042 MSE and 0.04968 MAE.

In this variation, it is important to highlight that the ENC-DEC-CNN-LSTM and ENC-DEC-LSTM algorithms performed significantly better than others when applied to both datasets. These algorithms were the only ones capable of achieving satisfactory results in this scenario. Although the overall prediction time was slightly longer than other variations, it remained under 1 second, which is considered acceptable for most applications.

Examining the predicted values (as depicted in Fig. 21), it becomes evident that the encoder-decoder algorithms produced decent results close to the expected values in the test datasets. However, their performance was slightly lower when compared to the results obtained in the base variation (illustrated in Fig. 17). This discrepancy can be

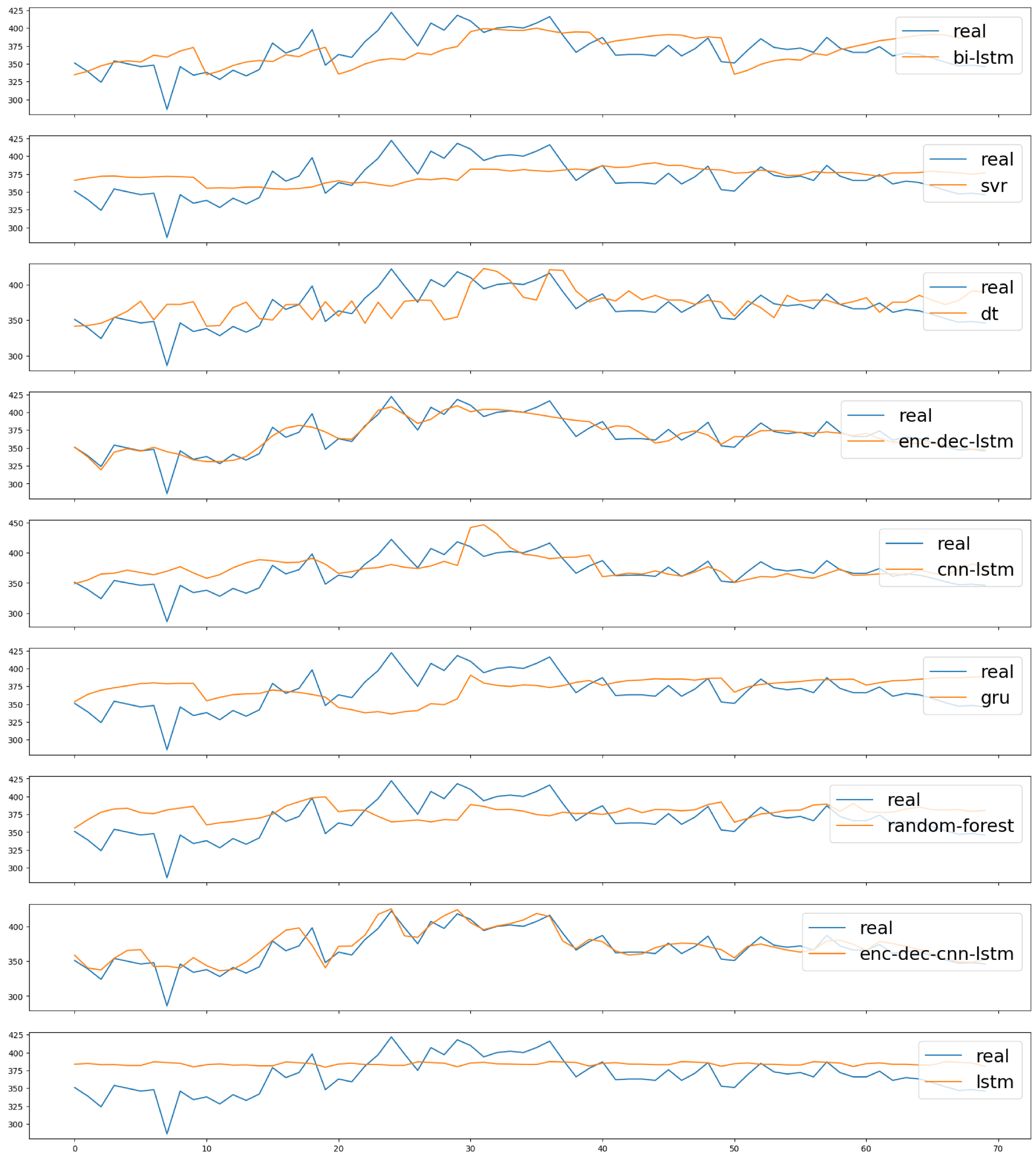


Figure 21 – Predicted values compared to the original/expected values, using the test portion of the KDN dataset and variation III.

attributed to the increased complexity associated with predicting multiple steps (WANG et al., 2016). The dependencies and interactions between the predicted steps introduce additional uncertainty into the forecasting task, accumulating cumulative error over the forecast horizon. Since each predicted step relies on previous predictions, any inaccuracies or errors in earlier steps can propagate and amplify over time, potentially resulting in significant deviations from the actual future values (GALICIA et al., 2019).

Finally, in IoT/Edge environments, we understand that predicting multiple future steps adds complexity that cannot be justified solely by the time it takes to make predic-

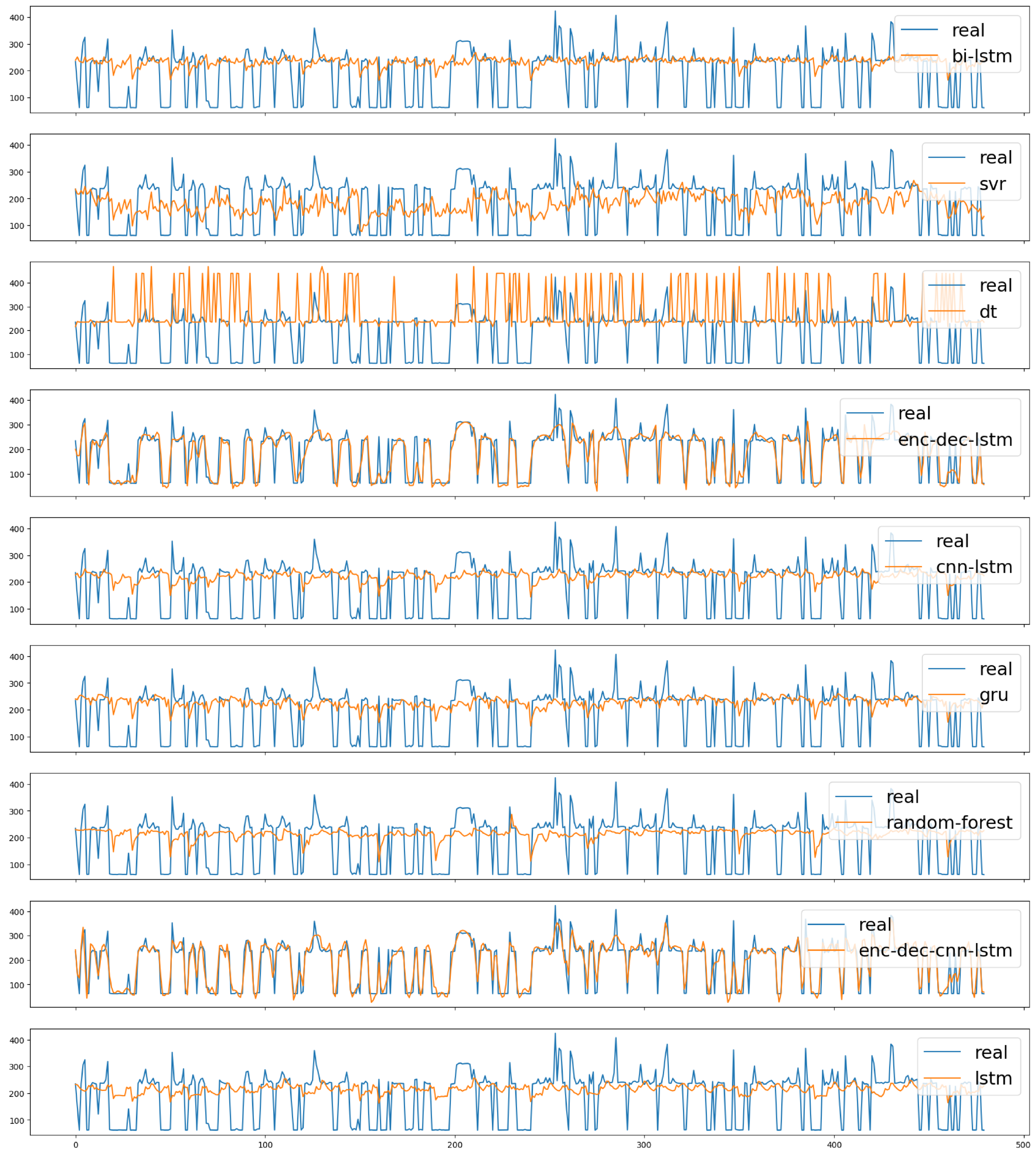


Figure 22 – Predicted values compared to the original/expected values, using the test portion of the BONO dataset and variation III.

tions because most algorithms and variations have demonstrated mean prediction times of less than 1 second. However, there are scenarios where obtaining current input values for the forecasting model is difficult or expensive or when predictions are made in the network's core and scaling decisions are delegated to edge elements (CHANTRY et al., 2021). In such cases, the use of a multi-step forecasting approach can be justified despite the added complexity and potentially less optimal results, and based on the results, the most prominent algorithms to address these tasks are the encoder-decoder-based algorithms.

4.2.4 Classification Analysis (Variation IV)

Table 8 compares algorithm performance in variation IV. The table and simulation tests were conducted using the same methodology as the previously mentioned variants. In this particular variation, different from the base scenario, the forecasting is performed through classification rather than regression. It involves predicting the probabilities of being assigned to a specific class from a fixed number of classes. As observed in Table 8, the cross-entropy results were not computed for the classical algorithms (DT, SVR, and RANDOM-FOREST) due to constraints with the framework used for measuring this metric. Consequently, the comparison was based only on the ACC (accuracy) results.

Table 8 – Performance results in variation IV (univariate - one-step - classification). Ordered by best to worst results.

Algorithm	**ACC (CI, 95%)**	**C.E. (CI, 95%)**	**Time To Train**
BI-LSTM	0.85714 (±0.00963)	0.46172 (±0.0512)	207.26 s
CNN-LSTM	0.85713 (±0.0079)	0.62572 (±0.08336)	115.16 s
ENC-DEC-LSTM	0.85143 (±0.0115)	0.51387 (±0.02724)	197.95 s
SVR	0.84286 (±0.0)	—	0.02 s
DT	0.83429 (±0.01815)	—	0.0 s
LSTM	0.83 (±0.03897)	0.65927 (±0.15105)	256.41 s
GRU	0.83 (±0.02125)	0.62711 (±0.05545)	238.3 s
RANDOM-FOREST	0.82857 (±0.0)	—	1.04 s
ENC-DEC-CNN-LSTM	0.81071 (±0.07271)	0.86193 (±0.37217)	312.55 s

(a) For KDN dataset.

Algorithm	**ACC (CI, 95%)**	**C.E. (CI, 95%)**	**Time To Train**
CNN-LSTM	0.81566 (±0.01481)	1.17626 (±0.0146)	1080.67 s
BI-LSTM	0.80647 (±0.04619)	1.16248 (±0.06533)	6419.48 s
SVR	0.80125 (±0.0)	—	4.11 s
LSTM	0.79729 (±0.02601)	1.16716 (±0.01271)	1647.53 s
RANDOM-FOREST	0.75324 (±0.0)	—	16.02 s
ENC-DEC-LSTM	0.74248 (±0.04455)	1.27947 (±0.08448)	10894.73 s
GRU	0.73215 (±0.0551)	1.30274 (±0.09124)	2737.17 s
DT	0.70877 (±0.03744)	—	0.02 s
ENC-DEC-CNN-LSTM	0.66169 (±0.01154)	5.91393 (±0.19814)	1122.86 s

(b) For BONO dataset.

In this specific variation of the KDN dataset, the BI-LSTM algorithm displayed superior performance, achieving 0.85714 ACC and 0.46172 CROSS-ENTROPY (its results were plotted in the first chart in Fig. 23). The ENC-DEC-LSTM model closely fol-

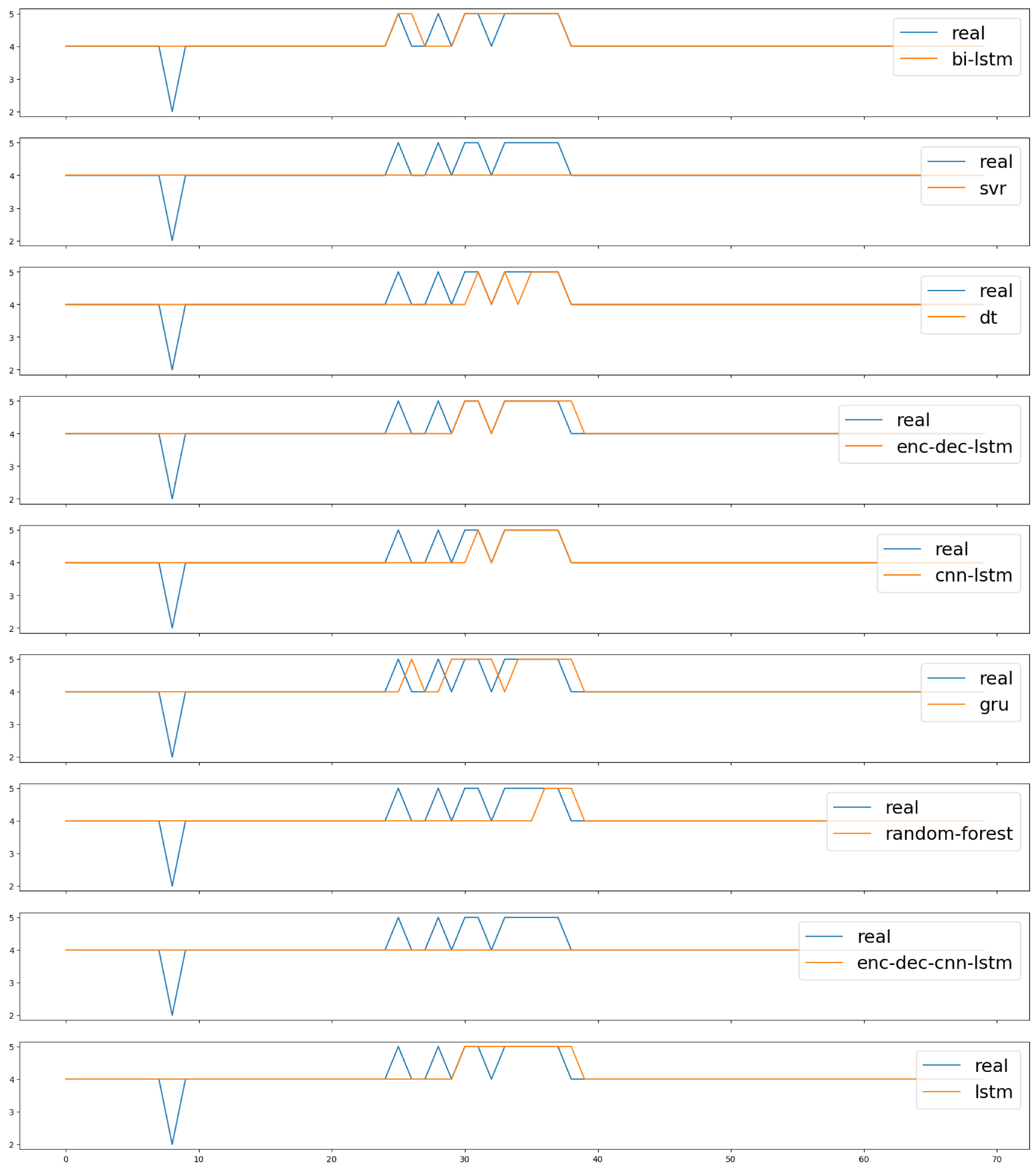


Figure 23 – Predicted values compared to the original/expected values, using the test portion of the KDN dataset and variation IV.

lowed with similar results, obtaining 0.85143 ACC and 0.51387 CROSS-ENTROPY. The SVR algorithm ranked third, exhibiting a 0.84286 ACC. In contrast, the CNN-LSTM algorithm performed the worst among the evaluated models, with 0.79429 ACC and 0.62572 CROSS-ENTROPY, followed by the ENC-DEC-CNN-LSTM with 0.81071 ACC and 0.86193 CROSS-ENTROPY.

Regarding the BONO dataset, the CNN-LSTM model achieved the highest performance with 0.81566 ACC and 1.17626 CROSS-ENTROPY (its results were plotted in the second chart in Fig. 24). Following closely, the BI-LSTM model secured the second

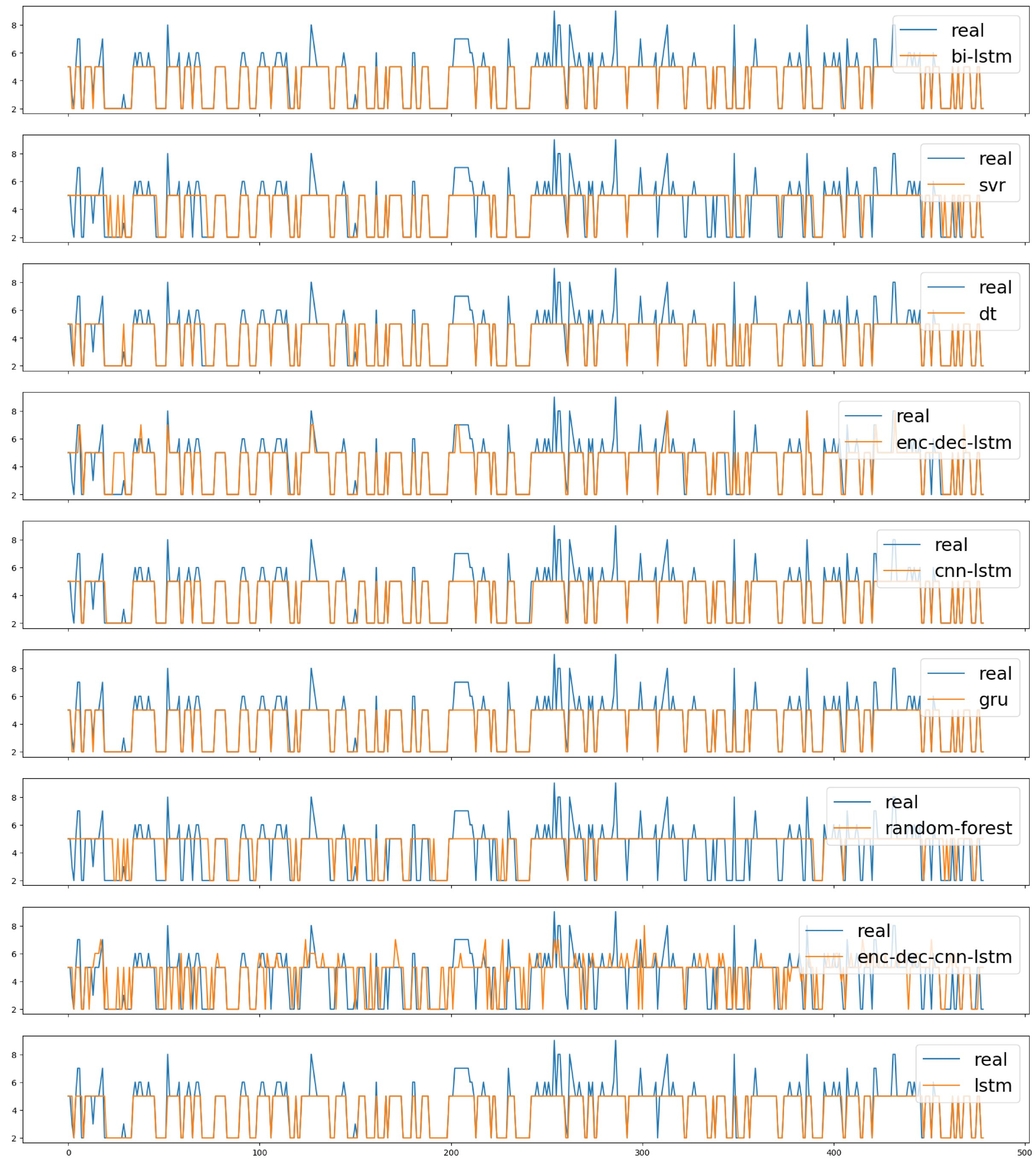


Figure 24 – Predicted values compared to the original/expected values, using the test portion of the BONO dataset and variation IV.

position with 0.80647 ACC and 1.16248 CROSS-ENTROPY. The SVR algorithm ranked third, demonstrating a 0.80125 ACC. On the other hand, among the evaluated models, the ENC-DEC-CNN-LSTM algorithm performed the poorest, yielding a 0.66169 ACC and 5.91393 CROSS-ENTROPY, followed by the DT model with a 0.70877 ACC.

As depicted in Figure 23 and Figure 24, the performance using the BONO dataset exhibited better results in this particular variation, benefiting from a larger pool of available training data. However, Table 8 reveals that the KDN dataset yielded higher overall accuracy than the BONO dataset. This highlights the significance of data characteristics, particularly when approaching forecasting as classification problems. The quantity of data available during the training phase is not the primary factor; instead, the relationship between the data's date, scale, and quality (its generality and diversity) plays a crucial role in achieving favorable outcomes (DOGAN; BIRANT, 2021).

However, in an IoT/Edge scenario, while obtaining significant results, approaching the solution as a classification problem instead of a regression problem imposes limits on the range of predicted values. This can restrict scaling actions to horizontal scaling approaches, where new instances are added or removed. Predicting scalar values such as CPU and memory as discrete classes, as done in this approach, may not always be feasible or optimal (SILVA et al., 2022). It restricts precise optimization and limits the flexibility of scaling decisions, potentially leading to suboptimal resource utilization at the edge. This lack of flexibility is detrimental to the IoT, where efficient resource utilization is crucial.

4.3 Overall Analysis

The findings revealed that when confronted with the diverse scenarios presented in an IoT/Edge environment, no single algorithm can serve as a universal solution to achieve optimal results for all demands. However, the results provide valuable guidance for determining effective approaches based on specific variations. For instance, in a common scenario where the objective is to predict future values of a monitored metric using univariate data and forecasting a single step, the results recommend employing LTSM or BI-LSTM models. Among these options, BI-LSTM is preferable as it demonstrates a good performance with fewer network layers, resulting in a shorter training time.

When dealing with multivariate data, our results indicate a preference for the RANDOM-FOREST and BI-LSTM models. Still, they also highlight that implementing a feature selection mechanism is crucial to enhance their effectiveness. In scenarios requiring predictions of multiple steps into the future, such as situations where obtaining input values for the forecasting model is challenging or expensive or when scaling decisions are delegated to edge elements within the network's core, the encoder-decoder-based algorithms, such as ENC-DEC-LSTM or ENC-DEC-CNN-LSTM, presented a clear advantage.

Lastly, if the task at hand involves classification rather than regression, which can simplify the scaling process, the CNN-LSTM or BI-LSTM algorithm proved to be a good choice. Nonetheless, it is worth noting that this approach may limit scalability flexibility by allowing only horizontal scaling actions, potentially compromising resource utilization efficiency in IoT environments where optimal resource allocation is of utmost importance.

Thus, using the above-found key points, for this work focused on developing an auto-scaling framework with an advanced predictive forecasting mechanism designed to handle a diverse range of metrics effectively, we opted to conceptualize the time-series forecasting

Algorithm	Mean Time To Predict
BI-LSTM	249.595 ms
SVR	9.3925 ms
DT	3.3975 ms
ENC-DEC-LSTM	255.925 ms
CNN-LSTM	204.122 ms
GRU	193.955 ms
RANDOM-FOREST	1141.385 ms
ENC-DEC-CNN-LSTM	210.715 ms
LSTM	223.2 ms

Table 9 – Average time of prediction of each algorithm, considering all prediction variation and datasets.

algorithm as a univariate single-step regression mechanism. This approach was grounded in utilizing a BI-LSTM model, empowering us to train and predict accurately while achieving fewer network layers and consequently lower required training time and less hardware resource demands. This decision was mainly based on the results demonstrated above and took into account the following considerations and analysis:

- Univariate: presenting better results compared to dealing with multivariate scenarios, this approach was chosen to compose the proposed auto-scaling tool influenced by: (1) It offered a significantly shorter training time, making it compatible with strategies like training new models instead of retraining old ones. This attribute aligns well with the proposed re-validation mechanism in RAFE; (2) The univariate approach tended to consume fewer hardware resources during training and prediction phases due to its simpler data structure. It relies on an array of data rather than multidimensional matrices, requiring less hardware resources. This factor makes it particularly suitable for network edge scenarios, and mainly, (3) This approach, in contrast to others, permitted a finer level of configurability in vertical scaling, not limiting the solution to horizontal scaling. This adaptability contributes to achieving optimal performance in the context of auto-scaling (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014b). The vertical scaling aspect will be explored in future research efforts. Moreover, it's worth noting that multiple metrics can be combined into a single one using a solution as proposed in works like (CUNHA, 2021).

- Single-step: as presented in Table 9, the average prediction time, or time to predict, tends to be smaller than 1 second for all the algorithms. This time is a sufficiently short duration for the majority of applications to adopt single-step forecasting. Consequently, we opted for the single-step approach, which exhibited superior performance in our evaluations and required less time and hardware resources during the training phases. However, it's noteworthy that collecting current metric values for predictions in certain contexts may cause significant operational costs or introduce delays. For these cases, a multi-step mechanism can be implemented in future work.

❏ Regression: in contrast to the classification method, which requires a predefined discrete set of classes, the regression approach does not have such restrictions and is capable of predicting continuous values. This characteristic holds great significance for our proposed application, as it aims to be versatile, predicting various metrics and being suitable for multiple applications. The adoption of a classification approach could potentially limit the applicability of our method. Moreover, our tests have shown better performance results in using regression methods compared to classification methods.

❏ BI-LSTM Algorithm: As previously emphasized, the BI-LSTM algorithms consistently demonstrated high performance across various scenarios, occasionally outperforming others. Notably, in the first compared variation, base variation, which closely resembles the methods applied in our proposed mechanism, it presented some of the best results. Furthermore, it exhibited good performance even when composed with fewer network layers, leading to shorter training times. Consequently, we have selected the BI-LSTM algorithm as the primary choice for our application.

These insights were the source for building the auto-scaling framework architecture named RAFE. This architecture introduced several mechanisms to achieve optimal performance, including a retraining process, self-evaluation, and distributed training. The overarching goal is to maximize the benefits of these selected characteristics while mitigating their associated drawbacks. We discussed the details of this proposed architecture in Section 3.3.2, and its evaluation will be presented in the subsequent sections.

Chapter 5

Experimental Evaluation and Analysis

The contributions of this research can be delineated into two parts. In the second part, to simulate the dynamic characteristics of edge environments, a series of experiments are executed involving the deployment and training of each algorithm over various forecasting methodologies and configurations. These experiments evaluate the algorithm performance in typical edge and cloud computing scenarios. At last, the collective outcomes are compared to identify the algorithms best suited for integration into RAFE, the proposed auto-scaling solution.

This chapter introduces and assesses an innovative approach to address auto-scaling challenges in edge and cloud applications. This approach combines reactive and predictive strategies to ensure optimal scaling and policy decisions. Additionally, it introduces the approach to splitting the application into core and workers, which adeptly manages hardware resource limitations by executing machine learning training and resource-intensive tasks in a distributed way. The approach also introduces diverse tools crafted to handle the dynamic nature of edge and cloud environments. A multi-univariate-models approach is applied to achieve optimum results and keep a low time for training the model. Moreover, a re-validation mechanism is introduced to effectively handle shifts in workload patterns inherent in network environments. This comprehensive suite of tools aspires to enhance the performance and adaptability of auto-scaling in these intricate and ever-evolving contexts.

To assess the performance of RAFE, we first carry out three practical experiments in which we manipulate simulated network traffic and the auto-scaling mechanisms. These experiments evaluate the effectiveness of RAFE under diverse scenarios, encompassing both expected and unanticipated traffic patterns. Additionally, we assess the overall architecture, delving into critical aspects such as the time required for model training, the resource utilization of the solution, and its capacity for seamless integration.

5.1 Architecture Evaluation

This section comprehensively evaluates the RAFE *(Resource Auto-scaling For Everything)* framework. The architectural details are expounded upon in Section 3.3.2. First, we outline the experimental environment's architecture, purposefully designed to assess this auto-scaling framework's performance. Subsequently, leveraging this dedicated environment, we conduct a series of experiments. Finally, we present the outcomes and provide an in-depth analysis.

We initiate the assessments by evaluating RAFE's performance in terms of QoS and cost-effectiveness. To this end, we conduct three main experiments to assess RAFE's performance in: (1) common scenarios, where it has prior knowledge of the traffic patterns (2) its adaptability in complex scenarios, encompassing entirely novel and untrained traffic patterns. This is essential, given the inherent variability in network conditions. Moreover, we evaluate the invalidation mechanism and whether or not RAFE can achieve optimum results even when drastically different traffic patterns are performed. Other works in the state of the art do not commonly evaluate these less-explored edge scenarios. However, they hold significant relevance due to the network's dynamic nature, which requires adaptable and responsive auto-scaling applications.

Moreover, we evaluate key factors for the proposed framework. We analyze the time required to train the deep-learning models used in the predictive module. This is critical for supporting multiple models and revalidation approaches RAFE proposes. Additionally, we evaluate RAFE's integration capabilities, as it aspires to operate seamlessly across different network domains, including IoT, NFV, and MEC applications. To this end, we examine its compatibility with the widely applied ETSI's NFV Reference Architecture, commonly used in VNF and MEC environments. Finally, we provide a comprehensive analysis of both RAFE's performance and the findings of this study.

5.2 Experimental Environment

To assess the effectiveness of the proposed framework, we established an experimental testbed illustrated in Fig. 25. This architecture serves to generate network traffic, oversee and adjust VNF/MEC applications (functioning as a MANO system aligned with the ETSI's VNF reference architecture), monitor the utilization of hardware resources, and employ RAFE for application auto-scaling.

The K6 (GRAFANA, 2023b) load testing tool generates the network traffic. This tool facilitated the simulation of Virtual Users (VUs), representing entities capable of executing HTTP requests based on predefined scripts. By employing K6, we could emulate a substantial number of concurrent users engaging with VNF/MEC applications in a specified pattern, introducing variance in the number of requests to yield more realistic

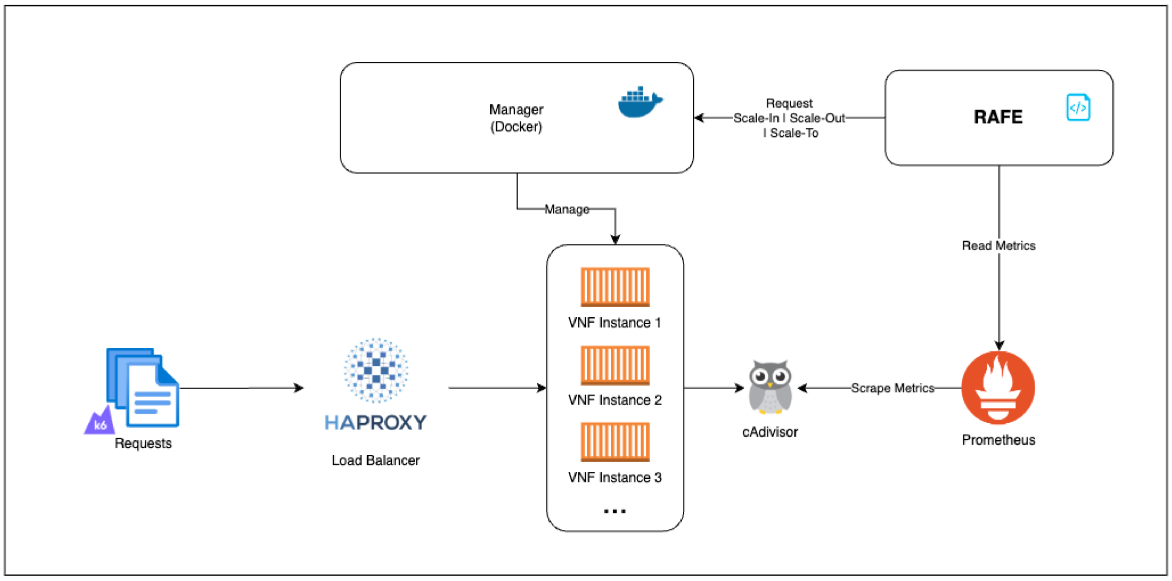


Figure 25 – Experimental testbed architecture.

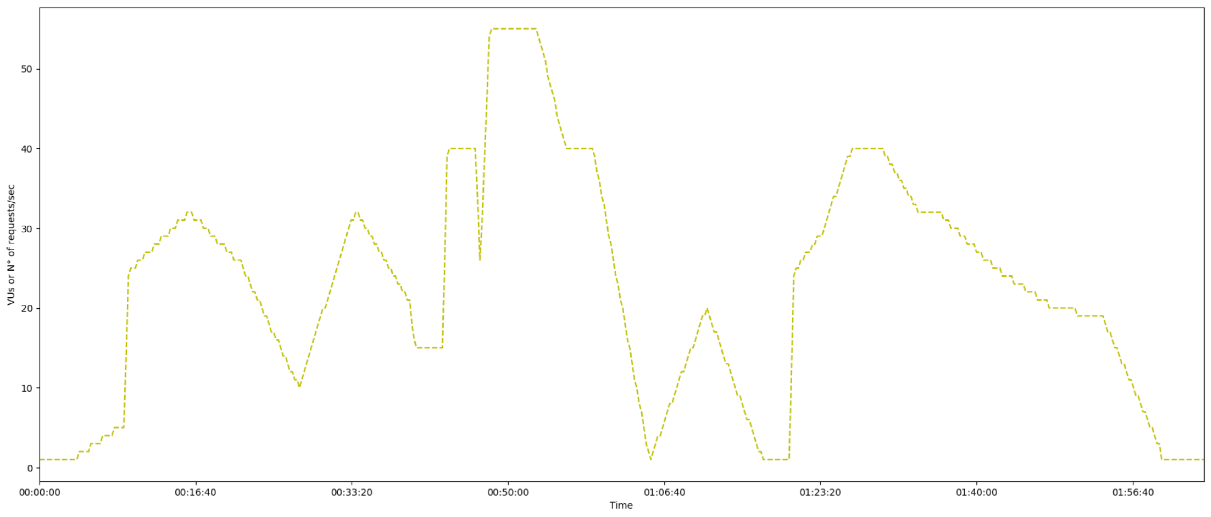
and robust results.

To control and host the VNF/MEC applications while validating the RAFE's architecture capacity of integrating with diverse virtualization solutions, we implemented a basic Docker ecosystem, serving as the orchestrator as depicted in Fig. 25. The orchestrated network traffic is directed through a load balancer, implemented using HAProxy (HAPROXY, 2023), employing a round-robin strategy to distribute requests among the available instances evenly.

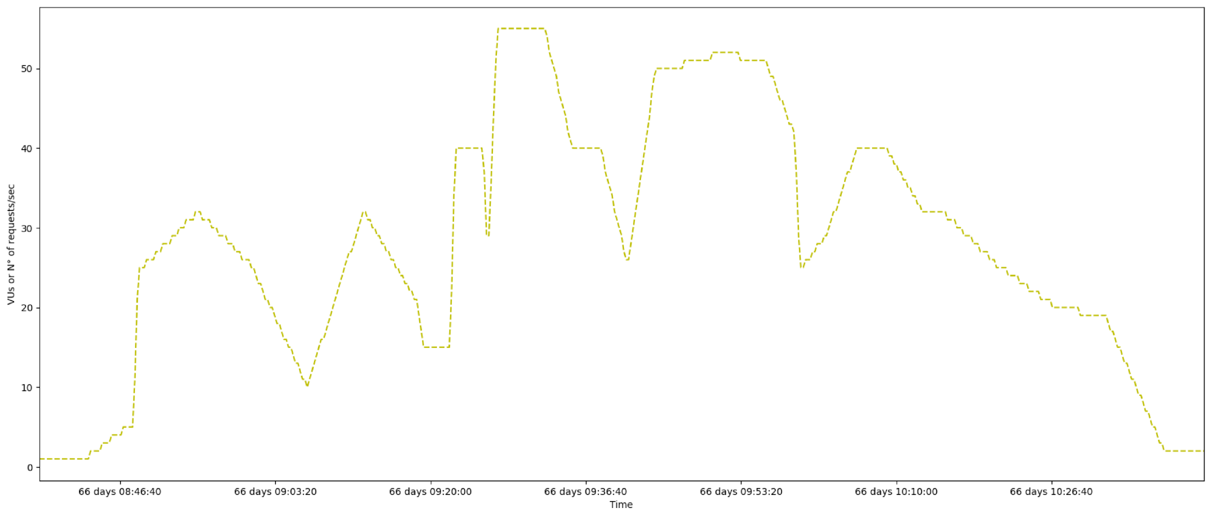
For our reference VNF/MEC application, executed within the Docker containers, we employed an HTTP Web Server developed in Python. This application exposes services that simulate intensive CPU and memory consumption, executing multiple factorial operations and allocating memory blocks accordingly.

To capture and export the hardware usage metrics of the containerized applications in real-time, we employed cAdvisor (GOOGLE, 2023a). Acting as a running daemon, cAdvisor collects, processes, and exports container metrics, retaining each container's resource isolation parameters and historical resource usage. These metrics are then monitored and stored using Prometheus (RABENSTEIN; VOLZ, 2015), which scrapes the exported data from cAdvisor and organizes them into a scalable and flexible time series database.

In summary, as illustrated in Fig. 25, the testbed simulation begins with K6 generating traffic, which is then directed and distributed by a load balancer to multiple containers hosting the reference VNF/MEC application—an HTTP API server simulating high CPU and memory consumption, all orchestrated within a Docker environment. The containers are continually monitored, and cAdvisor exposes their corresponding hardware usage metrics. Prometheus scrapes this data to construct a time series database, which the proposed application, the RAFE framework, utilizes. The RAFE framework processes and interprets the monitored metrics, making scale decisions through its reactive and predictive autoscaling mechanisms discussed earlier in this work.



(a) Initial traffic load used to train and validate the autoscaling mechanisms.



(b) Variant of the traffic load defined in Fig. 26a, used to validate the autoscaling mechanisms in a non-seen traffic load pattern.

Figure 26 – Traffic loads utilized in the experiments expressed by Virtual Users (VUs) or approximately the number of intensive resource-consuming requests per second.

We simulated a network scenario for the evaluations where each NFV/MEC application encounters simultaneous requests, following repetitive patterns outlined in Figs. 26. In this context, each VU generates approximately one resource-intensive request per second. For simplicity and expedited evaluations in prototype experiments, we adopted a temporal equivalence of 1 minute in real life to 10 minutes in the experimental environment (1-10). Consequently, each day in the experimental environment could be simulated in 144 minutes.

The workload pattern depicted in Fig. 26a was chosen to simulate a 24-hour interval of network traffic typical for a VNF/MEC application designed for vehicular or mobile solutions. This pattern includes intensive accesses at 6 AM, 12 PM, and 6 PM—representing common transit traffic peaks—with maximum VUs of 32, 55, and 40. It's crucial to note that due to the temporal equivalence of 1-10, the proposed application was configured to execute reactive autoscaling validations every 10 seconds (100 seconds in simulation) and predictive autoscaling validations every 1 minute in real life (10 minutes in simulation). The first traffic load was simulated using the RAFE testbed architecture with only the predictive autoscaling mechanism. Monitored CPU and memory resource usage data were used as a dataset to train the AI-based predictive autoscaling mechanism. This same traffic pattern was employed to validate the performance of all autoscaling mechanisms implemented by RAFE (predictive, reactive, and hybrid).

However, as network solutions may not always handle the same workload, we introduced an unseen traffic load pattern (Fig. 26b) to evaluate how different autoscaling approaches react to changes in the learned/expected traffic pattern. Although both traffic patterns are similar, they simulate slightly different scenarios to assess how the mechanisms handle variations. The patterns also include a significant contrast in the center, representing a burst of traffic simulating a non-expected and abrupt increase in network load.

In summary, our first experiment assesses the proposed framework. It compares different autoscaling mechanisms using the testbed and the traffic load described in Fig. 26a to train the predictive model and validate all mechanisms. The second experiment introduces the second traffic load, described in Fig. 26b, with the previously trained mechanisms to evaluate their response to untrained and different workloads. We measure performance based on the QoS offered by the VNF/MEC applications, maintaining a low timeout of 10 seconds for requests and collecting request errors during each mechanism evaluation.

Regarding configuration settings for scaling mechanisms, each VNF/MEC application was configured with a maximum of 500 millicore CPU units and 512 MB of memory. For reactive autoscaling, the scale-in threshold was set to 150 millicore CPU units, and the scale-out threshold was set to 400 millicore CPU units to enable prompt reactions to changes in network traffic. Predictive autoscaling parameters for training included a time lag of 72, a training factor of 14, and a step of 1 minute (10 minutes in the experiment's time equivalence). Considering the time equivalence, these configurations represent the utilization of the past 168 hours (7 days) of monitored metrics data, collected every 10 minutes, to train a deep learning model for predictive autoscaling. The experimental evaluations were conducted on a laptop featuring a Ryzen 7 5700U processor with 8 cores clocked at 1.8 GHz and 16GB of memory.

5.3 Handling Known Traffic Patterns (Performance Results)

This initial experiment assesses RAFE's performance in managing familiar and trained traffic patterns—a scenario commonly encountered by applications and frequently explored in state-of-the-art evaluations. The objective here is to subject the mechanism to a specific traffic pattern, allowing it to undergo training and subsequently reapply the same pattern to assess performance and ascertain whether the application has effectively learned from the initial exposure, leading to improved results upon pattern repetition. By focusing on known and trained patterns, this experiment provides valuable insights into RAFE's ability to adapt and optimize its operations based on familiar traffic scenarios. The repetition of the traffic pattern allows for examining the mechanism's learning capabilities and capacity to enhance performance over successive encounters with the same set of conditions. This scenario mirrors real-world conditions where applications often face recurring patterns, making it critical to evaluate autoscaling mechanisms' robustness and learning efficiency.

To this end, first, we simulated the traffic load pattern illustrated in Fig. 26a over a continuous span of 2 days, equivalent to more than 2 weeks due to the established time equivalence. As outlined in the testbed definition, all generated requests were directed toward the VNF/MEC applications. Throughout this period, the RAFE framework was configured to exclusively employ the reactive autoscaling mechanism. Following the 2-day simulation under the specified workload, the predictive module used the resulting metrics to train the forecasting AI models, utilizing the BI-LTSM-based algorithm elucidated earlier. The consumed CPU and memory metrics were employed as inputs for scale, leading to the development of two distinct models by the predictive mechanism (one for each metric).

With these outputted models, we conduct further simulations using the same testbed and traffic load. For the subsequent 6 hours, the RAFE framework was configured to use the predictive autoscaling mechanism exclusively. Following this, the process was reiterated for an additional 6 hours, with the framework employing both reactive autoscaling and predictive autoscaling mechanisms concurrently (hybrid approach).

The results of these interactions are presented in the subsequent sections, organized according to the context of each scaling mechanism.

5.3.1 Reactive Autoscaling

In this first experiment, RAFE's Reactive Autoscaling module achieved the results presented in Fig. 27. The blue line illustrates the number of instances/containers the reactive module instantiates over time. The horizontal blue bars indicate the number of

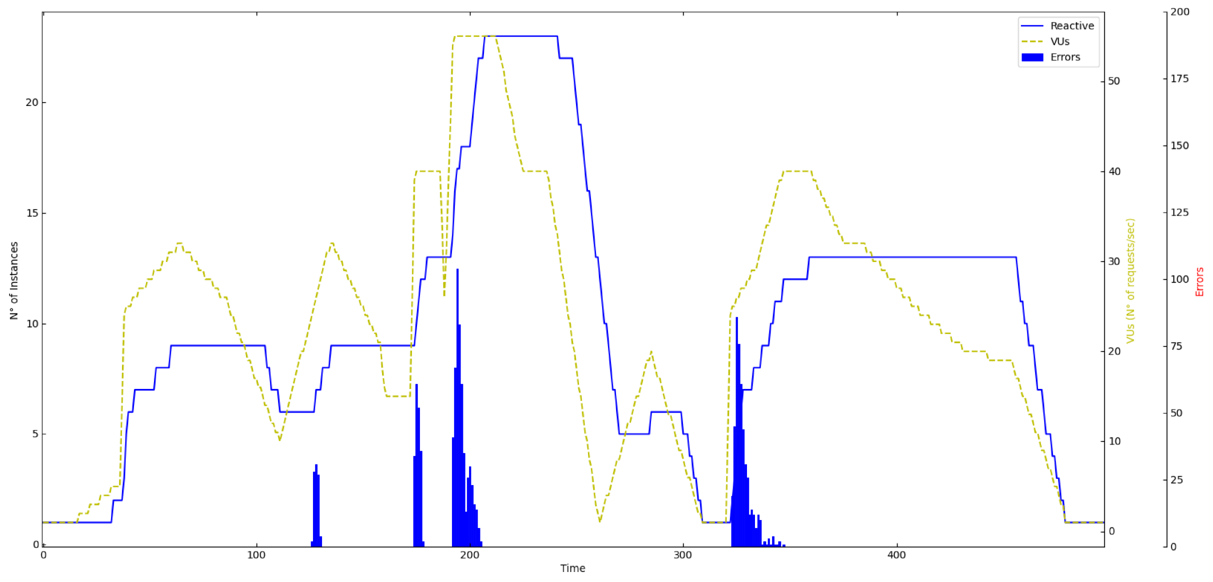


Figure 27 – Reactive Autoscaling module results, during the first experiment, handling the traffic pattern defined in Fig. 26a.

request errors observed throughout the experiment. The yellow dashed line represents the workload for comparative purposes, indicating the number of VUs or approximate requests per second over the same time.

Table 10 – Requests and errors during the first experiment using the reactive method.

Requests	**Errors**	**Errors (%)**
142845	1271	0.88977%

Furthermore, to complement the visual presentation of results and to provide a quantitative perspective on network request errors, Table 10 furnishes data on the total simulated requests, the instances of request errors encountered during the experiment, and the corresponding error percentage relative to the total request amount.

Assessing the performance of the predictive module in this first experiment, several key observations come to light:

- The number of instances aligns with the network's behavior but exhibits a noticeable lag, particularly when scaling down instances.

- The delay in scaling down instances primarily occurs due to the reactive module's decision-making process, which relies on the average utilization of monitored metrics (memory and CPU units in this test). Consequently, during reduced traffic and resource consumption periods, it spawns multiple instances that utilize only a fraction of the allocated resources instead of maintaining fewer instances using a larger slice of available resources.

- The general delay to implement the scaling actions is also largely attributable to the reactive solution's practice of incrementing (scale-out) or decrementing (scale-in) only one instance at a time when evaluating scaling decisions. Consequently, during rapid traffic spikes, there is a delay in meeting the required number of instances to handle the increased traffic load.

- Notably, during a quick increase in network traffic (particularly around points 140, 180, 200, and 330 in Fig. 27), an initial burst of errors occurs, followed by a gradual increase in the number of instances, resembling a step-like pattern. This pattern precisely reflects the correlation between traffic surges and bottleneck occurrences.

- Despite the aforementioned observations, it's important to highlight that the reactive module generally exhibits great performance, successfully handling over 99.2% of requests even within the network pattern depicted in Fig. 26a, which presents multiple occurrences of rapid and substantial traffic growth.

5.3.2 Predictive Autoscaling

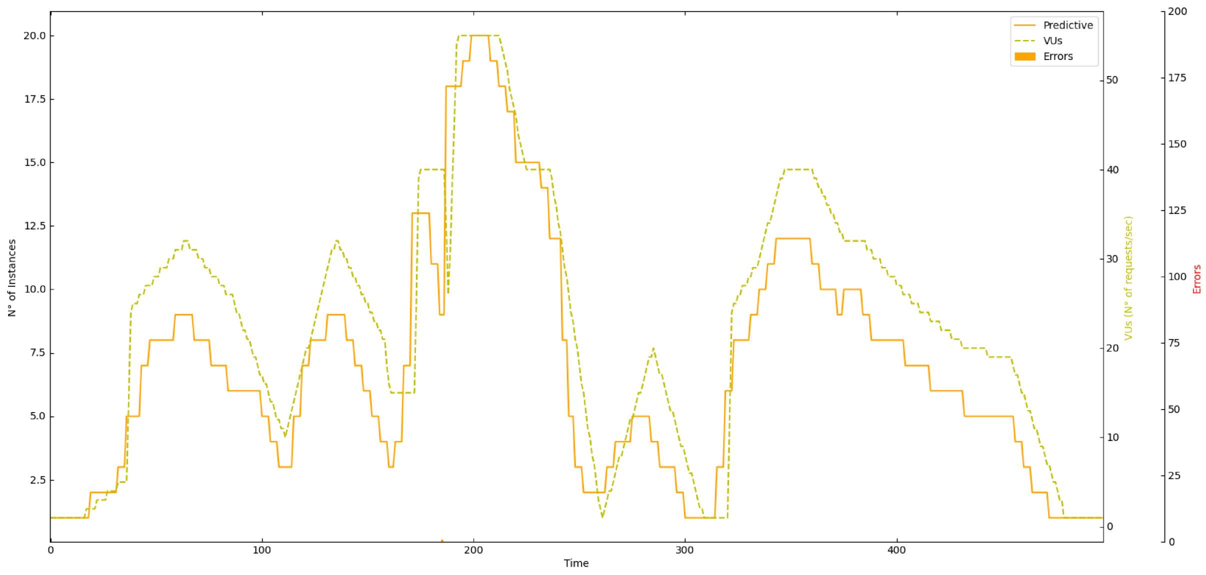


Figure 28 – Predictive Autoscaling module results, during the first experiment, handling the traffic pattern defined in Fig. 26a.

In this initial experiment, the Predictive Autoscaling module demonstrated its performance as depicted in Figure 28. The orange line in the graph illustrates the evolution of the number of instances or containers spawned by the predictive module over time. Concurrently, the horizontal orange bars (which, due to the short quantity, ended up not being present) correspond to the occurrences of request errors at various time points. To facilitate a workload reference for comparison, the yellow dashed line tracks the number of VUs, or roughly the requests per second, simultaneously.

Table 11 – Requests and errors during the first experiment using the predictive method.

Requests	**Errors**	**Errors (%)**
147231	1	0.00067%

Furthermore, to enhance the comprehensibility of the results and provide a quantitative perspective on request errors, Table 11 furnishes data on the total simulated requests, the instances of request errors encountered during the experiment, and the corresponding error percentage relative to the total request volume.

Evaluating the performance of RAFE's predictive module, it is to be noted:

- In an overall analysis, this approach shows a more precise alignment between the number of instances and the network's behavior, especially when comparing it to the Reactive Autoscaling approach. In this context, the number of instances closely mirrors the fluctuations in traffic, displaying remarkably similar patterns.
- Notably, the alteration in the number of instances (depicted by the orange line) occurs before changes in traffic (indicated by the yellow dashed line). This serves as a demonstration of the effect of forecasting future values.
- In Predictive Autoscaling, unlike in the reactive approach, the system can scale to any number of instances required to match the predicted values, whether it involves an increase or decrease in the number of instances. This shows significant improvements in the responsiveness of the system, particularly during sudden traffic spikes.
- It's worth noting that this approach eliminates any delay in scaling down, effectively preventing the scenario where multiple instances only utilize a fraction of the allocated resources, as observed in the Reactive Autoscaling approach. This improvement is achieved by dividing the model's predicted output by the specified maximum allocated resources per instance, ensuring optimal resource utilization. Consequently, this approach maximizes resource efficiency, leading to cost savings for the operator.
- Remarkably, when examining the error rates, as detailed in Table 11, it becomes evident that this approach outperforms the previous one. It exhibits significantly fewer errors (almost none) and achieves this with fewer instances, indicating superior performance while conserving resources.

5.3.3 Hybrid Autoscaling

In turn, the RAFE's Hybrid Autoscaling module achieved, in the first experiment, the results presented by Fig. 29. The blue line presents the number of instances/containers the

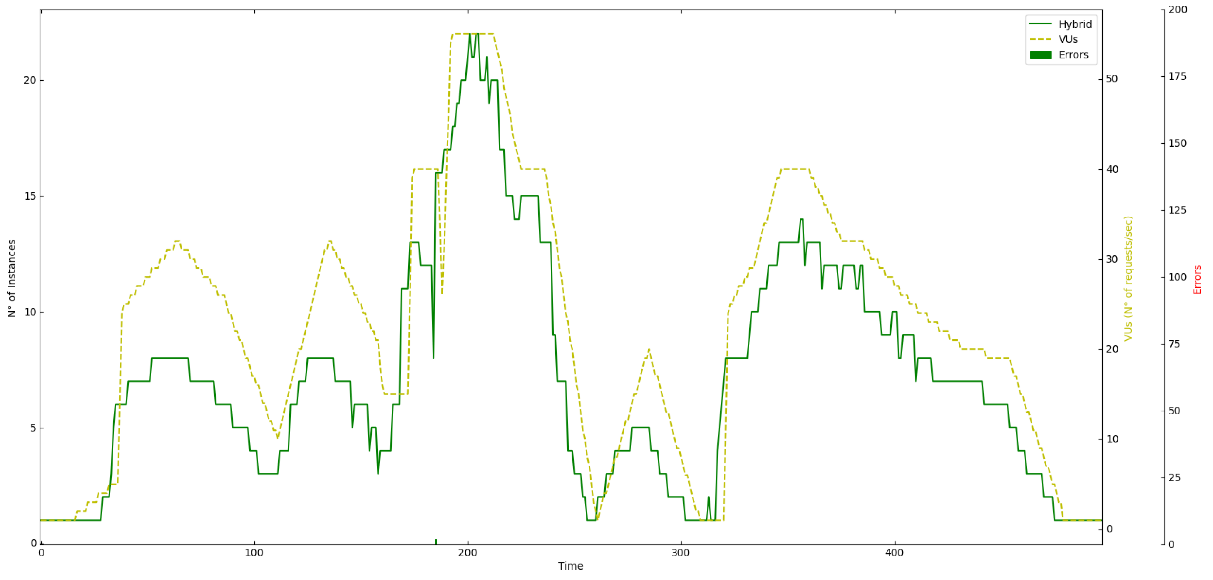


Figure 29 – Hybrid Autoscaling module results, during the first experiment, handling the traffic pattern defined in Fig. 26a.

reactive module instantiates over time. The horizontal bars in blue represent the number of request errors over time. Additionally, to reference the workload for the comparison, the yellow dashed line represents the number of VUs (or approximately the requests per second) over time for the period.

Table 12 – Requests and errors during the first experiment using the hybrid method.

Requests	**Errors**	**Errors (%)**
150238	2	0.00133%

Moreover, to complement the result visualization and to give a numeric vision of the request errors, Table 12 presents the number of requests simulated during the experimentation, the number of request errors that occurred during this evaluation and the percentage that these errors represent over the amount of the requests.

Evaluating the performance of the hybrid module and comparing it with the previous approaches, some highlights become apparent:

- It demonstrates similar results to those presented by the Predictive Autoscaling module regarding error rates and the observed pattern of instance numbers. This occurs mainly because the predictive approach takes action proactively before traffic changes occur, eliminating the need for reactive responses. It underscores the effectiveness of the predictive approach, particularly when dealing with a known traffic pattern.

- Especially around specific points in time, such as 170, 190, 210, 310, and 370 (as indicated along the X-axis in Fig. 29), we witnessed rapid fluctuations in the number of instances, resulting in shifts. These fluctuations were a consequence of conflicts arising between the actions of the Predictive Module and the Reactive Module. These conflicts occurred when one module initiated a scaling action that contradicted the scaling decision of the other module.

❏ When comparing the error rates, as depicted in Table 12, with the errors encountered during validation with other modules, a notable similarity emerges again with Predictive Scaling. This again underscores the effectiveness and consistency of the Predictive Scaling approach in delivering favorable results once dealing with a known traffic pattern.

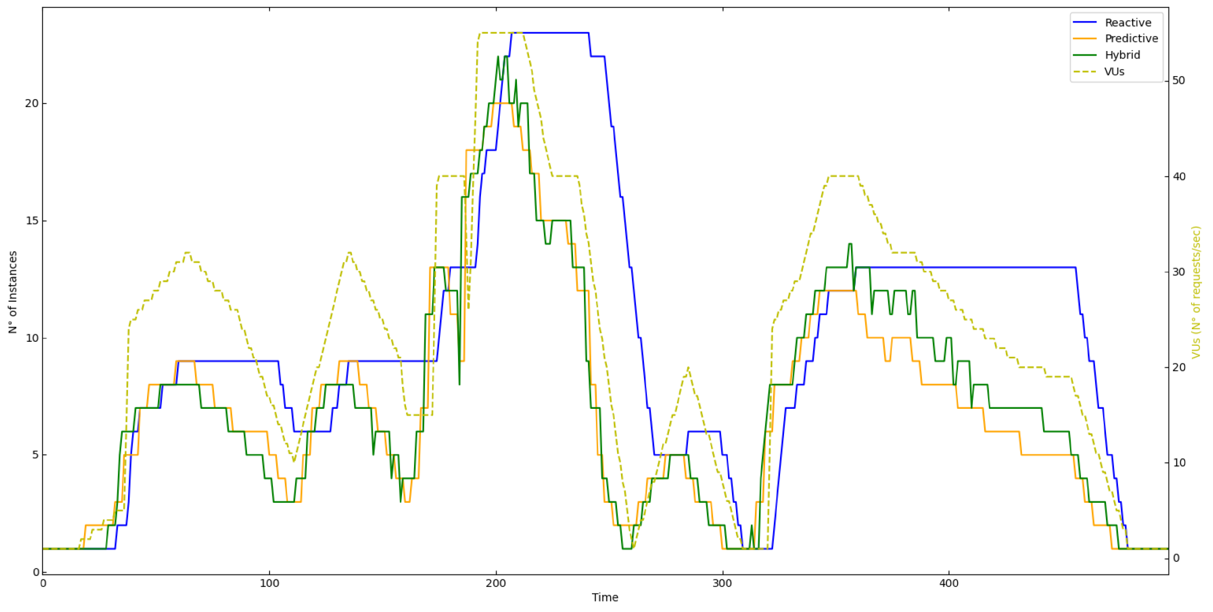
5.3.4 Experiment Analysis

Analyzing the overall results of the first experiment, Fig. 30 presents and compares the performance of each autoscaling mechanism (reactive, predictive, and hybrid) implemented in RAFE. In Fig. 30a, the blue, orange, and green lines depict the evolution of instances/containers over time initiated by the reactive, predictive, and hybrid modules, respectively. Additionally, To provide a workload reference for comparison, the yellow dashed line represents the number of VUs or, approximately, the requests per second over the same duration. Fig. 30b supplements Fig. 30a by incorporating horizontal bars that represent the occurrence of request errors during the validation of each autoscaling mechanism over time. The color coordination ensures a clear correspondence between each mechanism and its associated request errors, where, for instance, the yellow horizontal bar aligns with the mechanism represented by the yellow bar, and the same principle applies to other components.

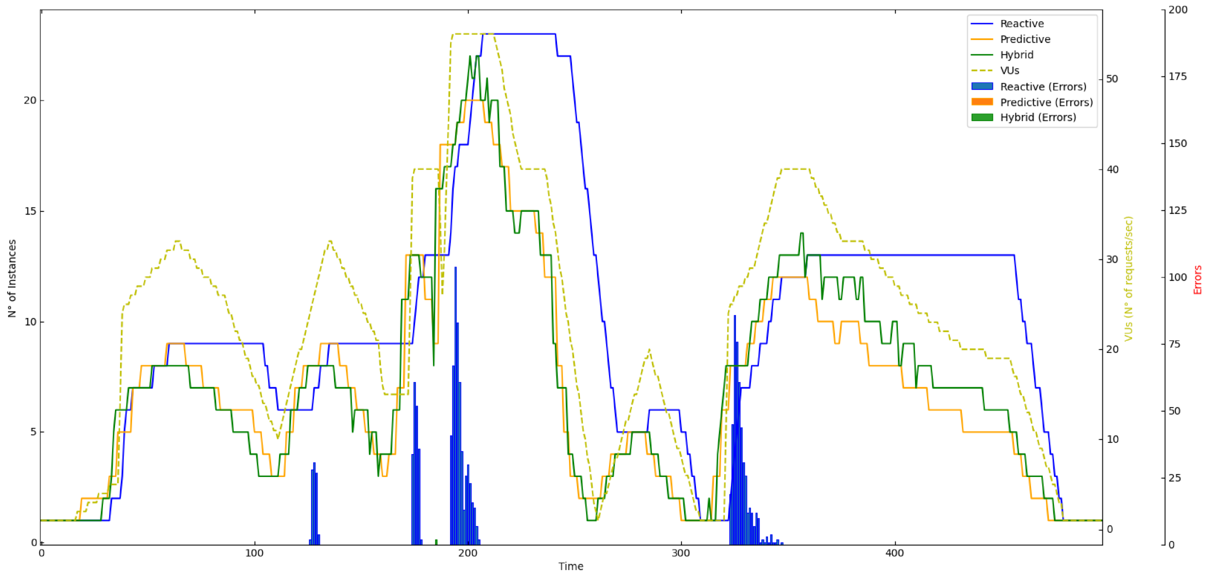
Table 13 – Requests and errors in each autoscaling mechanism during the first experiment.

	Requests	**Errors**	**Errors (%)**
Reactive	142845	1271	0.88977%
Predictive	147231	1	0.00067%
Hybrid	150238	2	0.00133%

Additionally, to complement the insights presented in Fig. 30 and provide a quantitative perspective, Table 13 furnishes details for each assessed autoscaling mechanism. This includes the count of simulated requests during the experiment, the instances of error encountered, and the corresponding percentage of errors relative to the total number of requests. It's worth noting that the variability in the number of requests stems from the dynamic nature of the simulated VUs. Rather than simulating a fixed number of requests, the VUs are designed to mimic various virtual users, introducing realism by simulating simultaneous requests with some degree of fluctuation.



(a) Comparison between the autoscaling mechanisms results in the first experiment.



(b) Comparison between the errors during each mechanism evaluation in the first experiment.

Figure 30 – Comparison conducted among reactive autoscaling, predictive autoscaling, and their hybrid implementation, addressing the traffic pattern outlined in Fig. 26a. Additionally, the chart includes the representation of VUs or requests per second, serving as a benchmark for workload reference throughout the comparison.

Analyzing the results, it is notable that reactive autoscaling (depicted by the blue line) tends to respond more slowly to sudden increases in traffic (indicated by the yellow line) compared to predictive autoscaling (represented by the orange line). Furthermore, it becomes evident that reactive autoscaling is less cost-efficient than its predictive counterpart. It necessitates more instances to handle an equivalent amount of traffic and takes more time to respond to decreases in traffic, reducing the number of instances.

In Fig. 30, the green line represents the hybrid approach, employing both reactive and predictive autoscaling simultaneously. While this hybrid strategy yields similar results to predictive autoscaling, some distinctions emerge, particularly during abrupt changes in traffic influenced by divergences with the reactive component.

A closer examination of the errors axis in Fig. 34b and the data in Table 13 reveals that a reactive autoscaling approach alone may not always meet the requirement for the correct number of containers to satisfy specific demands. This shortfall can impact the QoS and extend the overall round-trip time of an NFV/MEC application. Several factors contribute to this inefficiency: (1) the waiting time between reactive validations, as reactive autoscaling adjusts one instance at a time, sometimes failing to optimize the number of instances efficiently, particularly during sudden and significant increases/decreases in traffic. (2) the time required to initiate and make a container available to handle new requests, where even containers with low startup times compared to virtual machines (POTDAR et al., 2020b) may impact performance during sudden traffic increases. (3) the reactive approach relies on thresholds, generating multiple instances with moderate resource usage. In contrast, predictive autoscaling leverages past resource usage to forecast future resource demand, resulting in fewer instances operating within a medium to maximum usage of resources. Consequently, predictive autoscaling tends to utilize resources more effectively.

In conclusion, when dealing with familiar and trained network traffic patterns, predictive and hybrid autoscaling approaches demonstrate a clear advantage over the reactive approach, outperforming it. As highlighted in Table 13, predictive and hybrid models introduce almost zero errors, underscoring their robust predictive capabilities. This prompts the question: "Why not exclusively use the predictive autoscaling mechanism in RAFE?". The reason is that even applications with well-defined standards, consistently receiving the same types and quantities of requests, are susceptible to variations in these patterns (especially NFVs and MEC applications situated at the network's edge, prone to encountering diverse workflow patterns). Consequently, an autoscaling framework must be able to adapt to unforeseen traffic patterns, ensuring the maintenance of quality of service without compromising performance.

5.4 Handling Unknown Traffic Pattern (Performance Results)

The second experiment, in turn, is designed to evaluate RAFE under the distinctive conditions inherent to the mutable nature of the network, where the traffic workload pattern cannot remain the same. We evaluate how each of RAFE's autoscaling approaches adeptly manages unforeseen, unlearned, and unexpected traffic patterns. This validation is less commonly explored in current state-of-the-art works, however, it is a critical aspect, particularly in the context of edge networks, encompassing IoT, VNF, and MEC. Unlike conventional network scenarios, these contexts are uniquely influenced by the ebb and flow of human activity in the region where these applications operate. Factors such as human migrations and events significantly impact the traffic patterns in these scenarios. Therefore, a comprehensive understanding of RAFE's adaptability to such dynamic and unpredictable conditions is essential for ensuring its effectiveness in real-world edge scenarios.

To this end, we conducted additional tests utilizing the same testbed and the previously trained model derived from the initial experiment. In this iteration, we modified the traffic configuration to conform to a newly defined pattern as depicted in Fig. 26b. Upon comparison with the initial traffic pattern presented in Fig. 26a, it becomes evident that while the two patterns share similarities, they simulate slight variations. This deliberate adjustment aims to assess how the mechanisms respond to different yet closely related patterns, emphasizing a pronounced contrast at the center of the pattern. This is designed to simulate an unexpected surge in traffic, challenging the system's capacity to adapt to abrupt changes in the learned patterns.

Notably, for this experiment, the revalidation mechanism of RAFE's model was intentionally disabled. This deliberate choice allows us to specifically gauge the system's resilience and adaptability under conditions where the model is not revalidated, thereby providing insights into its performance in scenarios with unanticipated traffic variations.

5.4.1 Reactive Autoscaling

In this second experiment, the RAFE's Reactive Autoscaling module achieved the results presented by Fig. 31. The blue line presents the number of instances/containers the reactive module instantiates over time. The horizontal bars in blue represent the number of request errors over time. Additionally, to reference the workload for the comparison, the yellow dashed line represents the number of VUs (or approximately the requests per second) over time for the period.

Furthermore, to complement the visual presentation of results and to provide a quantitative perspective on network request errors, Table 14 furnishes data on the total simulated requests, the instances of request errors encountered during the experiment, and the corresponding error percentage relative to the total request amount.

Table 14 – Requests and errors during the second experiment using the reactive method.

Requests	**Errors**	**Errors (%)**
148502	942	0,63433%

Assessing the performance of the predictive module, now handling unknown traffic patterns, some key observations come to light:

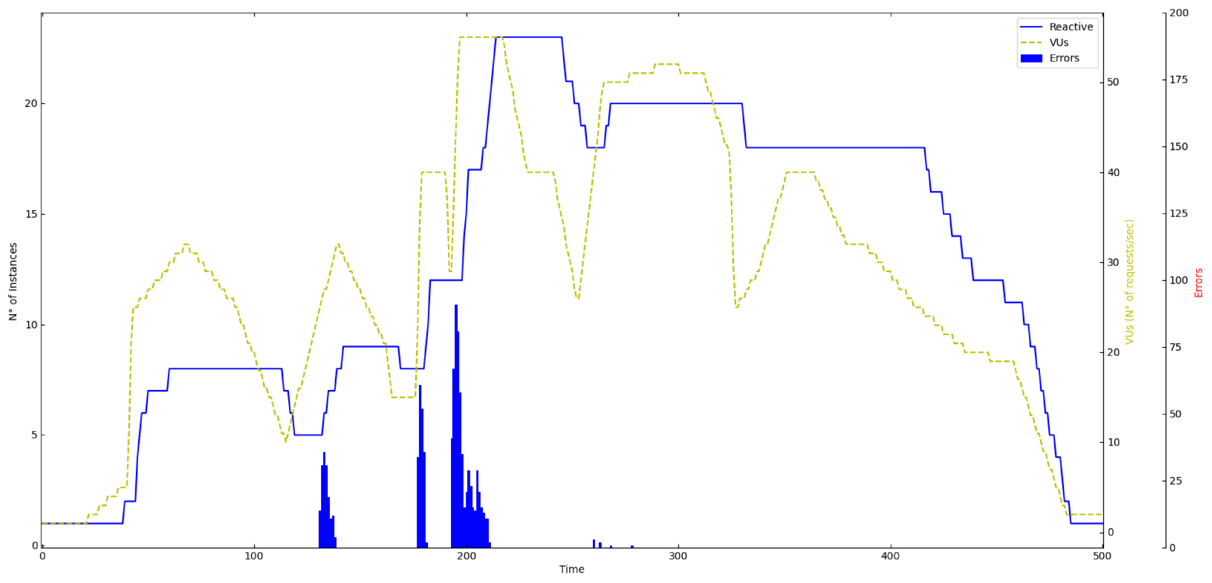


Figure 31 – Reactive Autoscaling module results, during the second experiment, handling the traffic pattern defined in Fig. 26b.

- ❏ The Reactive Module demonstrated good performance, particularly in response to changes in traffic, compared to its performance in the first experiment. Notably, it excelled during unexpected traffic behavior, especially in the time interval between 250 and 320 points on the timeline. However, this success is attributed to the inherent delay in scaling down instances, as observed in the first experiment with this reactive approach. Consequently, there was a sustained high instance count during the elevated traffic period from 200 to 230. This delayed scaling down allowed more effective handling of increased traffic near point 255, as more instances were available to manage the surge.

- ❏ Analysis of errors, as presented in Table 14, and comparison with errors from the first experiment (Table 10) reveals a reduction in error rates in the second experiment, despite using the same module. However, this reduction can be attributed to the second experiment's unique traffic characteristics, which leverage the delay to scale down to achieve the results.

- ❏ Despite encountering higher peak loads during the second phase, it is essential to note a significant increase in the instance count compared to the first experiment, particularly during the second half. This led to elevated hardware resource costs and decreased overall efficiency.

5.4.2 Predictive Autoscaling

In this second experiment, the RAFE's Predictive Autoscaling module demonstrated its performance as depicted in Figure 32. The orange line in the graph illustrates the

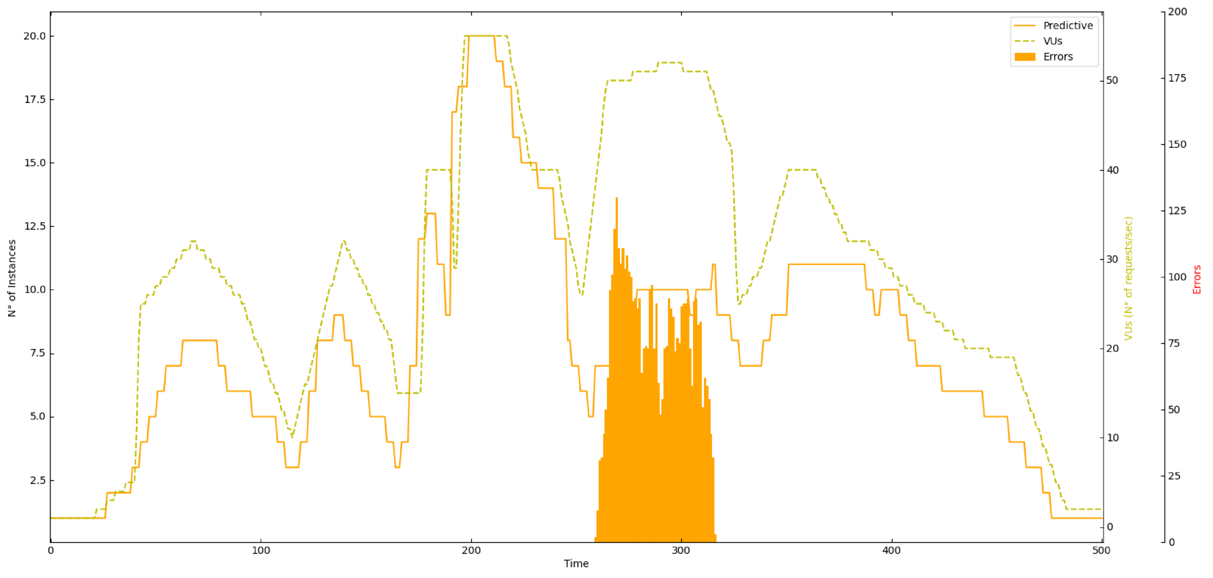


Figure 32 – Predictive Autoscaling module results, during the second experiment, handling the traffic pattern defined in Fig. 26b.

evolution of the number of instances or containers spawned by the predictive module over time. Concurrently, the horizontal orange bars (which, due to the short quantity, ended up not being present) correspond to the occurrences of request errors at various time points. To facilitate a workload reference for comparison, the yellow dashed line tracks the number of VU, or roughly the requests per second, simultaneously.

Furthermore, to enhance the comprehensibility of the results and provide a quantitative perspective on request errors, Table 15 furnishes data on the total simulated requests, the instances of request errors encountered during the experiment, and the corresponding error percentage relative to the total request volume.

Table 15 – Requests and errors during the second experiment using the predictive method.

Requests	**Errors**	**Errors (%)**
152680	4357	2,85368%

Evaluating the performance of RAFE's Predictive module, once handling unknown traffic patterns, it is to be noted:

- ❏ Examining the error axis (depicted by the orange bar) concerning the VUs or req/sec axis (indicated by the yellow dashed line) in Fig. 32, it is apparent that errors occurred precisely during the interval where an unprecedented traffic load was simulated, particularly in the vicinity of the 250-320 interval. This highlights a limitation of the predictive module when faced with roughly divergent unknown traffic patterns.

- ❏ Contrasting the error figures presented in Table 15 with those from the first experiment, as outlined in Table 11, reveals a noticeable increase in the number of errors.

This indicates a substantial decline in performance when solely relying on the predictive module to handle untrained traffic load patterns. Once again, this underscores a limitation of this module when confronted with unfamiliar traffic patterns.

5.4.3 Hybrid Autoscaling

The Hybrid Autoscaling module of RAFE, in turn, outcomed in this second experiment the results presented by Fig. 33. The blue line represents the number of instances/containers instantiated over time by the reactive module. The horizontal blue bars correspond to the number of request errors occurring over time. To provide a workload reference for comparison, the yellow dashed line represents the number of VUs or approximately the requests per second over the given time.

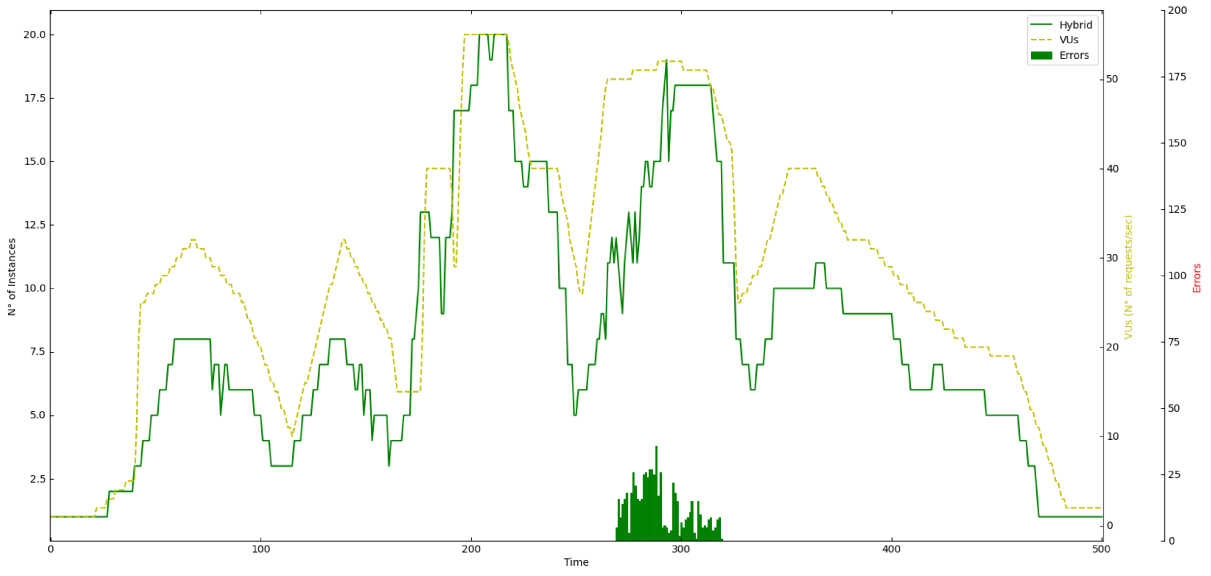


Figure 33 – Hybrid Autoscaling module results, during the second experiment, handling the traffic pattern defined in Fig. 26b.

Table 16 – Requests and errors during the second experiment using the hybrid method.

Requests	**Errors**	**Errors (%)**
148821	650	0,43677%

Moreover, to complement the result visualization and to give a numeric vision of the request errors, Table 16 presents the number of requests simulated during the experimentation, the number of request errors that occurred during this evaluation and the percentage that these errors represent over the amount of the requests.

Evaluating the performance of RAFE's hybrid module facing unfamiliar traffic patterns and comparing it with the previous modules, some items become apparent:

❏ Similar to the observations made in the first experiment, as depicted in Fig. 33, the hybrid module also exhibited fluctuations in the number of instances, indicative of rapid shifts in traffic. These fluctuations resulted from conflicts arising between the actions of the Predictive Module and the Reactive Module. Such conflicts occurred when one module initiated a scaling action that contradicted the scaling decision of the other module.

❏ Notably, when comparing the Reactive and Predictive Modules results in this second experiment, the hybrid module's performance stands out, showcasing fewer errors than both modules and demonstrating a closer alignment of the number of instances with the simulated traffic load over time. This indicates enhanced cost-effectiveness and performance.

❏ This module's better performance and cost-effectiveness can be attributed to the complementary relationship between the Reactive and Predictive Modules. During untrained/unknown traffic load patterns, the Reactive Module proactively mitigates issues caused by inaccurate predictions from the Predictive Module. Meanwhile, during periods of stable traffic, the Predictive Module maintains cost-effectiveness by instantiating the optimal number of instances in anticipation of traffic changes.

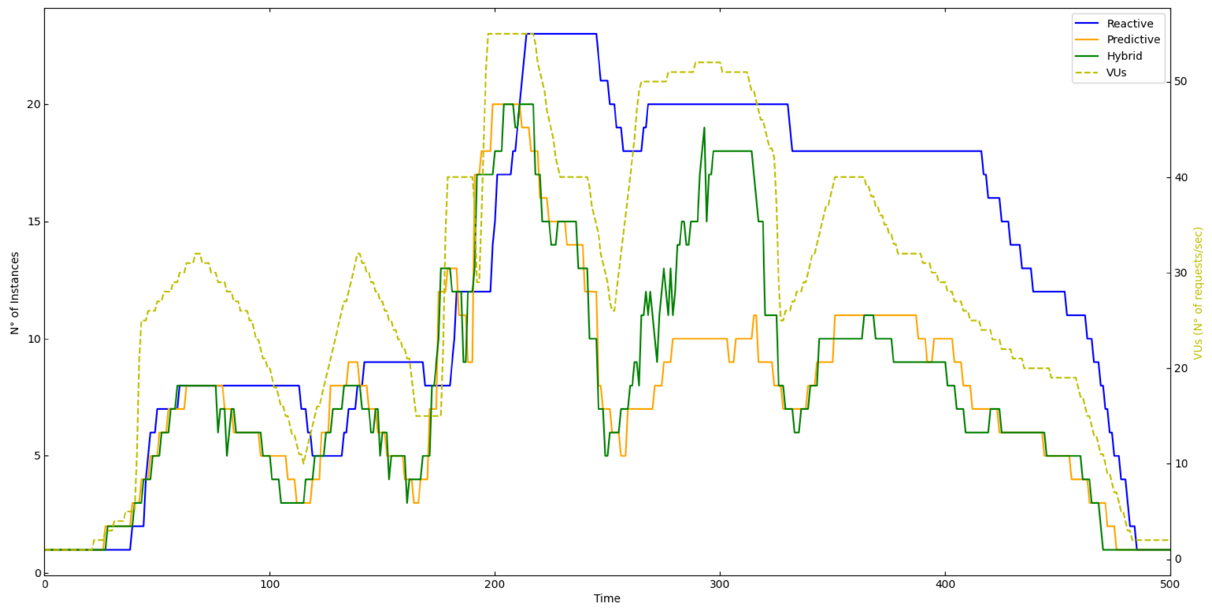
5.4.4 Experiment Analysis

Table 17 – Requests and errors in each autoscaling mechanism during the second experiment.

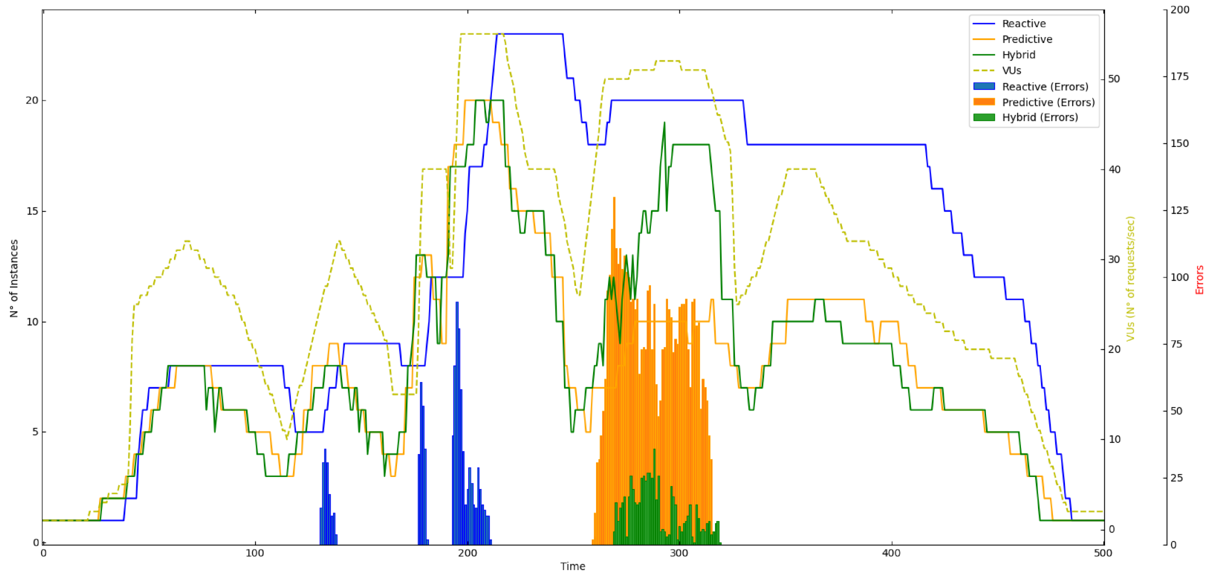
	Requests	**Errors**	**Errors (%)**
Reactive	148502	942	0,63433%
Predictive	152680	4357	2,85368%
Hybrid	148821	650	0,43677%

Analyzing the comprehensive results of the second experiment, Figure 34 and Table 17 provide a detailed account, following the same structure as in Fig. 30 and Table 13, of the performance of each autoscaling mechanism (reactive, predictive, and hybrid) implemented by the RAFE framework. Analyzing these results, particularly in instances where the simulated traffic closely resembles the initial traffic load, it is noteworthy that all algorithms react similarly, yielding results comparable to those achieved in the first experiment when confronted with minor variations in known traffic patterns. This reiterates the advantage and effectiveness of predictive and hybrid approaches in handling familiar traffic patterns, even with some variation.

Conversely, as depicted in samples 260 to 310 of Figure 34b, when faced with unfamiliar/untrained traffic workload patterns that significantly differ from the known ones,



(a) Comparison between the autoscaling mechanisms results in the second experiment.



(b) Comparison between the errors during each mechanism evaluation in the second experiment.

Figure 34 – Comparison between reactive autoscaling, predictive autoscaling, and both of them (hybrid), handling the traffic pattern defined in Fig. 26b. The VUs or requests per second are also presented to offer a reference for comparison.

the forecasting approaches tend to exhibit suboptimal results. In this scenario, predictive autoscaling demonstrates substantial shortcomings, causing nearly half of the simulated requests in this period to fail, thereby compromising the QoS. In the same scenario, while not always utilizing the most efficient number of containers/instances, reactive autoscaling effectively manages the unseen traffic pattern without compromising overall efficiency and QoS.

Moreover, in line with the observations from Figure 30, the hybrid autoscaling mechanism (green line and bar) emerges as the most robust overall performer. Despite encountering challenges with unseen patterns, it demonstrates the fewest request errors (as indicated in Table 17) and maintains cost-efficiency by allocating optimal resources to meet demands. Combining results from both experiments, the hybrid approach of predictive and reactive autoscaling enables RAFE to sustain resource efficiency without significant performance impact (or with minimal impact) and respond more effectively to intensive changes in known traffic patterns than relying solely on predictive scaling. This provides cost-efficiency and ensures the target applications' good QoS.

However, a critical observation arises in pursuit of a high service standard: the hybrid autoscaling mechanism presents expressive overall performance but falls short in handling unforeseen or untrained traffic patterns. This limitation predominantly arises from the activation of the predictive mechanism, which, when unfamiliar with a specific pattern, erroneously anticipates a lower resource demand, leading to incorrect scaling down. This results in the oscillations depicted in Figure 34b (samples 270-310) and impacts the overall performance.

This fact underscores the imperative need for a dynamic pattern update or invalidation mechanism, which can leverage the multiple AI models to carry out these adjustments selectively based on the specific metrics.

5.5 Model Suspense and Overall Performance Results

Recognizing the limitations exposed in the previous experiment, where the hybrid auto-scaling mechanism demonstrated a notable performance shortfall in the face of unforeseen or untrained traffic patterns, the RAFE's model suspense and invalidation mechanism, alternatively referred to as the model revalidation mechanism, takes center stage. We assess RAFE's performance in this third experiment when handling unexpected/unlearned/unseen traffic patterns. However, unlike the second experiment, we have enabled RAFE's model revalidation mechanism this time to validate the RAFE's performance with all of its capabilities enabled. This approach also lets us ascertain whether a hybrid strategy incorporating the suspense/revalidation mechanism can yield optimal results in the auto-scaling scenario.

To this end, we have trained a new model with the same process depicted in the first experiment and simulated the changed traffic defined in Fig. 26b, as in the second experiment. However, to leverage RAFE's capabilities this time, we used the hybrid approach with the model revalidation mechanism activated. In addition to the RAFE's configurations listed in section 5.2, we have configured the revalidation mechanism with the following settings:

❏ Monitoring the predicted instances and actual instantiated containers every 10 seconds.

- ❏ Evaluating the deep learning-based predictive model at 5-minute intervals.
- ❏ Utilizing 30 data points for evaluation by combining 10-second monitoring intervals to effectively span the past 5 minutes of data.
- ❏ A threshold of 1.6 MAE to trigger the suspension of the predictive model.

5.5.1 Experiment Analysis

As a result, the RAFE's Hybrid Autoscaling module, enhanced by its model revalidation mechanism, achieved the performance depicted in Fig. 35. In this figure, the brown line illustrates the count of instances/containers over time, while the black line represents the number of instances predicted by the predictive module. The juxtaposition of both lines allows for a direct comparison between the predicted and actual instances. A horizontal red bar graphically displays the occurrence of request errors over time. The light blue marker denotes the suspension interval of the predictive model and mechanism. The yellow dashed line portrays the number of VUs or approximate requests per second over the specified period to provide a workload reference for comparison.

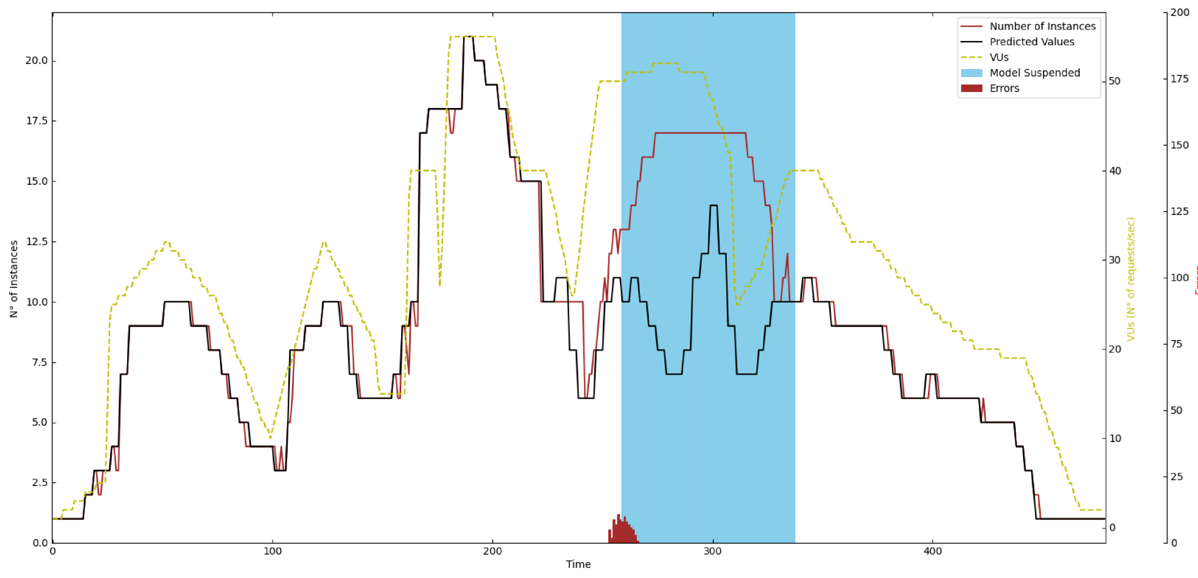


Figure 35 – Hybrid Autoscaling module with model revalidation mechanism results, during the third experiment, handling the traffic pattern defined in Fig. 26b.

Furthermore, to enhance the clarity of our results and provide a quantitative overview of request errors, Table 18 presents the following data points: the total number of requests simulated during the experimentation, the count of request errors that occurred in this evaluation, and the corresponding percentage, expressing the proportion of errors relative to the total number of requests.

Table 18 – Requests and errors during the third experiment using the hybrid method with model revalidation mechanism.

Requests	**Errors**	**Errors (%)**
148116	91	0.06144%

Evaluating the performance of RAFE, leveraging all of its capabilities, with the hybrid auto-scaling approach enhanced by the model revalidation (model suspense) mechanism, several key observations come to light:

- ❏ The hybrid module, enhanced by the revalidation mechanism, demonstrates superior performance when addressing unforeseen or unlearned traffic patterns. This is affirmed by the significant reduction in error rates, as illustrated in Table 18, where the difference with other approaches is striking. In managing such traffic patterns, it achieves an error rate of less than 0.1%, showcasing its exceptional efficacy.
- ❏ In the time frame delineated by the light blue range in Figure 35, a significant shift in traffic to an unfamiliar and distinctly different pattern triggers the suspension of the model. Just before this point, we observe a notable disparity between the predicted instance count and the actual instances, accompanied by a few errors. During this juncture, the reactive auto-scaling mechanism responds to the unlearned pattern, exacerbating the discrepancy between predicted and actual instances. This discrepancy is evaluated by the model revalidation mechanism, leading to the suspension of the predictive auto-scaling mechanism. As a result, the reactive module takes over, working independently to optimize its performance. This underscores the effectiveness of RAFE's hybrid approach, leveraging its capabilities, even in scenarios involving unknown or unexpected traffic patterns.
- ❏ A comparison between RAFE's hybrid auto-scaling approach with and without the model revalidation mechanism can be made by contrasting the current experiment results with the one predominantly featured in Section 5.4.3. Upon analyzing these results, it becomes evident that the primary challenge posed by an unlearned and significantly different traffic pattern was effectively addressed by including the revalidation mechanism.

RAFE has exhibited expressive performance, consistently delivering optimal results across a variety of scenarios commonly encountered in IoT, VNFs, MEC, and cloud environments. The predictive auto-scaling mechanism within RAFE, powered by deep learning models, emerges as a key factor contributing to these accomplishments. However, such forecasting strategies rely on historical data, and a challenge arises when confronting unknown/untrained patterns that substantially differ from learned patterns. To tackle this issue, RAFE employs a revalidation mechanism capable of suspending the model and, if necessary, requesting a new model training session when the predictive model's performance deteriorates. In conjunction with the hybrid approach that combines predictive and reactive modules, this mechanism is a critical element in achieving optimal results in terms of QoS and cost-effectiveness.

These three experiments yielded great results in terms of performance for RAFE. However, it's to be noted that in none of these experiments were network errors eliminated. This is primarily attributed to the simulated traffic pattern's nature, characterized by numerous rapid traffic peaks and as a consequence of the configurated thresholds and metrics, which aimed to strike a balance between cost-efficiency and QoS, rather than prioritizing a zero-error outcome. To attain a zero-error result, network or application operators/developers can utilize the Config Module provided by RAFE (as depicted in section 3.3.2) to fine-tune the thresholds with an emphasis on optimizing QoS. Conversely, the same module can be adjusted to focus on achieving the most cost-effective outcomes, depending on the specific objectives and priorities of the operator.

5.6 Invalidation of Forecasting Models

Network traffic patterns and the associated hardware resource consumption can exhibit high levels of dynamism, with abrupt spikes or declines influenced by multiple factors. For instance, consider an MEC application deployed on a base station, where the traffic pattern can be dramatically shaped by events that draw large crowds or shifts in population, resulting in a markedly different traffic pattern than what was used for training predictive models. This inherent variability is a characteristic of networks, and RAFE was designed to address such scenarios.

To tackle this challenge, as previously elucidated, RAFE incorporates a mechanism known as the "model invalidation mechanism" in the revalidation context. This mechanism identifies and subsequently invalidates a pre-trained deep learning model when it detects a significant deterioration in the model's performance, thus triggering a new model training process.

To evaluate the effectiveness of this mechanism and assess RAFE's ability to adapt by learning new patterns on demand (when these patterns are identified as recurrent), we experimented within the same scenario proposed in the model suspense evaluation (third experiment, Section 5.5). We configured a model invalidation threshold to be triggered after five occurrences of model suspense. Upon triggering, a series of sequential steps ensued: a new request for model training is made and added to the training queue, a dedicated worker processes the training request, a new training process is conducted, a fresh model is created and updated within RAFE's core application, and finally, the predictive model is reactivated with the newly trained pattern.

5.6.1 Experiment Analysis

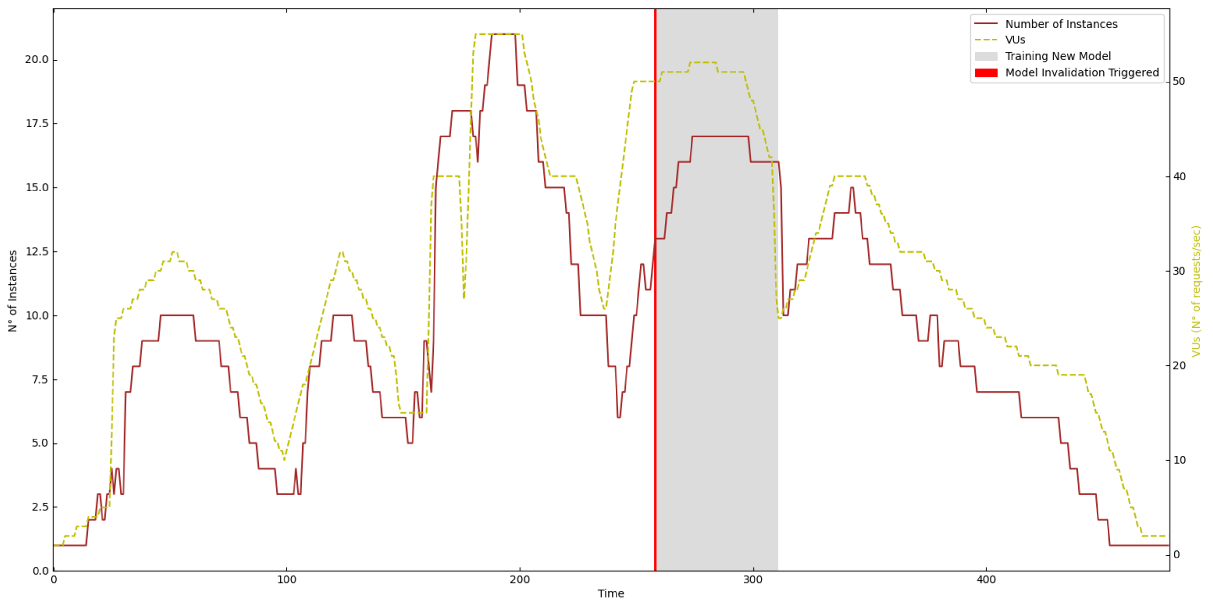


Figure 36 – Revalidation model triggering model invalidation and requesting a new model.

As a result, after the model suspense request was triggered five times, the model revalidation mechanism, instead of suspending the model, proceeded to invalidate it. This scenario is illustrated in Figure 36. In this representation, the brown line depicts the number of instances, and the dashed yellow line represents the number of VUs, maintaining consistency with previous examples. Furthermore, the red bar indicates when the revalidation mechanism triggers model invalidation. At the same time, the grey area signifies the duration during which the predictive module is disabled, and the training process takes place. In this specific case, the training process required approximately 13 minutes. Subsequently, a new model trained with the updated traffic pattern is seamlessly integrated into the predictive module. As evident, the predictive module was reactivated immediately after this period, and the number of instances was adjusted following the newly predicted values.

Following the invalidation of the old model and the generation of a new one, the hybrid module exhibited the performance illustrated in Figure 37. In this representation, the brown line tracks the number of instances, the dashed yellow line reflects the count of VUs, and the red bar denotes the number of errors (discernible due to the minimal error count). Throughout this experimentation, 129,889 requests were executed, with only 3 remaining unfulfilled, constituting approximately 0.00231%. An insightful analysis of this evaluation reveals that CAPE successfully learned the new traffic pattern, showcasing remarkable adaptability. Moreover, the invalidation module demonstrated precision in evaluating and deciding, based on the threshold, whether the newly observed traffic pattern is a transient anomaly (potentially caused by a one-time event inducing higher-than-normal hardware consumption of a VNF) or a genuinely new pattern necessitating learning (requiring the

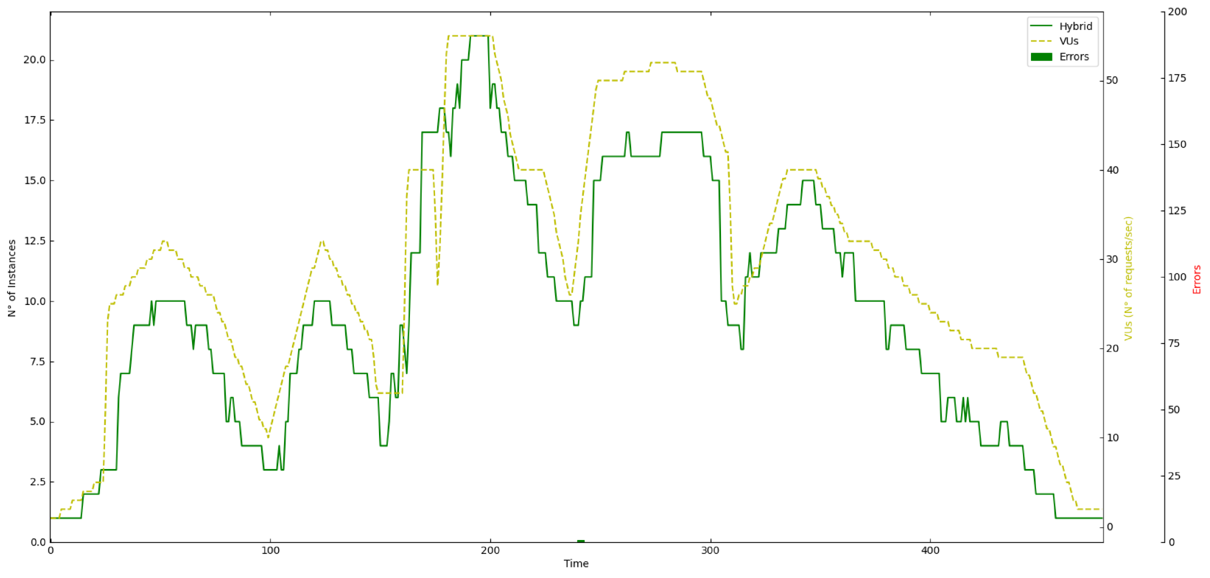


Figure 37 – Hybrid Autoscaling module performance results with the new model trained with the new pattern after the first model invalidation.

creation of a new model). Notably, the low model training time, 13 minutes, was a key factor for the effectiveness of the approach. This parameter is evaluated in the following section.

5.7 AI-Based Models Training Time

RAFE incorporates suspension and revalidation mechanisms, alongside a multi-AI models approach (creating an AI-based univariate model for each monitored metric), to achieve optimal resource allocation results for IoT, NFV, MEC, or cloud applications. Nevertheless, this approach involves multiple deep learning model training processes, which are both time-consuming and resource-intensive. Additionally, since the revalidation mechanism involves replacing the old model, training a new model is imperative to ensure optimal performance swiftly.

Hence, minimizing the model training time is of the greatest importance for this approach. A shorter training time allows the predictive mechanism to resume operation promptly after an invalidation. Moreover, a reduced training time translates to fewer allocated hardware resources and, consequently, lower costs, leading to a more cost-effective solution.

To assess the training time of RAFE's forecasting models, we conducted ten iterations of the training process, utilizing the metrics generated during the evaluations in Sections 5.3, 5.4, and 5.5. The same hardware described in section 5.2 was utilized for this assessment. The resulting mean training times for each model/metric (one for each configured monitored metric in the aforementioned evaluations) are presented in Table 19.

As presented, the model's training for learning CPU patterns required an average

Table 19 – Time spent to train the AI-based forecasting models used in the Predictive Module.

Model's Metric	**Mean Training Time**
CPU	12 min 03 sec (±2 min 34 sec)
MEM	14 min 26 sec (±3 min 12 sec)

training time of approximately 12 minutes, while the models used to understand MEM patterns necessitated an average training time of around 14 minutes. A 90% confidence interval accompanied both of these values.

Achieving such low training times, as evidenced by a maximum of 18 minutes, was of essential significance for our objectives, especially when combined with RAFE's distributed architecture, where these training procedures can be executed concurrently. This efficiency in training time can be attributed to several key factors: (i) the univariate nature of the datasets, which possess only one dimension, resulting in reduced computational resource requirements; (ii) the simplified data structure necessitating fewer layers and nodes in the neural network to achieve satisfactory performance, thus directly impacting the processing time for NN inputs, and (iii) the effectiveness of random search hyperparameter tuning, when combined with the less complex data structure, in generating favorable outcomes within a limited number of iterations (CASTAN-LASCORZ et al., 2022).

5.8 Integration With Other MANO Solutions

To assess RAFE's suitability for network-wide use and its compatibility with existing systems, we have integrated RAFE into a widely adopted stack within NFV and MEC environments, aligning with the ETSI's NFV Reference Architecture (DAHMEN-LHUISSIER, 2021). This evaluation gauges RAFE's adaptability and utility in various network scenarios. Moreover, it serves as a valuable reference for future endeavors. The proposed integration leverages established technologies commonly utilized in this context, as illustrated in Figure 38.

As depicted in this figure, the assembly of elements within ETSI's NFV architecture involved the selection of the following technologies:

MANO: For the crucial role of orchestrator and manager, we utilized the Open Source MANO (OSM) (OSM, 2020a). OSM is one of the market's most widely adopted technologies in these capacities. It assumes the responsibility of supervising and overseeing all entities within the architecture, maintaining a comprehensive awareness of each architectural block's usage, operational status, and utilization statistics (YU; YANG; FUNG, 2020).

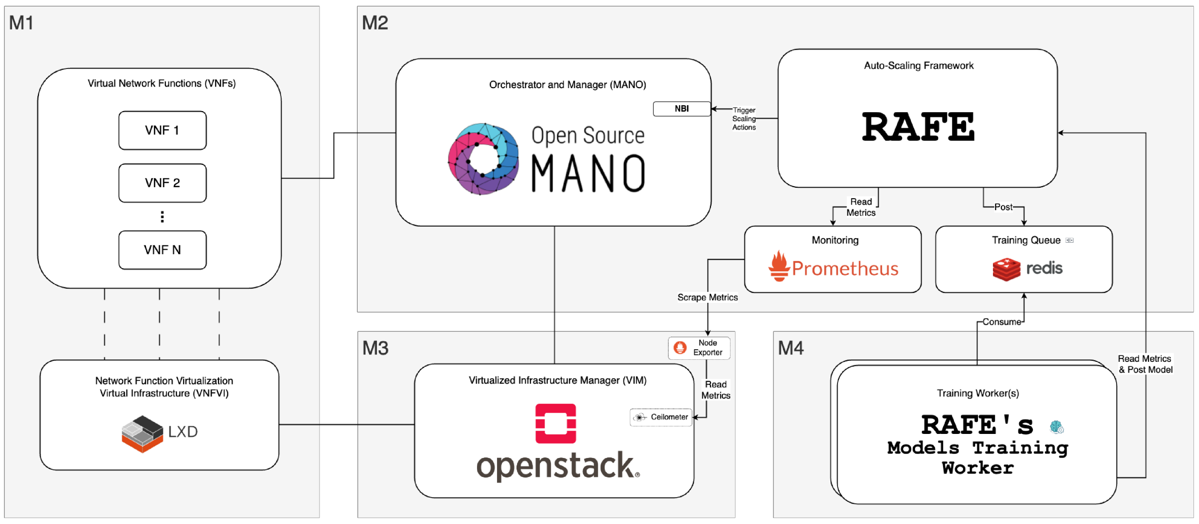


Figure 38 – ETSI's NFV Reference Architecture Integration

VIM: To fulfill the role of Virtualized Infrastructure Manager (VIM) within this architecture, we selected OpenStack (FOUNDATION, 2010). Being one of the most prevalent VIM-like infrastructure managers, OpenStack proved a solid choice for this task.

VNFVI: To provide the necessary virtual hardware for hosted VNFs, we integrated LXD (CANONICAL, 2017) into the virtualization layer. LXD is an open-source container management extension for Linux Containers (LXC). This choice facilitated seamless integration with the OpenStack component and enabled the execution of VNFs in containers. This, in turn, reduces the hardware resource requirements and accelerates the deployment and scaling processes (BHARDWAJ; KRISHNA, 2021).

With this VNF enabler architecture defined, we proceeded with the RAFE integration to deliver auto-scaling services. This integration involved two primary steps. The first step entailed the creation of an adapter designed to communicate with OSM and initiate scaling actions. As outlined in section 3.3.2 (under the MANO Interactor section), this adapter exposes four essential functions: scale-in, scale-out, scale-to, and a function to retrieve the current number of instances. To implement this adapter and ensure seamless integration with OSM, we leveraged the NBI provided by OSM, which offers HTTP services for managing. Further details regarding OSM's exposed NBI methods can be found in ETSI (2016b).

To serve as the data source for monitoring, retrieving, and storing the historical hardware resource consumption of VNFs, we have implemented a Prometheus instance. We've created a Node Exporter instance on the OpenStack host to populate the Prometheus database. This Node Exporter is responsible for gathering metrics from OpenStack's services and making them available for Prometheus scraping. OpenStack relies on Ceilometer (OPENSTACK, 2022) to monitor and collect data from its various services, including the compute service that encompasses both the VNFI sources and the VNFs themselves. Consequently, the Node Exporter captures these collected telemetry metrics and exposes them for scrapped by Prometheus.

To enable the predictive and hybrid mechanisms, we also implemented RAFE's worker stack, which comprises a Redis instance to work as a queue and one or more RAFE's Model Training Workers. Notably, when comparing this integration with an NFV enabler architecture to the testbed architecture used in previous evaluations, the single modification made in RAFE was the addition of a plugin designed to interface with OSM, incorporating just four straightforward methods. This straightforward adaptation underscores the ease of integrating RAFE into new architectural stacks, requiring only the implementation of two new plugins, a MANO Interactor, and a Metric Reader. This demonstrates the flexibility and simplicity of the integration process, making it compatible with various architectural setups.

In the context of the RAFE assessment, the wrappers labeled as $M1$ to $M4$ illustrated in Fig. 38, represent distinct host machines. In our implementation, these machines correspond to virtual machines hosted within the Compute Engine service on GCP (GOOGLE, 2023b). During our tests, RAFE's core presented minimal hardware resource consumption, emphasizing that it can safely co-deployed with OSM in a single instance. However, deploying RAFE's model training workers in a distributed fashion is advantageous, as deep learning model training is a resource-intensive task. Furthermore, we positioned the Prometheus instance on the same host as RAFE to minimize the time required for metric retrieval. Nonetheless, deploying Prometheus separately and establishing a network link between the hosts for communication is possible. Regarding allocated resources, hosts $M1$, $M3$, and $M4$ are equipped with 2 vCPUs and 4GB of memory. In comparison, host $M2$ boasts 2 vCPUs and 6GB of memory, meeting the minimum recommended specifications for hosting OSM effectively.

5.8.1 Experiment Results

Utilizing the methodology outlined in the initial experiment of this work (section 5.3), we simulated traffic over 2 days throughout the architecture, enabling only the predictive module in RAFE to generate historical metrics. Then, we enabled the predictive module, which triggered a new model training and utilized the metrics as a dataset. The entire procedure adhered to the aforementioned architecture, incorporating a distributed training process, automatically initiating training processes triggered by RAFE.

To evaluate RAFE's results, comparing them with those observed in the initial experiments, we simulated the identical traffic pattern utilized previously, as delineated in Figure 26a. This approach enables a comprehensive assessment of RAFE's performance across various environments within the workflow. Moreover, we experimented using the same configuration and traffic load described in the first experiment, presented in section 5.3.

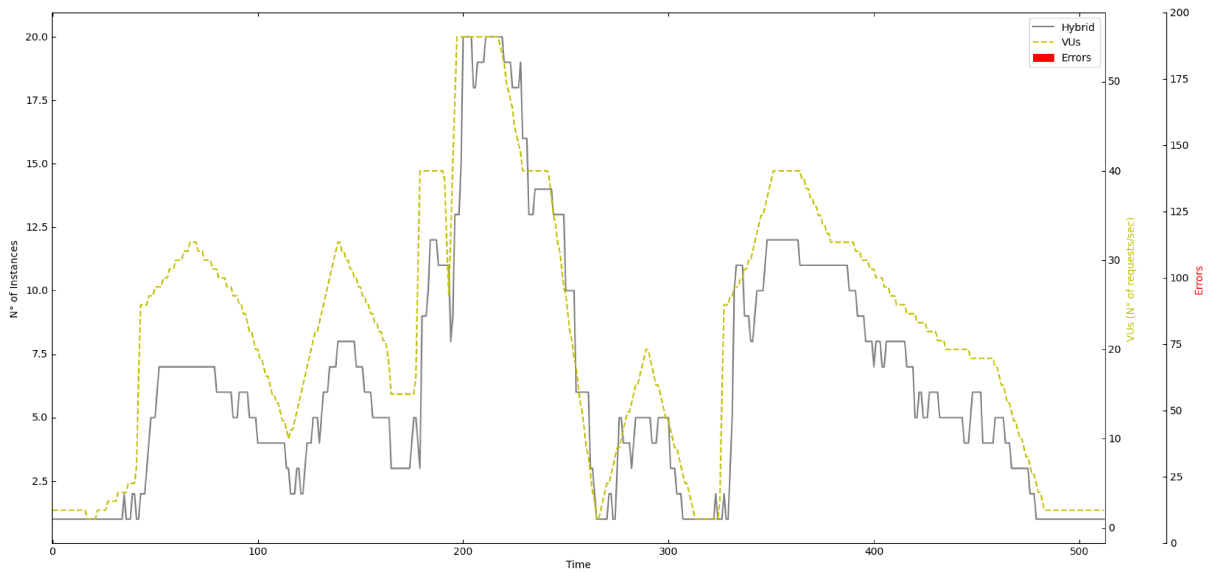


Figure 39 – Hybrid Autoscaling module results, with RAFE integrated with OSM and OpenStack, handling the traffic pattern defined in Fig. 26a.

As an outcome, Figure 39 showcases the performance of RAFE's hybrid autoscaling module when integrated into an architecture based on ETSI's NFV Reference Architecture, utilizing OSM and OpenStack as key components. The grey line illustrates the hybrid module's dynamic instantiation of instances/containers over time. The red horizontal bars denote the occurrence of request errors over time (expected to be minimal, rendering them almost imperceptible). For contextual workload comparison, the yellow dashed line represents the variation in VUs or approximate requests per second throughout the specified period.

Table 20 – Requests and errors during the first experiment using the hybrid method.

Requests	**Errors**	**Errors (%)**
112832	11	0.00974%

In addition to the graphical representation of results, Table 20 provides a quantitative overview of request errors. The table includes details on the total number of requests simulated during the experiment, the count of request errors recorded, and the corresponding percentage of these errors represented concerning the overall volume of requests. This tabular presentation enhances the clarity of the evaluation by offering a numeric perspective on the occurrence and impact of request errors.

Evaluating the performance of RAFE integrated with this commonly used NFV and MEC enabler architecture, some key points can be highlighted:

- It demonstrates results analogous to those presented in the Docker-based experiments, encompassing error rates and the observed instance number patterns. This underscores RAFE's adaptability without compromising performance when integrated into more complex and distributed architectures, showcasing expressive performance under such conditions.

❑ A minor delay in scaling new instances, as depicted in the chart (Fig. 39), is noticeable when compared to results from the Docker-based testbed with the same traffic load and configurations (depicted in Fig. 29). This delay is attributed to the time required for communication between MANO and VIM, along with readiness evaluations and health checks, which inherently consume more time in comparison to the testbed described in Section 5.2. Despite this, the execution time for scaling actions remains relatively short, primarily due to the concurrent execution of scaling actions adopted in RAFE, a key factor in responding promptly to traffic changes.

❑ In an overall evaluation of RAFE's integration capabilities with other widely used network technologies in different places of the network, it consistently demonstrates satisfactory effectiveness across different stacks, yielding similar outcomes in both scenarios.

5.9 Overall Analysis

Table 21 highlights a comparative analysis of the diverse solutions identified within the state-of-the-art literature, visually describing the aforementioned works. Several aspects were examined for each proposal. First, the solution's category can be Threshold-based Rules, Reinforcement Learning, Control Theory, Time Series Analysis, or Policies. Second and third, if the works provide or not auto-scaling reactive and prediction mechanisms. Fourth, the work category, as a Research Contribution (work only proposes a concept or idea, without practical implementation), a Research Project (work implements and validates an auto-scaling solution restrained to a specific scenario, not applicable to the market), or an Autoscaling Tool (work provides a full functional auto-scaling tool). Fifth, the main technologies applied in the solution. Sixth, if the work has mechanisms to handle changes in workload patterns, such as new training in predefined intervals, continuous learning or RAFE's revalidation module. Seventh, for cases where the solution provides a predictive approach based on AI models, this column shows the utilized approach, which can be a single model or, in the case of this work, a multi-model approach. Eighth, the solution's architecture, if it's a monolithic or distributed. And finally, eighth, where the target applications are placed in the network.

As illustrated in Table 21, RAFE advances the state of the art by introducing and evaluating an auto-scaling framework designed to accommodate applications deployed across diverse network locations. Its efficacy is demonstrated by successfully scaling VNFs, MEC, and cloud-placed applications. The framework's architectural flexibility allows seamless integration with various applications, owing to its conceptual robustness and generality. This integration extends to multiple monitoring tools and managers through plugins.

Distinguishing itself from some existing works that adopt either reactive or proactive auto-scaling methodologies, RAFE adopts a hybrid approach. This approach aims to ensure system stability while simultaneously anticipating future workload demands. The framework employs configurable threshold-based policies to prevent failures and leverages a proactive auto-scaling mechanism developed around a BI-LSTM time series forecasting model. This proactive approach aims to reduce the response time to network traffic changes.

Like other works in the field, RAFE analyzes resource usage metrics such as CPU and memory to dynamically adjust the allocated resources based on workload demand. Notably, RAFE does not limit itself to a specific monitoring stack or technology, allowing seamless integration with different monitoring tools through plugins. It only requires a time-series database (e.g., PrometheusDB, VictoriaMetrics) or a service that returns monitored metrics.

In addition, recognizing the dynamic nature of networks, we introduce a revalidation mechanism for RAFE. This mechanism is designed to adeptly manage shifts in workflow patterns, ensuring optimal performance for both observed and unforeseen patterns. Moreover, it facilitates the on-demand request for new models when there are changes in workload, thus ensuring adaptability to evolving network conditions.

Furthermore, a key contribution of this work lies in its exploration of multiple AI models, each dedicated to a specific metric. This approach contrasts with existing works that employ a single model for the entire target application or even a single model for all applications in service chains. This work also addresses architectural decisions and model training mechanisms to enhance the advantages and mitigate the drawbacks of this alternative approach. By implementing distributed training, scalability is achieved, accelerating model generation. This approach ensures that RAFE's core remains requiring few hardware resources, facilitating seamless integration and optimizing cost-efficiency.

Additionally, we introduce a Docker-based testbed implementation to substantiate the efficacy of our framework. This implementation generates traffic, oversees VNFs, and manages MEC applications. Essentially, it assumes the role of a MANO entity within the ETSI's VNF reference architecture. In assessing the performance of our framework, we gauged its response using the same pattern/dataset employed in the training process of our predictive approach. Notably, unlike most existing works, our validation process goes further. We conducted validation by simulating a distinct and previously unseen workload pattern. This comprehensive validation approach assesses how our proposed framework and its various auto-scaling mechanisms perform under significant variations in the learned network pattern.

Table 21 – State of the art summary.

	Approach	**Reactive Autoscaling**	**Predictive Autoscaling**	**Category**	**Enabling Technologies**	**React to Changes in Workload Pattern**	**AI Model Approach**	**Architecture**	**Applications Network Placing**
Han et al. (2012)	Threshold Rules	✓	○	Research Project	Static thresholds	-	-	Monolithic	Cloud / Core
Dutta, Taleb e Ksentini (2016)	Threshold Rules	✓	○	Research Project	Static thresholds	-	-	Monolithic	Edge
Arteaga, Risso e Rendon (2017)	Reinforcement Learning	✓	○	Research Project	Q-Learning and Gaussian Processes	-	-	Monolithic	Edge
Horovitz e Arian (2018)	Reinforcement Learning	✓	○	Research Project	Q-Learning	-	-	Monolithic	Cloud / Core
Lu, Yu e Pan (2022)	Reinforcement Learning	✓	○	Research Contribution	Semi-Markov Decision Process (SMDP)	-	-	-	Cloud / Core
Padala et al. (2009)	Control Theory	✓	○	Research Project	Control Theory	-	-	Monolithic	-
Kalyvianaki, Charalambous e Hand (2014)	Control Theory	✓	○	Research Contribution	Control Theory and Kalman Filter	-	-	-	-
Huang et al. (2016)	Queue Theory	✓	○	Research Project	Queue Theory	-	-	Monolithic	Cloud / Core
Shahin (2017)	Queue Theory	✓	○	Research Project	Queue Theory	-	-	Monolithic	Cloud / Core
Alawe et al. (2018)	Time Series Analysis	○	✓	Research Contribution	LSTM	-	Single Model	-	Edge
Moradi, Ahmadi e Nikbazm (2022)	Time Series Analysis	○	✓	Research Contribution	SVR, DT and KNN	-	Single Model	Monolithic	Edge
Kim et al. (2019b)	Time Series Analysis	○	✓	Research Contribution	CAT-LSTM	-	Single Model	-	Edge
Scalingi et al. (2019)	Time Series Analysis	○	✓	Research Contribution	GRU	-	Single Model	-	Edge
Zaman, Rahman e Naznin (2019)	Time Series Analysis	○	✓	Research Contribution	LSTM, CNN-LSTM and Bidirectional-LSTM	-	Single Model	-	Edge
Tao et al. (2021)	Time Series Analysis	○	✓	Research Project	LSTM	-	Single Model	Monolithic	Cloud / Core
EC2/ECS - AWS (2023), Scaling (2023)	Threshold Rules and Time Series Analysis	✓	✓	Autoscaling Tool	Static Thresholds and Proprietary AI-Based Algorithm	New training every 24h	Single Model	-	Cloud / Core
GKE - Google (2023b), Kubernetes (2023), Thompson (2019)	Threshold Rules and Time Series Analysis	✓	✓	Autoscaling Tool	Static Thresholds and Linear Regression or Holt-Winters Smoothing	New training every 24h	Single Model	-	Cloud / Core
OSM (2020b)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic / Distributed	Edge
Zafar et al. (2022)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Nicodemus, Boeres e Rebello (2020)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Vu et al. (2020)	Threshold Rules	✓	○	Autoscaling Tool	Static Thresholds	-	-	Monolithic	Edge
Al-Dhuraibi et al. (2017)	Policies	✓	○	Autoscaling Tool	MAPE-K principles	-	-	Monolithic	Edge
Bharanidharan, Jayalakshmi e Mayilvahanan (2022)	Time Series Analysis	○	✓	Research Project	LSTM and GRU	-	Single Model	Monolithic	Cloud
RAFE (this work)	Threshold Rules and Time Series Analysis	✓	✓	Autoscaling Tool	Static Thresholds and BI-LSTM	Revalidation On Demand	Multi Models	Distributed	Cloud / Core and Edge

CHAPTER 6

Conclusion

This work proposed and evaluated RAFE, a framework for enabling hybrid auto-scaling, with reactive and proactive mechanisms, that can be used across the network environments for NFV, MEC, and cloud applications. Additionally, it can be coupled to multiple monitoring and resource orchestration solutions.

RAFE employs threshold-based policies to prevent application failures under heavy workloads and enables predictive auto-scaling using multiple AI models based on BI-LSTM to forecast future resource demands. In addition, recognizing the dynamic nature of networks, RAFE introduces a revalidation mechanism to manage changes in workflow patterns adeptly, ensuring optimal performance for both observed and unforeseen patterns. This mechanism suspends and automatically requests new models when there are changes in workload, thus ensuring adaptability to network conditions. Furthermore, RAFE uses multiple AI models, each dedicated to a specific metric. This approach and RAFE's distributed training process enable more versatile vertical and horizontal scaling, speeding up the training processes and keeping the RAFE's core requiring fewer hardware resources, enabling better and easier integrability.

In the first part of this work, we evaluated and compared several machine learning-based forecasting algorithms. We assessed multiple solutions to validate their performance, aiming for an optimal solution for each forecasting category: univariate/multivariate, one-step/multi-step, and regression/classification. Our findings reveal that when confronted with diverse scenarios in an IoT/edge environment, no single algorithm can be a universal solution to achieve optimal results for all demands. However, our empirical results offer valuable guidance for selecting and composing a predictive approach in new projects based on specific necessities. Moreover, it serves as a base for choosing the predictive approach for RAFE, which uses a Bi-LSTM model and maps the forecasting problem as a univariate single-step regression problem.

In the second part of this work, we propose and evaluate the RAFE *(Resource Auto-scaling For Everything)* framework, designing and implementing a Docker-based orchestration testbed to integrate and evaluate it. Furthermore, we compare RAFE's reactive, predictive, and hybrid approach performances over different network scenarios. Additionally, we assess RAFE's integrability, connecting it to a commonly used ETSI's VNF architecture, RAFE's requested mean training time, and the effectiveness of RAFE's revalidation mechanism.

The findings demonstrate that RAFE effectively maintains system sustainability and optimizes application performance by offering bal, incorporating a model revalidation mechanism, the hybrid approach provisioning. Furthermore, RAFE exhibits remarkable integrability and performance, showcasing resilience even in the face of unexpected traffic shifts and significant changes in workload.

Additionally, the results highlight numerous advantages of using multiple AI models, each specifically designed to handle distinct metrics. This approach minimizes the time required for model training and ensures high-performance outcomes. It offers added flexibility in managing and treating these models individually. The distributed architecture, featuring a distributed queue mechanism for ML training, upholds the application's core as well-suited for diverse network environments ranging from edge to cloud-based systems.

Lastly, incorporating a model revalidation mechanism, the hybrid approach yields optimal performance across various scenarios. This is particularly evident in improving results when encountering untrained network traffic patterns and enhancing the overall adaptability of the solution.

6.1 Publications

This section presents publications directly related to this work. The first publication is the study about the most suitable machine-learning algorithms for MEC scenarios:

❏ VINHAL, L.; MOREIRA, R.; SILVA, FLÁVIO. A Comparative Analysis of Machine Learning Techniques for Enhanced Resource Management in Multi-access Edge Computing. **2023 IEEE 9th World Forum on Internet of Things (WF-IoT)**. out. 2023.

Also, there is a publication under preparation that presents RAFE and its hybrid approach for auto-scaling:

❏ VINHAL, L.; MOREIRA, R.; SILVA, FLÁVIO. Resource Auto-scaling For Everything (RAFE): A Smart Hybrid Auto-Scaling based on Machine Learning. **Expert Systems with Applications**. ISSN 0957-4174.

6.2 Future Work

The current study has provided valuable results and insights. However, several avenues for future research and development can enhance and extend the findings presented here:

- **Comparison With Other Frameworks:** Compare RAFE with the other state-of-the-art auto-scaling tools.
- **Dynamic Threshold Adaptation:** Explore adaptive threshold policies, dynamically adjusting the reactive mechanism based on performance metrics and real-time network conditions. This can lead to more responsive and efficient auto-scaling decisions in the reactive approach.
- **Real-World Deployment and Validation:** Deploy RAFE in real-world network environments to gather practical insights and validate its performance under diverse conditions. Collaborate with industry partners for large-scale deployments.
- **Auto-Configuration:** Research self-configuration mechanisms enable the framework to define its setting autonomously, thus reducing manual intervention.
- **Vertical Scaling:** Assess RAFE with horizontal and vertical scaling. The multi-model structure allows high configurability and granular scaling, which can be promising features for optimum vertical scaling.
- **Federated Learning:** Compare and assess RAFE's distributed learning with federated learning to evaluate the pros and cons of each approach.
- **Additional MANO Integrations:** Create more MANO integrations for RAFE, leveraging the plugin-based architecture to implement the connection with multiple MANO-like solutions.
- **Open-Source Community Engagement:** Consider releasing RAFE as an open-source project to foster collaboration, gather feedback, and accelerate its development. Engaging with the open-source community can lead to rapid advancements and adoption.

Bibliography

AALAM, Z.; KUMAR, V.; GOUR, S. A review paper on hypervisor and virtual machine security. **Journal of Physics: Conference Series**, IOP Publishing, v. 1950, n. 1, p. 012027, ago. 2021. Disponível em: <https://doi.org/10.1088/1742-6596/1950/1/012027>.

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Disponível em: <http://arxiv.org/abs/1603.04467>.

ABBAS, N. et al. Mobile edge computing: A survey. **IEEE Internet of Things Journal**, v. 5, n. 1, p. 450–465, 2018.

ADEGBIJA, T.; LYSECKY, R.; KUMAR, V. V. Right-provisioned iot edge computing: An overview. In: **Proceedings of the 2019 on Great Lakes Symposium on VLSI**. New York, NY, USA: Association for Computing Machinery, 2019. (GLSVLSI '19), p. 531–536. ISBN 9781450362528. Disponível em: <https://doi.org/10.1145/3299874.3319338>.

AHMAD, T.; CHEN, H. A review on machine learning forecasting growth trends and their real-time applications in different energy systems. **Sustainable Cities and Society**, v. 54, p. 102010, 2020. ISSN 2210-6707. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210670719335516>.

AHMED, N. K. et al. An empirical comparison of machine learning models for time series forecasting. **Econometric Reviews**, Informa UK Limited, v. 29, n. 5-6, p. 594–621, ago. 2010. Disponível em: <https://doi.org/10.1080/07474938.2010.481556>.

AL-DHURAIBI, Y. et al. Autonomic vertical elasticity of docker containers with elasticdocker. In: **2017 IEEE 10th International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2017. p. 472–479.

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys & Tutorials**, v. 17, n. 4, p. 2347–2376, 2015.

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.

ALI, B.; GREGORY, M. A.; LI, S. Multi-access edge computing architecture, data security and privacy: A review. **IEEE Access**, v. 9, p. 18706–18721, 2021.

ALSHARIF, M. H. et al. Green iot: A review and future research directions. **Symmetry**, v. 15, n. 3, 2023. ISSN 2073-8994. Disponível em: <https://www.mdpi.com/2073-8994/15/3/757>.

ALWAKEEL, A. M.; ALNAIM, A. K.; FERNANDEZ, E. B. A pattern for network function virtualization infrastructure (nfvi). In: **Proceedings of the 26th Conference on Pattern Languages of Programs**. [S.l.: s.n.], 2019. p. 1–9.

AN, N. H.; ANH, D. T. Comparison of strategies for multi-step-ahead prediction of time series using neural network. In: **2015 International Conference on Advanced Computing and Applications (ACOMP)**. [S.l.: s.n.], 2015. p. 142–149.

ANTONIADIS, A.; LAMBERT-LACROIX, S.; POGGI, J.-M. Random forests for global sensitivity analysis: A selective review. **Reliability Engineering & System Safety**, v. 206, p. 107312, 2021. ISSN 0951-8320. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0951832020308073>.

ANVITH, P. v. et al. A survey on network functions virtualization for telecom paradigm. In: **2019 TEQIP III Sponsored International Conference on Microwave Integrated Circuits, Photonics and Wireless Networks (IMICPW)**. [S.l.: s.n.], 2019. p. 302–306.

APSCHEDULER. **APScheduler**. 2012. Disponível em: <https://apscheduler.readthedocs.io>.

ARABNEJAD, H. et al. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: **2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)**. [S.l.: s.n.], 2017. p. 64–73.

ARTEAGA, C. H. T.; RISSO, F.; RENDON, O. M. C. An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc. In: **2017 13th International Conference on Network and Service Management (CNSM)**. [S.l.: s.n.], 2017. p. 1–7.

ATOUI, W. S. et al. Configurable deployment descriptor model in nfv. **Journal of Network and Systems Management**, Springer, v. 28, p. 693–718, 2020.

ATTO, A. M.; BENOIT, A.; LAMBERT, P. Timed-image based deep learning for action recognition in video sequences. **Pattern Recognition**, v. 104, p. 107353, 2020. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320320301564>.

AWS, A. **Amazon Web Services (AWS)**. 2023. Disponível em: <https://aws.amazon.com/>.

BADRINARAYANAN, V.; KENDALL, A.; CIPOLLA, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 39, n. 12, p. 2481–2495, 2017.

BAPPY, J. H. et al. Hybrid lstm and encoder–decoder architecture for detection of image forgeries. **IEEE Transactions on Image Processing**, v. 28, n. 7, p. 3286–3300, 2019.

BEHRAVESH, R.; CORONADO, E.; RIGGIO, R. Performance evaluation on virtualization technologies for nfv deployment in 5g networks. In: **2019 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2019. p. 24–29.

BELGIU, M.; DRAGUT, L. Random forest in remote sensing: A review of applications and future directions. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 114, p. 24–31, 2016. ISSN 0924-2716. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0924271616000265>.

BENDRISS, J. **Cognitive management of SLA in software-based networks**. Tese (Doutorado) — Evry Institut national des telecommunications, 06 2018.

BESERRA, D. et al. Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. In: **2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2017. p. 363–370.

BEYER, B. et al. Site reliability engineering: How google runs production systems. O'Reilly Media, 2016.

BHARANIDHARAN, G.; JAYALAKSHMI, S.; MAYILVAHANAN, P. Predictive scaling of elastic pod instances for modern applications on public cloud through long short-term memory. In: **2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)**. [S.l.: s.n.], 2022. p. 1710–1717.

BHARDWAJ, A.; KRISHNA, C. R. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. **Arabian Journal for Science and Engineering**, v. 46, n. 9, p. 8585–8601, set. 2021. ISSN 2191-4281. Disponível em: <https://doi.org/10.1007/s13369-021-05553-3>.

BHOLE, V. et al. Machine learning approach for intelligent and sustainable smart healthcare in cloud-centric iot. **International Journal of Intelligent Systems and Applications in Engineering**, v. 11, n. 10s, p. 36–48, ago. 2023. Disponível em: <https://ijisae.org/index.php/IJISAE/article/view/3232>.

BONFIM, M. S.; DIAS, K. L.; FERNANDES, S. F. L. Integrated nfv/sdn architectures: A systematic literature review. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 6, fev. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3172866>.

BONTEMPI, G.; TAIEB, S. B.; BORGNE, Y.-A. L. Machine learning strategies for time series forecasting. In: _____. **Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 62–77. ISBN 978-3-642-36318-4. Disponível em: <"https://doi.org/10.1007/978-3-642-36318-4\%5F3">.

BRAMER, M. Avoiding overfitting of decision trees. In: _____. **Principles of Data Mining**. London: Springer London, 2013. p. 121–136. ISBN 978-1-4471-4884-5. Disponível em: <https://doi.org/10.1007/978-1-4471-4884-5_9>.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, out. 2001. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1010933404324>.

BUI, C. et al. Time series forecasting for healthcare diagnosis and prognostics with the focus on cardiovascular diseases. In: VAN, T. V.; LE, T. A. N.; DUC, T. N. (Ed.). **6th International Conference on the Development of Biomedical Engineering in Vietnam (BME6)**. Singapore: Springer Singapore, 2018. p. 809–818. ISBN 978-981-10-4361-1.

BUYYA, R.; SRIRAMA, S. N. Internet of things (iot) and new computing paradigms. In: _____. **Fog and Edge Computing: Principles and Paradigms**. [S.l.: s.n.], 2019. p. 1–23.

CANONICAL. **LXD**. 2017. Disponível em: <https://github.com/canonical/lxd>.

CAO, K. et al. An overview on edge computing research. **IEEE Access**, v. 8, p. 85714–85728, 2020.

CASALICCHIO, E. Container orchestration: A survey. In: _____. **Systems Modeling: Methodologies and Tools**. Cham: Springer International Publishing, 2019. p. 221–235. ISBN 978-3-319-92378-9. Disponível em: <https://doi.org/10.1007/978-3-319-92378-9\%5F14>.

CASTAN-LASCORZ, M. et al. A new hybrid method for predicting univariate and multivariate time series based on pattern forecasting. **Information Sciences**, v. 586, p. 611–627, 2022. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025521012226>.

CASTRO, P. et al. The rise of serverless computing. **Communications of the ACM**, ACM New York, NY, USA, v. 62, n. 12, p. 44–54, 2019.

CELERY. **Celery**. 2013. Disponível em: <https://docs.celeryq.dev>.

CHAMBON, S. et al. A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, v. 26, n. 4, p. 758–769, 2018.

CHANTRY, M. et al. Opportunities and challenges for machine learning in weather and climate modelling: hard, medium and soft ai. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 379, n. 2194, p. 20200083, fev. 2021. Disponível em: <https://doi.org/10.1098/rsta.2020.0083>.

CHATFIELD, C. **The analysis of time series: An introduction.** [S.l.]: Chapman & Hall/CRC, 2003.

CHAUDHRY, S. R. et al. Improved qos at the edge using serverless computing to deploy virtual network functions. **IEEE Internet of Things Journal**, v. 7, n. 10, p. 10673–10683, 2020.

CHEN, H. et al. Low-dose ct with a residual encoder-decoder convolutional neural network. **IEEE Transactions on Medical Imaging**, v. 36, n. 12, p. 2524–2535, 2017.

CHEN, L.; XIAN, M.; LIU, J. Monitoring system of openstack cloud platform based on prometheus. In: **2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)**. [S.l.: s.n.], 2020. p. 206–209.

CHEN, L.-C. et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: **Proceedings of the European Conference on Computer Vision (ECCV)**. [S.l.: s.n.], 2018.

CHIANG, M. et al. Clarifying fog computing and networking: 10 questions and answers. **IEEE Communications Magazine**, v. 55, n. 4, p. 18–20, 2017.

CHIANG, Y. et al. Management and orchestration of edge computing for iot: A comprehensive survey. **IEEE Internet of Things Journal**, p. 1–1, 2023.

CHO, K. et al. **Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**. 2014.

CHOLLET, F. et al. **Keras**. 2015. Software. Disponível em: <https://keras.io>.

CHOWDHURY, N. M. K.; BOUTABA, R. A survey of network virtualization. **Computer Networks**, Elsevier, v. 54, n. 5, p. 862–876, 2010.

CHUA, L.; ROSKA, T. The cnn paradigm. **IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications**, v. 40, n. 3, p. 147–156, 1993.

CRUZ, P.; ACHIR, N.; VIANA, A. C. On the edge of the deployment: A survey on multi-access edge computing. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 55, n. 5, dez. 2022. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3529758>.

CUNHA, H. **GARNET: An Edge Virtualized Everything Function Management Architecture**. Tese (Doutorado) — Universidade Federal de Uberlandia, 2021. Disponível em: <https://doi.org/10.14393/ufu.di.2021.5578>.

DAHMEN-LHUISSIER. **ETSI - Our group Network Functions Virtualisation (NFV)**. 2021. Https://www.etsi.org/committee/1427-nfv. Accessed in 2023-09-29.

DHAL, P.; AZAD, C. A comprehensive survey on feature selection in the various fields of machine learning. **Applied Intelligence**, v. 52, n. 4, p. 4543–4581, mar. 2022. ISSN 1573-7497. Disponível em: <https://doi.org/10.1007/s10489-021-02550-9>.

DIAO, Y. et al. Self-managing systems: a control theory foundation. In: **12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS05)**. [S.l.: s.n.], 2005. p. 441–448.

DOCS, A. **EC2 predictive auto-scaling**. 2023. Disponível em: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html>.

DOGAN, A.; BIRANT, D. Machine learning and data mining in manufacturing. **Expert Systems with Applications**, v. 166, p. 114060, 2021. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095741742030823X>.

DOKEROGLU, T.; DENIZ, A.; KIZILOZ, H. E. A comprehensive survey on recent metaheuristics for feature selection. **Neurocomputing**, v. 494, p. 269–296, 2022. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092523122200474X>.

DUC, T. L. et al. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 5, set. 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3341145>.

DURAO, F. et al. A systematic review on cloud computing. **The Journal of Supercomputing**, Springer, v. 68, p. 1321–1346, 2014.

DUTTA, S.; TALEB, T.; KSENTINI, A. Qoe-aware elasticity support in cloud-native 5g systems. In: **2016 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2016. p. 1–6.

ETEMADI, M.; GHOBAEI-ARANI, M.; SHAHIDINEJAD, A. A cost-efficient auto-scaling mechanism for iot applications in fog computing environment: a deep learning-based approach. **Cluster Computing**, v. 24, n. 4, p. 3277–3292, dez. 2021. ISSN 1573-7543. Disponível em: <https://doi.org/10.1007/s10586-021-03307-2>.

ETSI. **ETSI Open Source MANO**. 2016. Disponível em: <https://osm.etsi.org/>.

_____. **NBI API - Open Source MANO**. 2016. Disponível em: <https://osm.etsi.org/wikipub/index.php/NBI\%5FAPI\%5FDescription>.

EXPORTER, P. N. **Prometheus Node Exporter**. 2015. Disponível em: <https://github.com/prometheus/node\%5Fexporter>.

FAOUZI, J. Time series classification: A review of algorithms and implementations. In: **Machine Learning (Emerging Trends and Applications)**. [S.l.]: Proud Pen, 2022.

FOUNDATION, O. **OpenStack**. 2010. Disponível em: <https://www.openstack.org/>.

GALICIA, A. et al. Multi-step forecasting for big data time series based on ensemble learning. **Knowledge-Based Systems**, v. 163, p. 830–841, 2019. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705118304957>.

GARIBO-MORANTE, A. A.; TELLEZ, F. O. Univariate and multivariate time series modeling using a harmonic decomposition methodology. **IEEE Latin America Transactions**, v. 20, n. 3, p. 372–378, jul. 2021. Disponível em: <https://latamt.ieeer9.org/index.php/transactions/article/view/5321>.

GIUST, F.; COSTA-PEREZ, X.; REZNIK, A. Multi-access edge computing: An overview of etsi mec isg. In: **IEEE Future Networks Enabling 5G and Beyond**. [S.l.: s.n.], 2017.

GONZALEZ, A. J. et al. Dependability of the nfv orchestrator: State of the art and research challenges. **IEEE Communications Surveys & Tutorials**, v. 20, n. 4, p. 3307–3329, 2018.

GOOGLE. **cAdvisor**. 2023. Disponível em: <https://github.com/google/cadvisor>.

_____. **Google Cloud Plataform**. 2023. Disponível em: <https://cloud.google.com/>.

GRAFANA. **Grafana Monitoring**. 2023. Disponível em: <https://grafana.com/>.

_____. **K6**. 2023. Disponível em: <https://k6.io/>.

GRINBERG, M. **Flask web development: developing web applications with python**. [S.l.]: OReilly Media, Inc., 2018.

HAIBEH, L. A.; YAGOUB, M. C. E.; JARRAY, A. A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches. **IEEE Access**, v. 10, p. 27591–27610, 2022.

_____. A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches. **IEEE Access**, v. 10, p. 27591–27610, 2022.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, v. 53, n. 2, p. 90–97, 2015.

HAN, R. et al. Lightweight resource scaling for cloud applications. In: **2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)**. [S.l.: s.n.], 2012. p. 644–651.

HAPROXY. **HAProxy**. 2023. Disponível em: <https://www.haproxy.org/>.

HAQUE, K. N.; YOUSUF, M. A.; RANA, R. **Image denoising and restoration with CNN-LSTM Encoder Decoder with Direct Attention**. 2018.

HAWILO, H. et al. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). **IEEE Network**, v. 28, n. 6, p. 18–26, 2014.

HEINZE, T. et al. Auto-scaling techniques for elastic data stream processing. In: **Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems**. [S.l.: s.n.], 2014. p. 318–321.

HERRERA, J. G.; BOTERO, J. F. Resource allocation in nfv: A comprehensive survey. **IEEE Transactions on Network and Service Management**, v. 13, n. 3, p. 518–532, 2016.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 11 1997. ISSN 0899-7667. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>.

HOFFMANN, G. A.; TRIVEDI, K. S.; MALEK, M. A best practice guide to resource forecasting for computing systems. **IEEE Transactions on Reliability**, v. 56, n. 4, p. 615–628, 2007.

HOFFMANN, M. et al. Sdn and nfv as enabler for the distributed network cloud. **Mobile Networks and Applications**, v. 23, n. 3, p. 521–528, jun. 2018. ISSN 1572-8153. Disponível em: <https://doi.org/10.1007/s11036-017-0905-y>.

HOROVITZ, S.; ARIAN, Y. Efficient cloud auto-scaling with sla objective using q-learning. In: **2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)**. [S.l.: s.n.], 2018. p. 85–92.

HUANG, A. et al. Low latency mec framework for sdn-based lte/lte-a networks. In: **2017 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2017. p. 1–6.

HUANG, G. et al. Auto scaling virtual machines for web applications with queueing theory. In: **2016 3rd International Conference on Systems and Informatics (ICSAI)**. [S.l.: s.n.], 2016. p. 433–438.

HUANG, X.; ZANNI-MERK, C.; CRéMILLEUX, B. Enhancing deep learning with semantics: an application to manufacturing time series analysis. **Procedia Computer Science**, v. 159, p. 437–446, 2019. ISSN 1877-0509. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050919313808>.

HUANG, Z.; XU, W.; YU, K. **Bidirectional LSTM-CRF Models for Sequence Tagging**. 2015.

IENCO, D. et al. Combining sentinel-1 and sentinel-2 satellite image time series for land cover mapping via a multi-source deep learning architecture. **ISPRS Journal of Photogrammetry and Remote Sensing**, v. 158, p. 11–22, 2019. ISSN 0924-2716. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0924271619302278>.

JAIN, G.; MALLICK, B. A study of time series models arima and ets. jan. 2017. Disponível em: <https://ssrn.com/abstract=2898968>.

JENNINGS, B.; STADLER, R. Resource management in clouds: Survey and research challenges. **Journal of Network and Systems Management**, v. 23, n. 3, p. 567–619, Jul 2015. ISSN 1573-7705. Disponível em: <https://doi.org/10.1007/s10922-014-9307-7>.

JIANG, T.; GRADUS, J. L.; ROSELLINI, A. J. Supervised machine learning: A brief primer. **Behavior Therapy**, v. 51, n. 5, p. 675–687, 2020. ISSN 0005-7894. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0005789420300678>.

JIANG, W. et al. The road towards 6g: A comprehensive survey. **IEEE Open Journal of the Communications Society**, v. 2, p. 334–366, 2021.

JOSHI, M.; HADI, T. H. **A Review of Network Traffic Analysis and Prediction Techniques**. 2015.

KALYVIANAKI, E.; CHARALAMBOUS, T.; HAND, S. Adaptive resource provisioning for virtualized servers using kalman filters. **ACM Transactions on Autonomous and Adaptive Systems**, v. 9, p. 1–35, 07 2014.

KARIMI, F.; DOWLATSHAHI, M. B.; HASHEMI, A. Semiaco: A semi-supervised feature selection based on ant colony optimization. **Expert Systems with Applications**, v. 214, p. 119130, 2023. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417422021480>.

KAUR, K.; MANGAT, V.; KUMAR, K. A review on virtualized infrastructure managers with management and orchestration features in nfv architecture. **Computer Networks**, v. 217, p. 109281, 2022. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128622003395>.

KHAN, M. Y. et al. Automated prediction of good dictionary examples (gdex): A comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques. **Complexity**, 09 2021.

KHAN, W. Z. et al. Edge computing: A survey. **Future Generation Computer Systems**, v. 97, p. 219–235, 2019. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X18319903>.

KIM, H.-G. et al. A deep learning approach to vnf resource prediction using correlation between vnfs. p. 444–449, 2019.

______. Machine learning-based method for prediction of virtual network function resource demands. In: **2019 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2019. p. 405–413.

KIM, T.-Y.; CHO, S.-B. Predicting residential energy consumption using cnn-lstm neural networks. **Energy**, v. 182, p. 72–81, 2019. ISSN 0360-5442. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360544219311223>.

KOMAREK, A. et al. Vnf orchestration and modeling with etsi mano compliant frameworks. In: GALININA, O. et al. (Ed.). **Internet of Things, Smart Spaces, and Next Generation Networks and Systems**. Cham: Springer International Publishing, 2017. p. 121–131. ISBN 978-3-319-67380-6.

KUBERNETES. **Kubernetes**. 2023. Disponível em: <https://cloud.google.com/>.

KUBERNETES-DOCS. **Kubernetes Horizontal Pod Autoscaler (HPA)**. 2023. Disponível em: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale>.

KUKADE, P. P.; KALE, G. Auto-scaling of micro-services using containerization. **International Journal of Science and Research (IJSR)**, v. 4, n. 9, p. 1960–1963, 2015.

KUNDIMANA, G.; VYUKUSENGE, A.; TSYM, A. Networks modernization using sdn and nfv technologies. In: **2021 Systems of Signals Generating and Processing in the Field of on Board Communications**. [S.l.: s.n.], 2021. p. 1–5.

KUNDU, A.; BERTINO, E. Structural signatures for tree data structures. **Proc. VLDB Endow.**, VLDB Endowment, v. 1, n. 1, p. 138–150, ago. 2008. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/1453856.1453876>.

LAL, S. et al. Securing vnf communication in nfvi. In: **2017 IEEE Conference on Standards for Communications and Networking (CSCN)**. [S.l.: s.n.], 2017. p. 187–192.

LARA-BENITEZ, P.; CARRANZA-GARCIA, M.; RIQUELME, J. C. An experimental review on deep learning architectures for time series forecasting. **International Journal of Neural Systems**, World Scientific Pub Co Pte Lt, v. 31, n. 03, p. 2130001, fev. 2021. Disponível em: <https://doi.org/10.1142/s0129065721300011>.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group UK London, v. 521, n. 7553, p. 436–444, 2015.

_____. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, maio 2015. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/nature14539>.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LEE, D.-Y. et al. Deep q-network-based auto scaling for service in a multi-access edge computing environment. **International Journal of Network Management**, Wiley Online Library, v. 31, n. 6, p. e2176, 2021.

LI, D. et al. Power grid load state information perception forecasting technology for battery energy storage system based on elman neural network. In: **2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)**. [S.l.: s.n.], 2019. p. 914–917.

LI, H. et al. Mobile edge computing: Progress and challenges. In: **2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)**. [S.l.: s.n.], 2016. p. 83–84.

LI, J. et al. Feature selection: A data perspective. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 6, dez. 2017. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3136625>.

LI, S.; XU, L. D.; ZHAO, S. 5g internet of things: A survey. **Journal of Industrial Information Integration**, v. 10, p. 1–9, 2018. ISSN 2452-414X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2452414X18300037>.

LIASHCHYNSKYI, P.; LIASHCHYNSKYI, P. Grid search, random search, genetic algorithm: A big comparison for nas. **CoRR**, abs/1912.06059, 2019. Disponível em: <http://arxiv.org/abs/1912.06059>.

LIJUAN, W.; GUOHUA, C. Seasonal svr with foa algorithm for single-step and multi-step ahead forecasting in monthly inbound tourist flow. **Knowledge-Based Systems**, v. 110, p. 157–166, 2016. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705116302398>.

LIM, B.; ZOHREN, S. Time-series forecasting with deep learning: a survey. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 379, n. 2194, p. 20200209, fev. 2021. Disponível em: <https://doi.org/10.1098/rsta.2020.0209>.

LLORENS-CARRODEGUAS, A. et al. An sdn-based solution for horizontal auto-scaling and load balancing of transparent vnf clusters. **Sensors**, v. 21, n. 24, 2021. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/21/24/8283>.

LOH, W.-Y. Classification and regression trees. **Wiley interdisciplinary reviews: data mining and knowledge discovery**, Wiley Online Library, v. 1, n. 1, p. 14–23, 2011.

_____. Fifty years of classification and regression trees. **International Statistical Review**, Wiley Online Library, v. 82, n. 3, p. 329–348, 2014.

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**, v. 12, n. 4, p. 559–592, dez. 2014. ISSN 1572-9184. Disponível em: <https://doi.org/10.1007/s10723-014-9314-7>.

_____. A review of auto-scaling techniques for elastic applications in cloud environments. **Journal of Grid Computing**, v. 12, n. 4, p. 559–592, dez. 2014. ISSN 1572-9184. Disponível em: <https://doi.org/10.1007/s10723-014-9314-7>.

LU, J.-b.; YU, Y.; PAN, M.-l. Reinforcement learning-based auto-scaling algorithm for elastic cloud workflow service. In: SHEN, H. et al. (Ed.). **Parallel and Distributed Computing, Applications and Technologies**. Cham: Springer International Publishing, 2022. p. 303–310.

MA, Y. et al. Virtual network function service provisioning in mec via trading off the usages between computing and communication resources. **IEEE Transactions on Cloud Computing**, v. 10, n. 4, p. 2949–2963, 2022.

MAHARANA, K.; MONDAL, S.; NEMADE, B. A review: Data pre-processing and data augmentation techniques. **Global Transitions Proceedings**, v. 3, n. 1, p. 91–99, 2022. ISSN 2666-285X. International Conference on Intelligent Engineering Approach(ICIEA-2022). Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666285X22000565>.

MANSOURI, Y.; BABAR, M. A. A review of edge computing: Features and resource virtualization. **Journal of Parallel and Distributed Computing**, v. 150, p. 155–183, 2021. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731520304317>.

MAO, M.; HUMPHREY, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: **Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis**. New York, NY, USA: Association for Computing Machinery, 2011. (SC '11). ISBN 9781450307710. Disponível em: <https://doi.org/10.1145/2063384.2063449>.

MAO, Y. et al. A survey on mobile edge computing: The communication perspective. **IEEE Communications Surveys & Tutorials**, v. 19, n. 4, p. 2322–2358, 2017.

MARSDEN, P. V. Survey methods for network data. **The SAGE handbook of social network analysis**, v. 25, p. 370–388, 2011.

MATSUO, Y. et al. Deep learning, reinforcement learning, and world models. **Neural Networks**, v. 152, p. 267–275, 2022. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608022001150>.

MAULUD, D.; ABDULAZEEZ, A. M. A review on linear regression comprehensive in machine learning. **Journal of Applied Science and Technology Trends**, Interdisciplinary Publishing Academia, v. 1, n. 4, p. 140–147, dez. 2020. Disponível em: <https://doi.org/10.38094/jastt1457>.

MECHTRI, M. et al. Nfv orchestration framework addressing sfc challenges. **IEEE Communications Magazine**, v. 55, n. 6, p. 16–23, 2017.

MEHDIYEV, N. et al. Time series classification using deep learning for process planning: A case from the process industry. **Procedia Computer Science**, v. 114, p. 242–249, 2017. ISSN 1877-0509. Complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30 – November 1, 2017, Chicago, Illinois, USA. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050917318707>.

MESTRES, A. et al. Knowledge-defined networking. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 47, n. 3, p. 2–10, set. 2017. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/3138808.3138810>.

MINOVSKI, D.; ÅHLUND, C.; MITRA, K. Modeling quality of iot experience in autonomous vehicles. **IEEE Internet of Things Journal**, v. 7, n. 5, p. 3833–3849, 2020.

Mohamad Noor, M. binti; HASSAN, W. H. Current research on internet of things (iot) security: A survey. **Computer Networks**, v. 148, p. 283–294, 2019. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128618307035>.

MORADI, M.; AHMADI, M.; NIKBAZM, R. Comparison of machine learning techniques for vnf resource requirements prediction in nfv. **Journal of Network and Systems Management**, v. 30, 01 2022.

MUSADDIQ, A. et al. A survey on resource management in iot operating systems. **IEEE Access**, v. 6, p. 8459–8482, 2018.

NADEEM, L. et al. Integration of d2d, network slicing, and mec in 5g cellular networks: Survey and challenges. **IEEE Access**, v. 9, p. 37590–37612, 2021.

NDIKUMANA, A. et al. Joint communication, computation, caching, and control in big data multi-access edge computing. **IEEE Transactions on Mobile Computing**, v. 19, n. 6, p. 1359–1374, 2020.

NETWORKS, M. **Clearwater**. 2016. Accessed: 31/07/2023. Disponível em: <https://clearwater.readthedocs.io>.

NICODEMUS, C. H.; BOERES, C.; REBELLO, V. E. Managing vertical memory elasticity in containers. In: **2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)**. [S.l.: s.n.], 2020. p. 132–142.

NKENYEREYE, L. et al. Virtual iot service slice functions for multiaccess edge computing platform. **IEEE Internet of Things Journal**, v. 8, n. 14, p. 11233–11248, 2021.

OPENSTACK. **Ceilometer - OpenStack**. 2022. Disponível em: <https://github.com/openstack/ceilometer>.

ORDONEZ-LUCENA, J. et al. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. **IEEE Communications Magazine**, v. 55, n. 5, p. 80–87, 2017.

OSM, E. **Open Source MANO**. 2020. Disponível em: <https://osm.etsi.org/>.

______. **OSM Monitoring and Autoscaling**. 2020. Disponível em: <https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html?highlight=auto\#monitoring-and-autoscaling>.

PADALA, P. et al. Automated control of multiple virtualized resources. In: **Proceedings of the 4th ACM European Conference on Computer Systems**. New York, NY, USA: Association for Computing Machinery, 2009. (EuroSys 09), p. 13–26. ISBN 9781605584829. Disponível em: <https://doi.org/10.1145/1519065.1519068>.

PAGANELLI, F. et al. Network service description model for vnf orchestration leveraging intent-based sdn interfaces. In: **2017 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2017. p. 1–5.

PAN, J. et al. **Key Enabling Technologies for Secure and Scalable Future Fog-IoT Architecture: A Survey**. 2018.

PANAGIOTIS, T. et al. Comparison of management and orchestration solutions for the 5g era. **Journal of Sensor and Actuator Networks**, v. 9, p. 4, 01 2020.

PEDREGOSA, F. et al. **scikit-learn: Machine Learning in Python**. 2011. Software. Disponível em: <https://scikit-learn.org>.

PEI, J. et al. Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks. **IEEE Journal on Selected Areas in Communications**, v. 38, n. 2, p. 263–278, 2020.

PETROPOULOS, F. et al. Forecasting: theory and practice. **International Journal of Forecasting**, v. 38, n. 3, p. 705–871, 2022. ISSN 0169-2070. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169207021001758>.

PLAGERAS, A. P. et al. Efficient iot-based sensor big data collection–processing and analysis in smart buildings. **Future Generation Computer Systems**, v. 82, p. 349–357, 2018. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17314127>.

PORAMBAGE, P. et al. Survey on multi-access edge computing for internet of things realization. **IEEE Communications Surveys & Tutorials**, v. 20, n. 4, p. 2961–2991, 2018.

POTDAR, A. M. et al. Performance evaluation of docker container and virtual machine. **Procedia Computer Science**, v. 171, p. 1419–1428, 2020. ISSN 1877-0509. Third International Conference on Computing and Network Communications (CoCoNet'19). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050920311315>.

_____. Performance evaluation of docker container and virtual machine. **Procedia Computer Science**, v. 171, p. 1419–1428, 2020. ISSN 1877-0509. Third International Conference on Computing and Network Communications (CoCoNet19). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050920311315>.

PRIYA, B. et al. Mobile edge communication an overview of mec in 5g. In: **2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)**. [S.l.: s.n.], 2019. p. 271–276.

QI, H.; GANI, A. Research on mobile cloud computing: Review, trend and perspectives. In: **2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)**. [S.l.: s.n.], 2012. p. 195–202.

QU, C.; CALHEIROS, R. N.; BUYYA, R. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. **Journal of Network and Computer Applications**, v. 65, p. 167–180, 2016. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804516300078>.

_____. Auto-scaling web applications in clouds: A taxonomy and survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 4, jul. 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3148149>.

QUESETH, O. et al. **5GPPP Architecture Working Group: View on 5G Architecture**. 2021. Disponível em: <https://5g-ppp.eu/wp-content/uploads/2021/11/Architecture-WP-V4.0-final.pdf>.

RABENSTEIN, B.; VOLZ, J. Prometheus: A next-generation monitoring system (talk). In: . Dublin: USENIX Association, 2015.

RAIHAN, A. S.; AHMED, I. **A Bi-LSTM Autoencoder Framework for Anomaly Detection – A Case Study of a Wind Power Dataset**. 2023.

RAMÍREZ-GALLEGO, S. et al. A survey on data preprocessing for data stream mining: Current status and future directions. **Neurocomputing**, v. 239, p. 39–57, 2017. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231217302631>.

RANDAL, A. The ideal versus the real: Revisiting the history of virtual machines and containers. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 1, fev. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3365199>.

RASHID, K. M.; LOUIS, J. Times-series data augmentation and deep learning for construction equipment activity recognition. **Advanced Engineering Informatics**, v. 42, p. 100944, 2019. ISSN 1474-0346. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474034619300886>.

RAY, P. P.; KUMAR, N. Sdn/nfv architectures for edge-cloud oriented iot: A systematic review. **Computer Communications**, v. 169, p. 129–153, 2021. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366421000396>.

RAY, S. A quick review of machine learning algorithms. In: **2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)**. [S.l.: s.n.], 2019. p. 35–39.

REDIS. **Redis**. 2009. Disponível em: <https://redis.io/>.

REN, Y. et al. Dynamic auto scaling algorithm (dasa) for 5g mobile networks. In: IEEE. **2016 IEEE global communications conference (GLOBECOM)**. [S.l.], 2016. p. 1–6.

RODERO-MERINO, L. et al. From infrastructure delivery to service management in clouds. **Future Generation Computer Systems**, v. 26, n. 8, p. 1226–1240, 2010. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X10000294>.

RONG, S.; BAO-WEN, Z. The research of regression model in machine learning field. **MATEC Web of Conferences**, EDP Sciences, v. 176, p. 01033, 2018. Disponível em: <https://doi.org/10.1051/matecconf/201817601033>.

SABHARWAL, N.; PANDEY, P. **Monitoring Microservices and Containerized Applications**: Deployment, configuration, and best practices for prometheus and alert manager. [S.l.: s.n.], 2020.

SAGIROGLU, S.; SINANC, D. Big data: A review. In: **2013 International Conference on Collaboration Technologies and Systems (CTS)**. [S.l.: s.n.], 2013. p. 42–47.

SAHNI, J.; VIDYARTHI, D. P. Heterogeneity-aware adaptive auto-scaling heuristic for improved qos and resource usage in cloud environments. **Computing**, v. 99, n. 4, p. 351–381, abr. 2017. ISSN 1436-5057. Disponível em: <https://doi.org/10.1007/s00607-016-0530-9>.

SAMEK, W. et al. Explaining deep neural networks and beyond: A review of methods and applications. **Proceedings of the IEEE**, v. 109, n. 3, p. 247–278, 2021.

SCALING, A. A. **AWS Auto Scaling**. 2023. Disponível em: <https://aws.amazon.com/autoscaling/>.

SCALINGI, A. et al. Scalable provisioning of virtual network functions via supervised learning. In: **2019 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2019. p. 423–431.

SCHARDONG, F.; NUNES, I.; SCHAEFFER-FILHO, A. Nfv resource allocation: a systematic review and taxonomy of vnf forwarding graph embedding. **Computer Networks**, v. 185, p. 107726, 2021. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128620313189>.

SCHOLKOPF, B.; SMOLA, A. J. **Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond**. Cambridge, MA: MIT Press, 2001.

SCHUSTER, M.; PALIWAL, K. Bidirectional recurrent neural networks. **Signal Processing, IEEE Transactions on**, v. 45, p. 2673–2681, 12 1997.

SCIANCALEPORE, V. et al. A double-tier mec-nfv architecture: Design and optimisation. In: **2016 IEEE Conference on Standards for Communications and Networking (CSCN)**. [S.l.: s.n.], 2016. p. 1–6.

SERBAN, I. et al. A hierarchical latent variable encoder-decoder model for generating dialogues. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 31, n. 1, fev. 2017. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/10983>.

SEZER, O. B.; GUDELEK, M. U.; OZBAYOGLU, A. M. Financial time series forecasting with deep learning : A systematic literature review: 2005-2019. **CoRR**, abs/1911.13288, 2019. Disponível em: <http://arxiv.org/abs/1911.13288>.

SHAHIN, A. A. Enhancing elasticity of saas applications using queuing theory. **International Journal of Advanced Computer Science and Applications**, The Science and Information Organization, v. 8, n. 1, 2017. Disponível em: <https://doi.org/10.14569\%2Fijacsa.2017.080136>.

SHAO, Z. et al. Modeling and forecasting the electricity clearing price: A novel belm based pattern classification framework and a comparative analytic study on multi-layer belm and lstm. **Energy Economics**, v. 86, p. 104648, 2020. ISSN 0140-9883. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140988319304451>.

SHEWALKAR, A. N. Comparison of rnn, lstm and gru on speech recognition data. North Dakota State University, 2018.

SHEYKHMOUSA, M. et al. Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 13, p. 6308–6325, 2020.

SHI, Z.; ZENG, Y.; WU, Z. Service chain orchestration based on deep reinforcement learning in intent-based iot. Springer, Singapore, p. 875–882, 2020.

SHINDE, P. P.; SHAH, S. A review of machine learning and deep learning applications. In: **2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)**. [S.l.: s.n.], 2018. p. 1–6.

SIAMI-NAMINI, S.; TAVAKOLI, N.; NAMIN, A. S. The performance of lstm and bilstm in forecasting time series. In: **2019 IEEE International Conference on Big Data (Big Data)**. [S.l.: s.n.], 2019. p. 3285–3292.

SIERRA-ARRIAGA, F.; BRANCO, R.; LEE, B. Security issues and challenges for virtualization technologies. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, maio 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3382190>.

SILVA, T. P. da et al. Online machine learning for auto-scaling in the edge computing. **Pervasive and Mobile Computing**, v. 87, p. 101722, 2022. ISSN 1574-1192. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1574119222001353>.

SINGH, P. et al. Research on auto-scaling of web applications in cloud: Survey, trends and future directions. **Scalable Computing: Practice and Experience**, Scalable Computing: Practice and Experience, v. 20, n. 2, p. 399–432, maio 2019. Disponível em: <https://doi.org/10.12694/scpe.v20i2.1537>.

SITTóN-CANDANEDO, I. et al. A review of edge computing reference architectures and a new global edge proposal. **Future Generation Computer Systems**, v. 99, p. 278–294, 2019. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X1930264X>.

SONG, Y. et al. Deep learning-based automatic segmentation of images in cardiac radiography: A promising challenge. **Computer Methods and Programs in Biomedicine**, v. 220, p. 106821, 2022. ISSN 0169-2607. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169260722002036>.

SOOSAI, M. R.; JAGDALE, B. Efficient time based scheduler for implementing reservation plan in cloud. In: **2014 IEEE Global Conference on Wireless Computing & Networking (GCWCN)**. [S.l.: s.n.], 2014. p. 165–168.

SOUSA, N. F. S. de et al. Network service orchestration: A survey. **Computer Communications**, v. 142-143, p. 69–94, 2019. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366418309502>.

SPINELLI, F.; MANCUSO, V. Toward enabled industrial verticals in 5g: A survey on mec-based approaches to provisioning and flexibility. **IEEE Communications Surveys & Tutorials**, v. 23, n. 1, p. 596–630, 2021.

SQLITE. **SQLite**. 2000. Disponível em: <https://www.sqlite.org>.

SRIVASTAVA, P.; KHAN, R. A review paper on cloud computing. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 8, n. 6, p. 17–20, 2018.

SUBRAMANYA, T.; RIGGIO, R. Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond. **IEEE Transactions on Network and Service Management**, v. 18, n. 1, p. 63–78, 2021.

SUKHIJA, N.; BAUTISTA, E. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In: **2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)**. [S.l.: s.n.], 2019. p. 257–262.

TALEB, T. et al. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. **IEEE Communications Surveys & Tutorials**, v. 19, n. 3, p. 1657–1681, 2017.

TAN, L.; WANG, N. Future internet: The internet of things. In: **2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)**. [S.l.: s.n.], 2010. v. 5, p. V5–376–V5–380.

TANG, E.; FAN, Y. Performance comparison between five nosql databases. In: **2016 7th International Conference on Cloud Computing and Big Data (CCBD)**. [S.l.: s.n.], 2016. p. 105–109.

TAO, J. et al. Adaptive vnf scaling approach with proactive traffic prediction in nfv-enabled clouds. In: **ACM Turing Award Celebration Conference - China (ACM TURC 2021)**. New York, NY, USA: Association for Computing Machinery, 2021. (ACM TURC 2021), p. 166–172. ISBN 9781450385671. Disponível em: <https://doi.org/10.1145/3472634.3474066>.

TAO, Z. et al. A survey of virtual machine management in edge computing. **Proceedings of the IEEE**, v. 107, n. 8, p. 1482–1499, 2019.

TASDELEN, A.; SEN, B. A hybrid cnn-lstm model for pre-mirna classification. **Scientific Reports**, v. 11, n. 1, p. 14125, jul. 2021. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-021-93656-0>.

THOMPSON, J. **Predictive Horizontal Pod Autoscaler**. 2019. Disponível em: <https://predictive-horizontal-pod-autoscaler.readthedocs.io/en/latest/>.

TORRES, J. F. et al. Deep learning for time series forecasting: A survey. **Big Data**, Mary Ann Liebert Inc, v. 9, n. 1, p. 3–21, fev. 2021. Disponível em: <https://doi.org/10.1089/big.2020.0159>.

TURNBULL, J. **Monitoring with Prometheus**. [S.l.: s.n.], 2018.

VERMA, S.; BALA, A. Auto-scaling techniques for iot-based cloud applications: a review. **Cluster Computing**, Springer, v. 24, n. 3, p. 2425–2459, 2021.

VINAY, K.; KUMAR, S. M. D. Auto-scaling for deadline constrained scientific workflows in cloud environment. In: **2016 IEEE Annual India Conference (INDICON)**. [S.l.: s.n.], 2016. p. 1–6.

VINHAL, L. **Codebase - Algorithms and Hyperparameters**. [S.l.]: GitHub, 2023. Https://github.com/vinhal/ai-timeseries-forecasting-compariosn.

VOSOUGHI, S.; VIJAYARAGHAVAN, P.; ROY, D. Tweet2vec: Learning tweet embeddings using character-level cnn-lstm encoder-decoder. In: **Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval**. New York, NY, USA: Association for Computing Machinery, 2016. (SIGIR '16), p. 1041–1044. ISBN 9781450340694. Disponível em: <https://doi.org/10.1145/2911451.2914762>.

VU, X. T. et al. An architecture for enabling vnf auto-scaling with flow migration. In: **2020 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2020. p. 624–627.

WANG, J. et al. Analysis and application of forecasting models in wind power integration: A review of multi-step-ahead wind speed forecasting models. **Renewable and Sustainable Energy Reviews**, v. 60, p. 960–981, 2016. ISSN 1364-0321. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364032116001441>.

WANG, S. et al. Degradation evaluation of slewing bearing using hmm and improved gru. **Measurement**, v. 146, p. 385–395, 2019. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224119306049>.

______. Uav photogrammetry and afsa-elman neural network in slopes displacement monitoring and forecasting. **KSCE Journal of Civil Engineering**, Springer Science and Business Media LLC, v. 24, n. 1, p. 19–29, dez. 2019. Disponível em: <https://doi.org/10.1007/s12205-020-1697-3>.

WANG, Z.; SU, X.; DING, Z. Long-term traffic prediction based on lstm encoder-decoder architecture. **IEEE Transactions on Intelligent Transportation Systems**, v. 22, n. 10, p. 6561–6571, 2021.

WU, F. et al. Edge-based hybrid system implementation for long-range safety and healthcare iot applications. **IEEE Internet of Things Journal**, v. 8, n. 12, p. 9970–9980, 2021.

WU, W. et al. Using gated recurrent unit network to forecast short-term load considering impact of electricity price. **Energy Procedia**, v. 158, p. 3369–3374, 2019. ISSN 1876-6102. Innovative Solutions for Energy Transitions. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1876610219309981>.

WU, Y. et al. Scaling social media applications into geo-distributed clouds. **IEEE/ACM Transactions on Networking**, v. 23, n. 3, p. 689–702, 2015.

XIANG, J. et al. Multi-time scale wind speed prediction based on wt-bi-lstm. **MATEC Web of Conferences**, v. 309, p. 05011, 01 2020.

XIE, M. et al. Service assurance architecture in nfv. In: **2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2017. p. 229–235.

XU, H. et al. An improved cnn-lstm model-based state-of-health estimation approach for lithium-ion batteries. **Energy**, v. 276, p. 127585, 2023. ISSN 0360-5442. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360544223009799>.

YADUVANSHI, D.; SHARMA, A.; MORE, P. V. Application of queuing theory to optimize waiting-time in hospital operations. **Operations and Supply Chain Management: An International Journal**, OSCM Forum, v. 12, n. 3, p. 165–174, 2019.

YAHIA, I. B. **VNFDataset: Virtual IP multimedia IP System**. 2020. Disponível em: <https://www.kaggle.com/datasets/imenbenyahia/clearwatervnf-virtual-ip-multimedia-ip-system>.

YALA, L.; FRANGOUDIS, P. A.; KSENTINI, A. Latency and availability driven vnf placement in a mec-nfv environment. In: **2018 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2018. p. 1–7.

YAN, H.; OUYANG, H. Financial time series prediction based on deep learning. **Wireless Personal Communications**, Springer Science and Business Media LLC, v. 102, n. 2, p. 683–700, dez. 2017. Disponível em: <https://doi.org/10.1007/s11277-017-5086-2>.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, v. 415, p. 295–316, 2020. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231220311693>.

YU, H.; YANG, J.; FUNG, C. Fine-grained cloud resource provisioning for virtual network function. **IEEE Transactions on Network and Service Management**, v. 17, n. 3, p. 1363–1376, 2020.

YU, W. et al. A survey on the edge computing for the internet of things. **IEEE Access**, v. 6, p. 6900–6919, 2018.

ZAFAR, S. et al. Framework for efficient auto-scaling of virtual network functions in a cloud environment. **Sensors**, v. 22, n. 19, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/19/7597>.

ZAMAN, Z.; RAHMAN, S.; NAZNIN, M. Novel approaches for vnf requirement prediction using dnn and lstm. In: **2019 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2019. p. 1–6.

ZEROUAL, A. et al. Deep learning methods for forecasting covid-19 time-series data: A comparative study. **Chaos, Solitons & Fractals**, v. 140, p. 110121, 2020. ISSN 0960-0779. Disponível em: <https://www.sciencedirect.com/science/article/pii/S096007792030518X>.

ZHANG, F.; O'DONNELL, L. J. Chapter 7 - support vector regression. In: MECHELLI, A.; VIEIRA, S. (Ed.). **Machine Learning**. Academic Press, 2020. p. 123–140. ISBN 978-0-12-815739-8. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128157398000079>.

ZHANG, G.; QI, M. Neural network forecasting for seasonal and trend time series. **European Journal of Operational Research**, v. 160, n. 2, p. 501–514, 2005. ISSN 0377-2217. Decision Support Systems in the Internet Age. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221703005484>.

ZHANG, Q. et al. A comparative study of containers and virtual machines in big data environment. In: **2018 IEEE 11th International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2018. p. 178–185.

ZHANG, W. Y. et al. Single-step and multi-step time series prediction for urban temperature based on lstm model of tensorflow. In: **2021 Photonics & Electromagnetics Research Symposium (PIERS)**. [S.l.: s.n.], 2021. p. 1531–1535.

ZHOU, J.; HUA, Z. A correlation guided genetic algorithm and its application to feature selection. **Applied Soft Computing**, v. 123, p. 108964, 2022. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494622003052>.

ZHOU, Z.-H. **Machine learning**. [S.l.]: Springer Nature, 2021.